OSBORNE/ McGRAW-HILL

TURBO PROLOG

GUIA DO USUÁRIO

McGRAW-HILL

PHILLIP R. ROBINSON

TURBO PROLOG

Guia do Usuário

TURBO PROLOG

Guia do Usuário

Phillip R. Robinson

Tradução
Lars Gustav Erik Unonius
Engenheiro Eletrônico

Revisão Técnica
Geraldo Coen
Software Básico, Consultoria, Projetos
Rua Francisco Dias Velho, 104
São Paulo

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • *Lisboa* • *Porto* • *Bogotá* • *Buenos Aires* • *Guatemala* • *Madrid* • *México* • *New York* • *Panamá* • *San Juan* • *Santiago*

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sidney • Tokyo • Toronto

Do orginal
Using Turbo Prolog

Editor: Milton Mira de Assumpção Filho
Coordenadora de Revisão: Daisy Pereira Daniel
Supervisor de Produção: José Rodrigues
Capa: layout: Cyro Giordano

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Robinson, Phillip R.
R556t Turbo Prolog : guia do usuário / Phillip R. Robinson ; tradução Lars Gustav Erik Unonius ; revisão técnica Geraldo Coen. -- São Paulo : McGraw-Hill, 1988.

Bibliografia.

1. Turbo Prolog (Linguagem de programação para computadores) I. Título.

87-2318 CDD-001.6424

Índices para catálogo sistemático:

1. Turbo Prolog : Linguagem de programação : Processamento de dados 001.6424

AGRADECIMENTOS

É natural que eu agradeça às pessoas que ajudaram na escrita deste livro. Aparecem de duas formas. John Erickson de Osborne/McGraw-Hill e Brenda McLaughlin da revista "BYTE" mantiveram o mundo exterior afastados de modo que eu pudesse escrever um livro enquanto cuidava de uma filha com oito meses e trabalhava em um periódico. Robin Shepard Tygh, Phillipe Kahn e Dan Kerman da Borland International apresentaram-me o Turbo Prolog, mantiveram-me atualizado com a última versão, e responderam às minhas perguntas técnicas a respeito de seus trabalhos.

SUMÁRIO

PREFÁCIO

Estou realmente muito emocionado a respeito do novo livro de Phil, e tenho um prazer imenso em recomendá-lo aos interessados em Turbo Prolog. O interesse crescente por Inteligência Artificial (IA) e pelas linguagens de quinta geração, incluindo Turbo Prolog, é uma característica da segunda metade da década de 80.

Novas linguagens de programação e novas implementações destas linguagens têm sido sempre o estímulo para o desenvolvimento de novas e excitantes aplicações em áreas anteriormente não tocadas por computadores e tecnologia de computação. Estes novos desenvolvimentos serão particularmente interessantes na medida em que virmos mais implementações de tecnologia de IA em áreas como desenvolvimento de sistemas especialistas e interfaces de linguagens naturais, que não são apenas representativas dos avanços na programação de software, mas que também se processam no sentido de tornar os computadores e a sua tecnologia na área de armazenamento de conhecimento disponíveis a todos.

Quando introduzimos Turbo Prolog, tornamos as ferramentas para desenvolvimento de aplicativos de IA disponíveis para todos programadores, e futuros programadores, interessados, fornecendo-lhes um ambiente de desenvolvimento bastante avançado mas fácil de ser utilizado e barato. Nós ainda otimizamos a linguagem Prolog no que diz respeito a velocidade e eficiência. O nosso manual de Turbo Prolog dá os primeiros passos no sentido de apresentar-lhe a linguagem e fornecer-lhe informações de referência para as suas diversas funções e características.

O livro de Phil foi desenvolvido para dar-lhe os passos seguintes. Vai além de nossa introdução de "como" iniciar a descrição e elaboração de exemplos da utilização do

Turbo Prolog. Entra também nas diversas características da linguagem de como e onde utilizá-las.

Conheci Phil há muitos anos, e tenho muito respeito pela suas capacidades de escritor e repórter. Durante a sua carreira na revista "BYTE", ele cobriu o crescimento de toda a indústria de microcomputação. A sua compreensão da indústria do computador pessoal é das melhores que existem.

Sei que você vai gostar deste livro, e combinando a informação existente em *Turbo Prolog – Guia do Usuário* com nosso manual e compilador Turbo Prolog, tenho certeza de que começará a desenvolver novos aplicativos úteis de software. Estamos na expectativa de conhecer seus programas de Turbo Prolog e as suas contribuições pessoais ao "maravilhoso mundo novo da inteligência artificial".

Philippe Kahn
Presidente
Borland International, Inc.

PREFÁCIO

Um conhecimento de Prolog é importante para qualquer pessoa envolvida com computadores porque esta linguagem se tornou o principal ambiente de programação da Inteligência Artificial (IA), uma das principais áreas de aplicação emergente para computadores. A IA está intimamente ligada a algumas das áreas mais excitantes da ciência de computação: processadores de quinta geração, sistemas especialistas e processamento da linguagem natural.

Até recentemente, porém, a maioria das implementações do Prolog era projetada para minicomputadores e estações de trabalho, tornando a linguagem acessível apenas aos poucos programadores que tinham acesso a este tipo de hardware de computação. Uma vez que o Turbo Prolog é projetado para ser utilizado com computadores pessoais como o PC da IBM (e compatíveis com o PC), o poder e a excitação de aplicativos de IA está disponível para praticamente qualquer pessoa. (E por menos de $ 100, Turbo Prolog é também acessível a praticamente todos.)

O pacote do Turbo Prolog – que inclui um compilador, um ambiente de desenvolvimento, programas-exemplos, um manual e o código-fonte de um aplicativo – fornece um completo ambiente de programação. Apesar destas ferramentas serem poderosas, Turbo Prolog é fácil de ser utilizado, mesmo que você seja um iniciante em programação. O ambiente de janelas permite que você liste, edite, compile, processe e depure os programas rápida e eficientemente. E assim que começar a escrever programas em Turbo Prolog, verá que é um dos Prolog mais rápidos disponíveis. Testes padrões mostram que em LIPS (Interferências Lógicas Por Segundo, a maneira padrão para medir a velocidade de pensar do compilador Prolog), o Turbo Prolog executa programas com uma rapidez muito maior do que a maioria dos compiladores Prolog existentes para o PC. De fato, Turbo Prolog compete favoravelmente com alguns Prolog processados em minicomputadores.

A empresa de software responsável pelo Turbo Prolog, Borland International, é uma das empresas de linguagem de computação mais bem sucedidas em toda história. O Turbo Pascal da Borland tornou-se um padrão, não apenas pelo número de cópias vendidas, mas também devido à reputação do programa no que diz respeito a confiabilidade, potência, facilidade de utilização e valor. Estou prevendo que Turbo Prolog vai estabelecer um novo padrão na arena do Prolog, da mesma forma que Turbo Pascal estabeleceu novos padrões ao Pascal.

Ao projetar Turbo Prolog, Borland tomou a decisão de incluir a maioria das convenções Prolog padrão, deixar algumas de fora, e acrescentar muitas características poderosas e que são únicas ao Turbo Prolog. (Algumas das características adicionais incluem velocidade, gráficos e suporte para som, e uma sintaxe simplificada.) Quando mais você utilizar Turbo Prolog, mais verá como atende a suas necessidades de programação para muitas aplicações práticas e fascinantes.

A RESPEITO DESTE LIVRO

Turbo Prolog – Guia do Usuário é uma introdução à linguagem de programação Prolog em geral e ao Turbo Prolog em particular. Este livro vai levá-lo através do Turbo Prolog desde o primeiro momento em que colocar em seu acionador de disco o disquete Sistema/Programa. Depois de aprender como carregar o programa, será apresentado ao ambiente de desenvolvimento (as janelas e menus) e ao Editor onde você realmente começa a digitar seus programas. Depois você continuará passo a passo através das operações dos predicados padrões de Prolog e Turbo Prolog. Cada capítulo utiliza programas demonstrativos para ilustrar o trabalho dos predicados na tela, de modo a ver como os programas funcionam em cada estágio de seu desenvolvimento. Estes programas vão dos predicados mais simples aos aplicativos mais avançados como sons, gráficos e avançados controles de compilação. Alguns dos itens de GeoBase, o grande aplicativo escrito em Turbo Prolog que Borland fornece no disco de Biblioteca/Exemplo, também serão descritos.

Apesar de poder aprender os fundamentos da programação Prolog simplesmente com a leitura de *Turbo Prolog – Guia do Usuário,* obterá um conhecimento melhor de como programar em Turbo Prolog, trabalhando os exemplos e discussões deste livro. Uma das maneiras mais eficientes de assimilar o processo lógico do Prolog é percorrer os programas.

Caso você tenha Turbo Prolog, poderá questionar porque deve se procupar com outro livro quando Borland lhe envia um manual combinado com a introdução e as refe-

rências juntamente com os discos. Por um motivo, este livro oferece-lhe um ponto de vista diferente e uma voz diferente. Para a maioria dos usuários de computadores, aprender Prolog não é tão fácil como aprender outra linguagem de programação. A maioria dos proprietários de PC, por exemplo, foi exposta a alguma versão de BASIC ou Pascal. São linguagens procedurais, e apesar do Prolog ter alguns elementos procedurais, é muito diferente pelo fato de ser bastante declarativa, uma diferença que exige uma maneira diversa de pensar por parte do programador. Conseqüentemente, é muito provável que você precisará de muitos recursos para ajudá-lo a abordar este novo desafio em computação – relações humanas.

Outro bom motivo por que deve colocar *Turbo Prolog – Guia do Usuário* em sua mesa junto com outros manuais é que este livro utiliza muitas telas e programas-exemplo para explicar conceitos abstratos. Você não precisa se preocupar com o que acontece na tela ou tentar criar uma imagem mental: está na página à sua frente. Isto deve ajudá-lo no ambiente de desenvolvimento. Este livro utiliza uma variedade de colocação de janelas e configurações para lembrá-lo de que o ambiente de desenvolvimento é maleável e de que você pode fazer o ambiente atender às suas necessidades. A sua tela pode não ser idêntica àquela mostrada no livro, especialmente com alterações na configuração padrão, *default*, mas as telas devem dar-lhe uma idéia da maneira com que Turbo é e trabalha.

Além disto, os programas-exemplos deste livro são explicados em detalhes. Se você tentou aprender uma linguagem de programação antes, sabe que quanto mais exemplos olhar – e tentar modificar – melhor conhecerá a linguagem.

Conforme você ler *Turbo Prolog – Guia do Usuário*, verá que o manual de usuário do Turbo Prolog é uma fonte de referência excelente. Os programas-exemplos encontrados no manual são magníficos, e os detalhes técnicos fornecidos terão valor incalculável na medida em que começar a escrever os seus próprios programas sofisticados. Se você não tivesse comprado Turbo Prolog mas copiado de um outro disco, deveria comprar imediatamente o compilador e o manual do usuário. O manual não é apenas muito valioso, mas nenhuma das duas razões normalmente oferecidas para justificar a cópia de software (ou pirataria) – custo e instalação do disco rígido – é válida no caso do Turbo Prolog. Pagar menos do que $ 100 por uma linguagem completa com documentação, um editor de janela, diversos programas-exemplos em disco, um aplicativo de amostra e um suporte técnico é difícil. Além disto, você não deve ter nenhum problema na instalação do Turbo Prolog em seu disco rígido ou em fazer o número de duplicações que desejar, porque o programa não tem proteção contra cópia.

A primeira edição do *Turbo Prolog* foi referenciada como versão 1.0. Como com

Turbo Pascal (ou qualquer outro pacote de software) versões subseqüentes e atualizações são sempre editadas. Estas versões são tipicamente denominadas de versão 1.02, versão 1.1, versão 2.0 e assim por diante. Caso esteja utilizando a versão 1.0 do Turbo Prolog, muitas das telas deste livro não serão idênticas àquelas em sua tela, uma vez que *Turbo Prolog – Guia do Usuário* não está baseado na versão 1.0 e 1.02. Virtualmente, todas as alterações feitas na versão 1.02 foram alterações à interface do usuário, que é o que você vê na tela. (Poderá haver, por exemplo, uma diferença na linha de status da janela Editor ou uma seleção adicional no menu Opções.) Existirão poucas, se houver, alterações operacionais. Borland poderá não lançar a versão 1.02 para o público em geral. A empresa poderá optar por esperar até que a versão 1.1 esteja pronta para ser liberada. Todos os programas deste livro operam na versão 1.0 assim como lançamentos futuros de novas versões do programa Turbo Prolog. Onde é aplicável, o texto descreve as diferenças entre a versão 1.0 e liberações posteriores. Adicionalmente, a versão 1.0 se referia aos dois discos Turbo Prolog como o "disco de programa" e o "disco de Utilitários e Programa Amostra". Estes nomes foram alterados nas versões posteriores para "disco Sistema/Programa" e "disco Biblioteca/Exemplo", respectivamente, e são assim referidos neste livro.

INTRODUÇÃO

Prolog é um tópico excitante em todos os níveis de computação porque a Inteligência Artificial (IA) é um tópico excitante e Prolog é uma das linguagens de programação mais avançadas utilizadas pelos pesquisadores de IA. Para os programadores que estão investigando IA, Prolog oferece uma aproximação do software diferente da empregada pelas linguagens mais familiares, como BASIC, COBOL, Pascal e C – um enfoque que se adapta muito melhor a alguns dos problemas que os pesquisadores de IA estão tentando resolver.

Caso você tenha sido atraído pelo Turbo Prolog devido a sua associação com IA, é importante compreender que aquela linguagem não transforma seu PC em um cérebro, nem lhe oferece caminhos imediatos a aplicações superpoderosas. Em vez disto, permite que você aprenda como projetar e aplicar alguns métodos que foram primeiramente desenvolvidos por pesquisadores de IA. Na medida em que você passar por este livro e verificar como o Turbo Prolog encontra soluções às metas, observará como o computador pode ser levado a executar alguns dos trabalhos do programador e conseguirá vislumbrar como estes métodos podem levar um computador a se comportar quase como um ser humano.

UMA CURTA HISTÓRIA DO PROLOG

Prolog é representativo de *"PRO*gramming *LOG*ic" (Lógica Programada) e é uma linguagem de computação inventada por volta de 1970 por Alain Colmerauer e seus colegas na Universidade de Marseille. Prolog tornou-se rapidamente o líder das linguagens de IA na Europa enquanto LISP (outra linguagem de programação utilizada pelos pesquisadores de IA) foi utilizado inicialmente por programadores nos Estados Unidos. Em fins de 1970, começaram a aparecer versões do Prolog para microcomputadores. Um dos

compiladores Prolog mais populares de microcomputação foi micro-Prolog, e muitos livros sobre Prolog são devotados a ele.

Mas o micro-Prolog não oferece a mesma quantidade de predicados que uma linguagem como Turbo Prolog. É ainda muito mais lento do que o Turbo Prolog, uma vez que um programa micro-Prolog leva muito mais tempo para tomar um decisão lógica que o Turbo Prolog.

Não havia muito interesse no Prolog até o momento em que os cientistas em computação japoneses lançaram seu projeto de quinta geração, com a finalidade de desenvolver novos computadores com software que reinaria supremo na década de 90 e além. Não foi uma simples coincidência a escolha que fizeram da linguagem Prolog para se basear. Repentinamente, as pessoas passaram a olhar o Prolog e suas possibilidades de outra forma.

PARA QUE SERVE O PROLOG?

Linguagens de computação específicas raramente são boas para todos os tipos de problemas. FORTRAN (significando "*FOR*mula *TRAN*slation" – tradução de Fórmulas), por exemplo, é utilizado principalmente por cientistas e matemáticos, ao passo que COBOL ("*CO*mmon *B*usiness *O*riented *L*anguage" – Linguagem comum orientada para o comércio) é utilizado principalmente no mundo dos negócios. Muitas implementações de Prolog não têm a habilidade de tratar problemas "numericamente complicados" ou "processamento de texto"; em vez disto, Prolog foi projetado para cuidar de "problemas lógicos" (isto é, problemas para os quais uma decisão tem que ser tomada de uma forma ordenada). Prolog tenta fazer com que o computador "raciocine" ao encontro de uma solução. É especialmente bem adequado para diversos problemas típicos de inteligência artificial. Entre estes, os dois de maior importância são sistemas especialistas e *processamento de linguagem-natural.*

SISTEMAS ESPECIALISTAS

Sistemas especialistas são programas que imitam uma habilidade humana. Contêm informações (isto é, um banco de dados) e uma ferramenta para a compreensão de informação e o encontro de respostas corretas àquelas questões no banco de dados (isto é, uma *máquina de dedução*). Aplicações típicas para sistemas especialistas incluem a medicina (onde o computador é solicitado para diagnosticar doenças) e a geologia (na qual o computador é solicitado onde pode ser encontrado petróleo).

Turbo Prolog tem uma estruturação interna para a criação de banco de dados e tem uma máquina dedutiva pronta-para-ser-processada. A única coisa que você precisa fazer é informar ao programa as regras que deve seguir, e ele descobre seu próprio caminho para a informação apropriada. Muitas outras linguagens de programação exigiriam que você escrevesse as regras, as explicasse, especificasse os percursos, e de uma forma geral trabalhasse em um nível de detalhes muito maior para criar a máquina dedutiva antes de começar a obter a informação necessária. Ainda, o banco de dados do Turbo Prolog é escrito com as mesmas estruturas utilizadas para escrever as regras – aprenda um fato sobre o problema e saberá como tratar com ambos.

PROCESSAMENTO DE LINGUAGEM-NATURAL

O processamento de linguagem-natural é a técnica para levar o computador a compreender as linguagens humanas. Os cientistas que estudam o processamento de linguagem-natural esperam criar hardware e software que lhes permitam digitar **"Levar arquivos de Turbo Prolog ao diretório prolog"** e deixar o computador seguir suas instruções. Hoje, você tem de utilizar um comando do tipo **b: copy a: *. pro \ prolog [v]** para atender a esta finalidade. O Turbo Prolog pode utilizar a idéia de um banco de dados e uma máquina dedutiva para dividir a linguagem humana em partes diferentes e diferentes relações e assim tentar "compreender" o significado.

LÓGICA

Como o nome implica, Prolog depende da manipulação lógica. Como uma linguagem de programação, não é a única: existem outras linguagens no campo geral da "programação lógica". Tais linguagens operam com *lógica de proposição,* conhecida também por *lógica de predicado* ou *cálculo de proposição*. Por exemplos, as seguintes declarações são fatos:

Pessoas com tênis para baseball jogam baseball.
A pessoa#1 tem tênis para baseball.

O que pode ser deduzido, determinado, calculado, computado e determinado destes fatos é que a pessoa#1 joga baseball.

Turbo Prolog faz com que o computador cuide da parte dedutiva. De fato, Turbo Prolog tem uma máquina de dedução interna que automaticamente percorre os fatos e constrói ou testa conclusões lógicas. O problema dado pode parecer trivial, mas se você tivesse um banco de dados técnico, carregado com milhares de fatos e regras, não seria prático passar esta lista para uma pessoa e solicitar uma resposta rápida. (No processamento

de linguagem natural, o banco de dados conteria regras a respeito da análise gramatical e compreensão da linguagem.)

RESOLUÇÃO E UNIFICAÇÃO

Regras lógicas matemáticas que podem deduzir declarações textuais a uma representação simbólica existem há muito tempo. Mas em 1965, J. Alan Robinson descobriu o *princípio da resolução,* que mostrava como estas representações poderiam ser moldadas na forma adequada e dadas a um computador para análise.

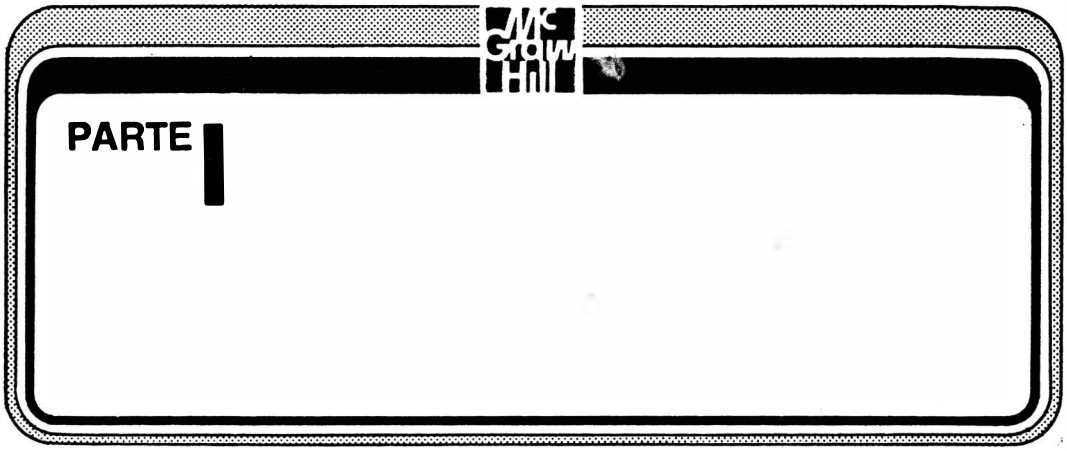
Resolução é uma regra de dedução – permite ao computador dizer que proposições seguem de uma forma lógica de outras proposições. O software que emprega o princípio da resolução opera com "cláusulas" lógicas (você as verá no decorrer do livro). Utiliza *unificação* na tentativa de combinar os lados direito e esquerdo das cláusulas de uma forma lógica pela investigação dos valores variáveis que permitem uma concordância adequada. Caso você queira conhecer mais a respeito das bases teóricas do Prolog e a respeito de resolução e unificação, o livro *Programming in Prolog*, de W. F. Clocksin e C. S. Mellish, é uma excelente fonte de informações (veja a bibliografia no Apêndice B). Enquanto existem muitos livros bons sobre lógica e computação, o Capítulo 10 no livro de Clocksin e Mellish realiza um trabalho muito bom, no estilo Prolog para descrever a lógica de predicados.

LINGUAGEM PROCEDURAIS "VERSUS" LINGUAGENS DECLARATIVAS

Uma vez que Prolog é uma linguagem *declarativa,* ela exige que você declare regras e fatos a respeito de determinados símbolos, para depois perguntar-lhe se uma determinada meta segue de uma forma lógica destas regras e fatos. A máquina dedutiva que faz esta verificação é parte do próprio Prolog. Muitas das características únicas do Turbo Prolog envolvem interações de procedimentos com *o núcleo* declarativo do Prolog. Permitem que você, por exemplo, manipule logicamente características como gráficos e sons que não são tipicamente parte do Prolog ou programação lógica.

Pascal, BASIC e outras linguagens de programação são linguagens *procedurais* quanto a sua natureza. Não é necessário apenas declarar fatos e regras; é preciso ainda utilizar a linguagem para informar ao compilador como procurar uma solução, onde olhar, quando parar, e assim em diante. Poucas linhas de uma linguagem declarativa podem fazer tanto trabalho como muitas linhas de uma linguagem procedural.

A distinção entre linguagens procedurais e linguagens declarativas é uma das razões pela qual uma implementação de linguagem como Turbo Prolog é uma ferramenta tão boa para aplicações IA, especialmente quando comparada com outras linguagens.



A Parte I fornece-lhe a introdução básica ao Turbo Prolog, focalizando as ferramentas que você vai precisar para iniciar a programação e as características únicas do ambiente do Turbo Prolog. Esta seção descreve também o poderoso Editor Turbo Prolog que lhe permite escrever e depurar programas rápida e eficientemente.

CAPÍTULO 1

INSTALAÇÃO

Este capítulo fornece os detalhes a respeito do sistema que você vai precisar para o processamento do Turbo Prolog e informa como fazer discos de trabalho a partir dos originais. Ele ensina o que fazer e a quem pedir ajuda se as coisas não estiverem correndo bem. Um programa poderá não funcionar como se deseja por diversos motivos, e, ao contrário das reclamações dos fabricantes de computadores, as razões são freqüentemente problemas com os computadores e não devido a erros humanos. Caso não tenha o sistema correto (tanto software como hardware) Turbo Prolog não funciona, não importa quantas tentativas se façam.

Se você sabe como duplicar seus discos, que seu sistema processa Turbo Prolog, e como obter ajuda técnica de um fornecedor de software, ignore este capítulo e siga para o Capítulo 2.

O QUE VOCÊ PRECISA PARA PROCESSAR TURBO PROLOG

Caso você tenha um PC-IBM funcionando, provavelmente tem um sistema que possa processar Turbo Prolog. A única exigência difícil é que o seu PC tem que ter ao menos 384k de RAM. Enquanto a maioria das máquinas comerciais agora tem 512k ou mais, ainda existem diversos PCs que têm 256k ou menos. Caso você tenha um sistema deste tipo, considere a ampliação da RAM pra 512k ou um 640k completos. Não se preocupe, não importa que software finalmente vai utilizar, e você não precisará gastar muito para obter a memória adicional.

HARDWARE DO SISTEMA

Serão necessários um IBM-PC ou AT, portátil, 3270 PC ou PCjr para processar

e Turbo Prolog. Você poderá, ainda, utilizar qualquer máquina que *realmente seja compatível* com um dos sistemas mencionados. Mesmo os sistemas perfeitamente compatíveis – isto é, aqueles que processam todo software escrito para os sistemas IBM e que podem ser ligados diretamente a qualquer periférico ou placa feitos para sistemas IBM – podem ainda diferir de alguma forma da arquitetura IBM.

Pelo fato de existirem diversos níveis de compatibilidade, alguns sistemas processam a maior parte do software projetado para máquinas IBM, e alguns processarão todo software IBM. Algumas máquinas compatíveis (ou *clones*) podem ser ligadas a qualquer periférico construído para as máquinas IBM, e alguns podem trabalhar apenas com dispositivos específicos.

Se tiver um compatível completo, prossiga para a próxima seção deste capítulo. Caso não tenha certeza disto, peça de seu vendedor uma garantia de compatibilidade antes de fazer a compra (assim, você poderá devolvê-lo se o Turbo Prolog não funcionar nele), ou verifique análises publicadas nas revistas sobre computadores.

SOFTWARE DO SISTEMA

Você precisa ter um DOS 2.0 ou mais avançado. PC-DOS, o sistema operacional para disco mais popular para os PCs da IBM, evoluiu junto com o hardware do PC. Cada versão substancialmente nova do DOS recebe um novo número de versão: a 1.0 foi seguida pela ligeiramente melhorada 1.1; a bastante melhorada, 2.0 foi seguida pelas 2.1, 3.0, 3.1 e assim por diante. Algumas máquinas européias oferecem DOS 4.0. Microsoft, o criador do DOS, diz estar planejando a introdução do DOS 5.0.

A maioria dos usuários de IBM PC tem a versão DOS 2.0 ou uma mais nova. Quando as revisões são feitas, novas máquinas são vendidas com a nova versão do DOS. DOS 2.0 tem muitas características melhoradas com relação à versão DOS 1.0 e 1.1, e você deveria consegui-lo mesmo que não venha a operar muito com Turbo Prolog.

Turbo Prolog não funciona com DOS 1.0 ou 1.1. Funcionará com qualquer DOS de número maior ou igual a 2.0 (2.0, 2.1, 3.0, 3.1, 3.2 e assim por diante). Cada revisão do DOS é escrita de tal forma que todos os programas escritos em uma versão anterior continuem funcionando em uma versão posterior. Entretanto, um software que exige uma versão posterior do DOS algumas vezes utilizará características e programas utilitários que não existem em versões anteriores. Caso você tenha um compatível, o seu DOS poderá algumas vezes ser denominado MS-DOS (o nome genérico), Z-DOS ou X-DOS. Caso sua máquina tenha compatibilidade IBM, estes sistemas operacionais deverão operar

com Turbo Prolog. Ainda, mais: você precisará da versão 2.0 ou versão mais recente para operar Turbo Prolog.

MEMÓRIA

Os programas que você processa em seu computador são mantidos em memória RAM. Os projetistas de linguagem da Borland estão orgulhosos de fazerem um Prolog que processa em apenas 384k de RAM. Enquanto muitos IBM-PCs e compatíveis agora estão sendo vendidos com 512k de RAM, o IBM-PC típico deverá ter 256k. Muitos sistemas anteriores tinham apenas 64k. A maneira mais comum de acrescentar memória ao PC é encaixar uma placa adicional de circuito impresso no conector interno à unidade central de processamento. Os integrados de memória neste cartão adicional são somados à memória que já existe no sistema. Estes cartões podem conter 256k, 384k ou mais. Caso você tenha mais RAM, seu Turbo Prolog poderá realizar mais trabalho. A quantidade máxima de RAM que uma máquina PC da IBM poderá utilizar sem esquemas especiais é 640k. Não é má idéia expandir imediatamente até este limite. Mesmo que a maioria de seus programas aplicativos não utilizem tudo isto, você poderá utilizar a memória adicional para programas residentes – ou como um disco RAM. Muitos programas estão aparecendo no formato residente. Este tipo de software é carregado antes de seu programa principal e depois permanece na memória, esperando ser chamado ao toque de uma combinação especial de teclas. Borland apresentou o primeiro grande best-seller residente, SideKick. Aquele programa continha um bloco de anotações, uma calculadora e outras ferramentas particularmente úteis porque estavam imediatamente disponíveis. Os discos RAM emulam a atividade dos acionadores de disco mas fazem seu trabalho em RAM. Isto torna a gravação, o carregamento e o processamento muito mais rápidos para muitos programas. Depois que você levar sua memória a 512k ou 640k obtenha um programa de disco RAM, disponível comercialmente, ou gratuito, de muitos grupos de usuários, livros e revistas, e configure uma parte de sua RAM sobrando como disco RAM.

ACIONADORES DE DISCO

Você precisa de um acionador de disco para utilizar o Turbo Prolog. Entretanto, a maioria dos sistemas tem ao menos dois acionadores de disco. Caso você tenha apenas um acionador, considere a obtenção de mais um. Ter ao menos dois acionadores de disco ajuda o trabalho com os programas-exemplos Turbo Prolog, a duplicar os discos, e a guardar seus próprios programas com mais eficiência.

Apesar de não ser necessário um disco rígido, é interessante tirar vantagem de

sua enorme capacidade de armazenamento e velocidade. Muitos usuários de computadores não percebem que os discos rígidos não são apenas 20 ou mais vezes maiores (em espaço de armazenamento) que os discos floppy, mas que eles podem ler ou escrever a informação de forma muito mais rápida que o floppy.

CARTÕES DE ADAPTAÇÃO GRÁFICA

Um adaptador de tela monocromático da IBM, igual àquele que acompanha o PC-IBM padrão, é adequado para a maioria dos Turbo Prolog. Para tirar vantagens dos gráficos coloridos do Turbo Prolog, porém, você vai precisar de um Adaptador de Gráfico Colorido da IBM ou uma placa compatível. Existem algumas placas, como o adaptador Hércules, com a qual o Prolog não funciona. Isto poderá mudar com versões posteriores do programa.

COPIANDO DISCOS DE SISTEMA

Leia os direitos autorais que acompanham o programa. Depois faça cópias de cada disco do Turbo Prolog. Guarde os discos originais em um local fresco, seco e limpo. Utilize as cópias como discos de trabalho. Talvez você deseje fazer um segundo conjunto de discos de reserva para serem mantidos em um arquivo separado. Você deveria seguir estas mesmas regras com os discos que faz para armazenar seus próprios programas. Mantenha os diversos conjuntos de discos de reserva em locais diferentes. Infelizmente, algumas pessoas que regularmente duplicam seus discos mantêm-nos duplicados no mesmo local dos discos de trabalho. Isto expõe os discos de reserva às mesmas condições que poderiam danificar os discos de trabalho.

Caso você tenha um disco rígido, provavelmente vai querer copiar todos seus arquivos Turbo Prolog em um único diretório. Neste caso, deveria manter o cuidado de armazenar os discos floppy em um local seguro. Ainda, deve considerar a obtenção de algum tipo de duplicação de seu disco rígido – discos floppy (caso o seu disco rígido não seja muito grande; por exemplo, um disco de 10 megabyte) com um software especial de duplicação rápida, discos removíveis ou uma fita do tipo streamers.

PROCEDIMENTOS PARA CÓPIA DOS DISCOS (SOFTWARE)

Retire o plástico de seu livro Turbo Prolog. Em seu interior vai encontrar dois disco de 5 1/4": o disco Biblioteca/Exemplo (também denominado disco de Utilitários e

Programa-Exemplo) e o disco Sistema/Programa (também denominado disco de Programa). O disco Sistema/Programa tem um número de série no canto direito inferior do rótulo. Escreva aquele número na capa frontal interna de seu manual Borland.

Enquanto está olhando a parte interna da capa frontal de seu livro Borland, observe os termos de garantia e licenciamento. A garantia é padrão para os fabricantes de software dos microcomputadores de hoje. Ela promete apenas que o disco é um disco, o livro é um livro e ambos estão livres de defeito no material e fabricação. Caso não estejam serão substituídos pela Borland. A garantia informa especificamente que a Borland não está dizendo que o programa é bom para uma determinada finalidade ou utilização.

O termo de licenciamento, *Borland's No-Nonsense License Statement* é diferente de muitos outros. A empresa não pede que você deixe de fazer cópias do software, empreste-o a um amigo ou utilize-o em um computador diferente. Em vez disto, a Borland apenas pede que trate o software como um livro: não importa quantas cópias de reserva sejam feitas, assegure-se de que apenas uma cópia está sendo utilizada por vez. O assunto cópia de todos os arquivos em um disco rígido será abordado posteriormente neste capítulo. Retire o cartão de garantia e envie-o à Borland. Apesar de conseguir informação técnica e ajuda, sem que este cartão esteja arquivado em algum arquivo da Borland, responder às perguntas no cartão vai ajudar os projetistas de software da Borland na melhora do Turbo Prolog e apresentar outros programas. O envio do cartão coloca-o também na lista de correspondência da Borland; você vai receber informação técnica sobre o Turbo Prolog e informações sobre novas versões.

INICIALIZANDO

Ligue seu IBM-PC e seu monitor e coloque seu disco DOS no acionador A. Caso tenha um sistema com disco rígido, provavelmente terá o DOS no disco rígido. Neste caso, basta iniciar o sistema. Um sistema-padrão vai pedir-lhe a data e hora, que devem ser dados de maneira correta para que você os tenha corretos em seus arquivos Turbo Prolog. Caso não lhe seja solicitado introduzir data e horário, a razão talvez seja porque você tem um relógio interno e um arquivo AUTOEXEC.BAT. Quando um destes arquivos está no disco de inicialização, o sistema executa automaticamente os comandos do arquivo.

Caso utilize programas residentes (como SideKick ou ProKey), não deverá ter muitos problemas em seu carregamento a não ser que utilize muita memória. Lembre-se: é preciso ao menos 384k, que inclui 20 ou 30k para o DOS e o restante para o Turbo Prolog, livre e limpo. Uma vez que alguns dos novos programas residentes requerem mais que 100k, não será muito difícil sobrecarregar mesmo um sistema com 512k ao ponto de não

se poder processar o Turbo Prolog. Caso isto ocorra, será enviada uma mensagem tipo "Memória Insuficiente" à tela, ao se tentar processar o Turbo Prolog.

IMPRIMINDO O ARQUIVO README

Os discos Turbo Prolog podem ser modificados com uma freqüência muito maior do que os manuais. Por este motivo, a Borland coloca em um dos discos um arquivo README que inclui correções do manual, novas informações e outras notas que você poderá necessitar. Você pode ler este arquivo na tela ou imprimi-lo. A Borland inclui um programa README.COM que facilita ambas as opções. Coloque o disco Sistema/Programa no acionador A, digite

```
readme
```

e pressione RETURN. O programa README.COM vai dizer-lhe que está lendo o arquivo README. Depois vai apresentar a primeira página daquele arquivo na tela, juntamente com um menu na última linha com os comandos que manuseiam o arquivo README, conforme indicado na Figura 1-1. Para imprimir o arquivo README, assegure-se

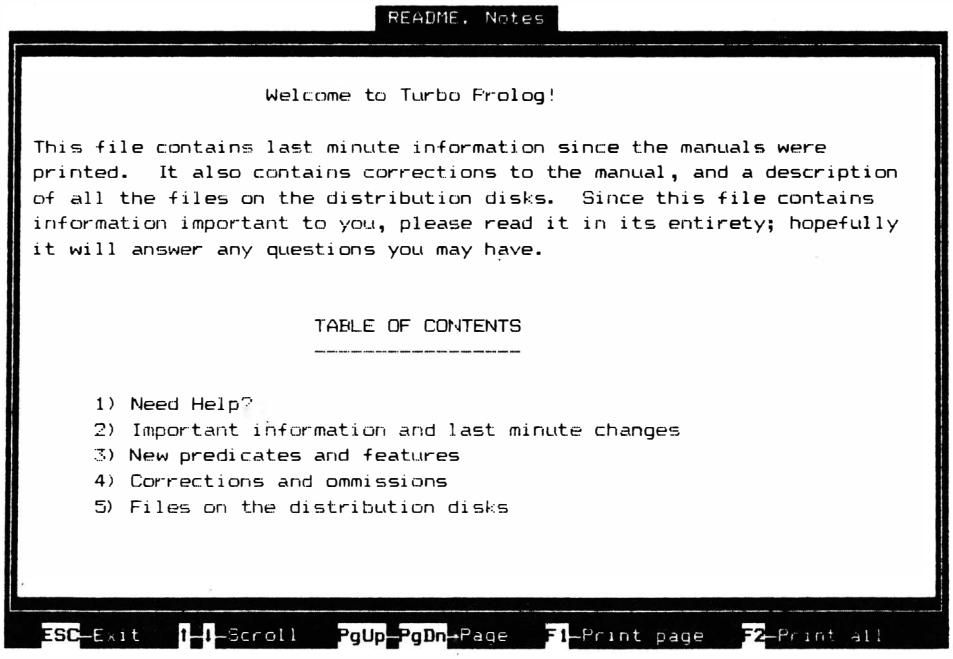


Figura 1-1 Primeira tela do programa README.COM com o arquivo README

de que a sua impressora está instalada e ligada. Depois pressione a tecla de função F2 enquanto estiver no programa README.COM.

VERIFICANDO TODOS OS ARQUIVOS

Antes de realmente copiar todos os discos, verifique se todos os arquivos necessários estão neles. Retire o disco do DOS e coloque o disco Sistema/Programa do Turbo Prolog no acionador A (ou qualquer acionador que esteja utilizando para discos flexíveis caso tenha um sistema de disco rígido). Digite

dir:

e RETURN para obter a lista de todos os arquivos naquele disco. Você deveria ter os mesmos arquivos que os listados na Figura 1-2. Retire o disco Sistema/Programa e substitua-o pelo disco Biblioteca/Exemplo. Execute os mesmos passos para a obtenção de um diretório, e compare aquele diretório com a lista da Figura 1-3. A lista deveria ser a mesma ou deveria haver uma explicação no arquivo README. O seu tamanho exato e os dados não têm importância, apenas o nome do arquivo.

COMO COPIAR OS DISCOS

Existem dois comandos para cópia de disco: Diskcopy e Copy. Em qualquer um, serão necessários dois discos floppy em branco ou que contenham arquivos dos quais você

```
README   COM     17867   4-24-86   4:11p
PROLOG   EXE    237888   6-19-86   9:17a
PROLOG   SYS        92   4-28-86  12:02p
PROLOG   ERR      7131   5-15-86   9:26a
PROLOG   HLP      9962   4-25-86   6:27a
PLINK    BAT       510   5-09-86   9:05a
HANOI    PRO      2348   4-25-86  12:55p
INSTALL  COM     12308   4-28-86   1:58p
README           10843   6-23-86  11:15a
COMMAND  COM     17664   3-08-83  12:00p
       10 File(s)     39936 bytes free
```

Figura 1-2 Diretório de arquivos no disco Sistema/Programa

não precisa mais. Obtenha algumas etiquetas para disco e escreva "Turbo Prolog Sistema/Programa – Cópia de Trabalho", em uma e "Turbo Prolog Biblioteca/Exemplo – Cópia de Trabalho" na outra. Depois, coloque as etiquetas nos discos. Não coloque primeiro as etiquetas no disco para depois escrever com uma esferográfica, uma vez que isto pode danificar os discos.

O comando Disckcopy faz melhor o trabalho de duplicação para você do que comando Copy. Para utilizá-lo, coloque o disco DOS no acionador principal e digite

```
diskcopy a: b:
```

com exatamente estes espaços simples entre as partes, conforme indicado. Depois digite RETURN. Caso seu programa Diskcopy esteja em seu disco de sistema, o computador responderá pedindo que você coloque seu disquete-fonte no acionador A e o disco-destino no acionador B. (Caso o seu comando Disckcopy não esteja em seu disco DOS, tente achá-lo ou tente o método do comando Copy, explicado posteriormente.) Coloque o disco Sistema/Programa em A e um disco recém-etiquetado em B. Depois, opere qualquer tecla. Quando a cópia estiver concluída, opere Y (de "Sim, quero copiar outro"). Depois coloque o disco Biblioteca/Exemplo em A e o segundo disco recém-etiquetado em B. Opere novamente qualquer tecla e espere o término da cópia. Depois, opere N de "Não", retire os discos e guarde os originais em local seguro.

UTILIZANDO O COMANDO COPY

Para utilizar o comando Copy em lugar do comando Disckcopy, você precisa primeiro formatar os discos recém-etiquetados (Diskcopy faz isto para você). Ponha etiquetas em seus discos de trabalho como acabei de descrever. Depois, coloque seu disco DOS no acionador A, digite

```
format b:/s
```

e RETURN. O computador vai solicitar que você coloque um disco no acionador B. Utilize o primeiro dos discos recém-etiquetados. Depois digite qualquer tecla, e a formatação se inicia. A formatação prepara o disco para a informação. Discos em branco, recém-saídos do envólucro do fornecedor não têm os dados de sinalização que lhes permitem organizar e, portanto, armazenar informação. A formatação coloca estes sinais. O "/s" depois do comando de formatação coloca a informação básica do sistema operacional no disco. Um disco com a informação de sistema pode ser utilizado como o primeiro disco no com-

```
PROLOG   LIB   155648   6-19-86   9:17a
INIT     OBJ     1079   6-12-86   6:54p
GEOBASE  PRO     6406   4-25-86   6:15p
GEOBASE  INC    22536   4-25-86  10:41a
GEOBASE  DBA    45237   4-28-86  10:55a
GEOBASE  HLP     3428   4-28-86  10:56a
EXAMPL1  PRO      293   4-01-86   1:35p
EXAMPL2  PRO      292   4-01-86   1:37p
EXAMPL3  PRO      214   4-01-86   1:37p
EXAMPL4  PRO      525   4-25-86  11:27a
EXAMPL5  PRO     1986   4-25-86  11:28a
EXAMPL6  PRO      318   4-25-86  11:28a
EXAMPL7  PRO      412   6-12-86  12:06p
EXAMPL8  PRO      174   4-01-86   1:39p
EXAMPL9  PRO      539   4-24-86   4:05p
EXAMPL10 PRO      278   5-22-86   4:10p
EXAMPL11 PRO      223   4-01-86   1:40p
EXAMPL12 PRO      393   4-24-86   4:29p
EXAMPL13 PRO      423   4-24-86   4:40p
EXAMPL14 PRO      426   5-19-86   5:01p
EXAMPL15 PRO      340   4-25-86  11:26a
EXAMPL16 PRO      790   4-01-86   1:41p
EXAMPL17 PRO      249   4-25-86  11:26a
```

Figura 1-3 Diretório-amostra de arquivos no disco Biblioteca/Exemplo

putador quando este é ligado. Acrescente "/s" apenas ao disco que utilizará para conter o programa Turbo Prolog. Você não precisa fazê-lo para o disco de trabalho Biblioteca/Exemplo.

Para formatar um disco de trabalho para programas utilitários e exemplo, coloque o segundo disco etiquetado no acionador B, e com o disco DOS no acionador A, digite

```
format b:
```

e RETURN. O programa de formatação lhe pede para colocar os discos B, que já estão ali, de modo que basta digitar qualquer tecla. Este procedimento vai formatar um disco vazio, sem a informação de sistema. Não é necessário, realmente, que os dois discos de trabalho sejam discos de sistema. Além disto, o disco Biblioteca/Exemplo está tão cheio que a informação de sistema não cabe nele a não ser que você se livre de algum outro arquivo.

Agora que ambos os discos de trabalho, com etiquetas, estão formatados, coloque o disco de Sistema/Programa original no acionador A e o disco vazio, de trabalho de Sistema/Programa, no acionador B. Digite

```
copy a:*.* b:
```

e RETURN. Todos os arquivos no disco original serão copiados para o disco de trabalho. Quando isto estiver feito, ponha de lado o disco original e coloque o disco de trabalho em seu arquivo de discos. Coloque o disco original Biblioteca/Exemplo no acionador A e o disco de trabalho em branco Biblioteca/Exemplo no acionador B, e depois digite o comando Copy como antes. Novamente, assim que a cópia estiver concluída, guarde o original em local seguro. Utilize apenas as cópias de trabalho dos discos de Sistema/Programa e Biblioteca/Exemplo.

SISTEMAS DE DISCO RÍGIDO

Caso seu disco rígido seja chamado de "C", coloque o disco de Sistema/Programa no acionador A. Com o DOS no acionador C, digite

```
copy a:*.* c:
```

e RETURN. Caso você queira colocar Turbo Prolog em um determinado diretório, terá de incluir a informação nas instruções. Diretórios de disco rígido estão além da discussão deste livro. Você deve ir até uma loja de computação, comprar um dos livros a respeito de PC-DOS ou MS-DOS, para depois se dedicar aos comandos básicos do DOS, incluindo "mkdir" e "chdir".

Tabela 1-1 Descrição dos arquivos no Disco Original

Arquivo	Descrição
README.COM	Um programa que apresenta um arquivo README, com comandos para visualizá-lo e percorrê-lo.
README	Um arquivo de texto que contém atualizações e sugestões para Turbo Prolog. Pelo fato do disco poder ser mudado com muito mais freqüência do que o manual, a informação mais recente – incluindo correções e adições ao manual – é colocada neste arquivo.
PROLOG.EXE	O principal programa do compilador Turbo Prolog. Este arquivo contém o editor, compilador, gerenciador de arquivos e rotinas de execução.

PROLOG.SYS	Um arquivo contendo informações a respeito de cor, tamanho e localização de janela e diretórios default.
PROLOG.ERR	O arquivo de mensagens de erro. O compilador vai encontrar erros e mencioná-los por número sem este arquivo, mas não será capaz de acrescentar uma descrição em inglês sobre o erro.
PROLOG.HLP	O arquivo de mensagens de auxílio. Pelo fato deste arquivo ser em texto ASCII, você pode modificá-lo como deseja.
PROLOG.LIB	O arquivo de biblioteca, vital para a operação LINK que cria arquivos. EXE (arquivos executáveis) a partir de arquivos. OBJ (arquivos-objeto).
INIT.OBJ	Outro arquivo envolvido no LINK. Tem que ser o primeiro arquivo. OBJ em todos os comandos LINK para a criação de arquivos. EXE. INIT.OBJ tem o código para inicializar o PC antes que você trabalhe no programa Turbo Prolog.
PLINK.BAT	Outro arquivo para LINK. Este arquivo de lote liga arquivos de código. OBJ.
*.PRO	Um arquivo que inicia com letras e tem a extensão .PRO no final. Este é o formato utilizado para arquivos de Programa Turbo Prolog. A maioria dos arquivos no disco Biblioteca/Exemplo tem este formato, como qualquer arquivo de programa gravado a partir do Turbo Prolog.
GEOBASE.PRO	Um arquivo que contém o programa-fonte para o banco de dados de informação geográfica da linguagem-natural GeoBase.
GEOBASE.INC	Este arquivo contém mais programas-fontes GeoBase.
GEOBASE.DBA	O banco de dados GeoBase.
GEOBASE.HLP	As mensagens de auxílio para o programa GeoBase.
READ64.ME	Outro arquivo de informação adicional (como o README principal). Entretanto, este arquivo diz respeito apenas ao programa-exemplo EXEMPL64.PRO.

ARQUIVOS MÍNIMOS PARA DISCOS COM POUCO ESPAÇO

Nem todos os arquivos Turbo Prolog enviados pela Borland são necessários para todas as utilizações de Turbo Prolog. Mesmo a maioria dos materiais no disco Sistema/Programa são em parte dispensáveis. A Tabela 1-1 detalha o que são os arquivos principais e o que fazem.

Os únicos arquivos necessários para iniciar Turbo Prolog na maioria das circunstâncias são PROLOG.EXE e PROLOG.SYS. Para a obtenção de mensagens de erro juntamente com os números de erro, precisará PROLOG.ERR no mesmo diretório que PROLOG.EXE. Outros arquivos são necessários para determinados tipos de compilação (veja o Capítulo 12).

CAPÍTULO 2

O AMBIENTE DO TURBO: MENUS E JANELAS

Aprender a utilizar uma determinada linguagem de computação não é apenas uma questão de aprender a teoria geral daquela linguagem: é preciso dominar uma determinada implementação. Faça a comparação com o ato de dirigir um carro, onde o conhecimento sobre as funções da barra de direção, do acelerador e do breque não são suficientes. Você não conhece realmente o seu carro antes de tê-lo dirigido em diversas condições, aprendido a disposição de instrumentos e controle, tido uma impressão da mudança de marchas e saber onde consertá-lo.

Computadores e linguagens de computação não são tão padronizados como carros. Não existem dois Prolog exatamente iguais. O conjunto de instruções do Turbo Prolog, bem como seus comandos, diferem de qualquer outro Prolog. Adicionalmente, o Turbo é apresentado com seu próprio pacote de janelas e edição.

O Turbo Prolog tem um sistema intrínseco de menus e janelas para ajudá-lo a escrever, compilar, processar e depurar programas. Este capítulo vai descrever estes menus e janelas, de modo que você esteja pronto para dar o próximo passo no Capítulo 3, onde aprenderá a editar (escrever e modificar) um programa. Caso você saiba como iniciar um programa a partir do DOS, o que são janelas, e como utilizar menus de vários níveis, e se ainda está disposto a aprender os comandos de menu do Turbo Prolog na medida em que programa, você não precisa ler este capítulo.

Dois pontos lhe permitirão uma manipulação básica dos menus do Turbo Prolog: uma familiaridade com os principais comandos do Turbo Pascal – Edit, Compile, Run, Save e outros – e a experiência com um programa como o Microsoft Word que utiliza menus de níveis múltiplos na parte inferior da tela. Caso você tenha trabalhado com uma das linguagens no Macintosh da Apple, que utiliza janelas separadas para listagem, depuração

etc., poderá deixar este capítulo de lado por ora. Posteriormente, poderá voltar a dar uma olhada melhor em algumas opções específicas do menu de múltiplos níveis.

INICIANDO O PROGRAMA

Agora que você copiou seus discos e sabe o que fazer quando ocorre algum problema (explicado no Capítulo 1), está pronto para começar com o Turbo Prolog e dar uma olhada em seu ambiente de programação. Coloque seu disco de trabalho Sistema/Programa do Turbo Prolog no acionador A e o disco Biblioteca/Exemplo do Turbo Prolog no acionador B. Caso você tenha um sistema de disco rígido, vá ao diretório que contém seus arquivos Turbo Prolog. O arquivo mais importante é o PROLOG.EXE. Este é o programa principal do Turbo Prolog. Digite:

```
prolog
```

e RETURN. Depois de alguns segundos verá a tela de status inicial do Turbo Prolog, mostrada na Figura 2-1.

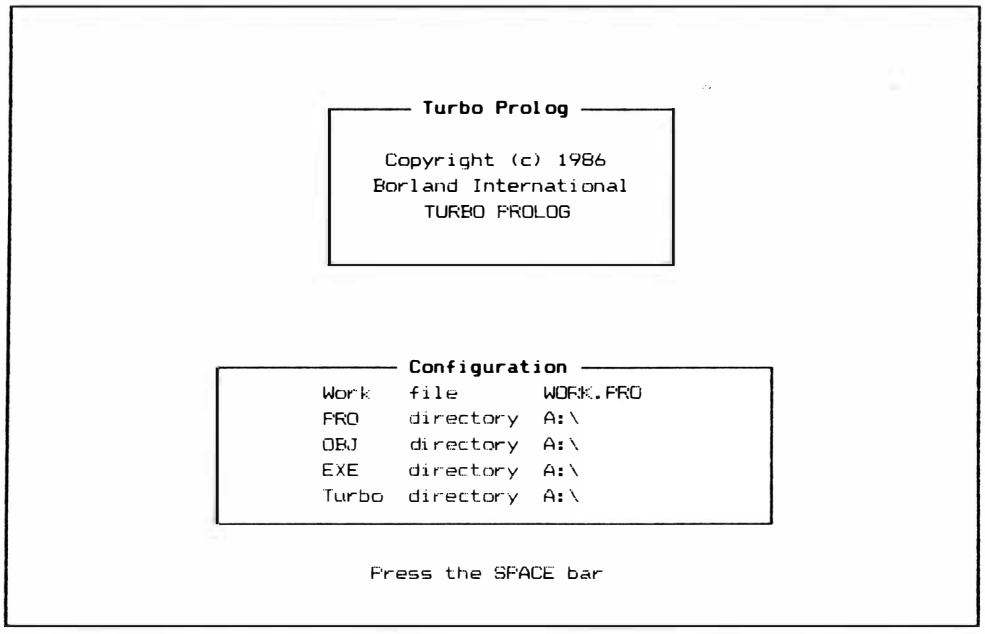


Figura 2-1 Tela inicial de direitos autorais

TELA INICIAL DE DIREITOS AUTORAIS

Não pressione a barra de espaço ainda. Dê primeiro uma olhada melhor nesta tela, e observe algumas coisas a respeito do Turbo Prolog que está em vias de utilizar. Poderá ver em qual versão está trabalhando. Correções de problemas e adições à linguagem provavelmente farão com que a Borland emita versões com números do tipo 1.02, 1.1 e 2.0. (Todos os programas têm problemas, e a complexidade dos programas de linguagem normalmente significa que os têm mais do que deveriam.)

A área rotulada "configuration" informa-lhe que o nome temporário de qualquer arquivo com o qual você começa a trabalhar é WORK.PRO. Informa-lhe, também, que todos os seus diretórios estão presentemente alocados no disco A. Isto não importa muito agora, mas ficará mais importante na medida em que você avançar mais com o Turbo Prolog.

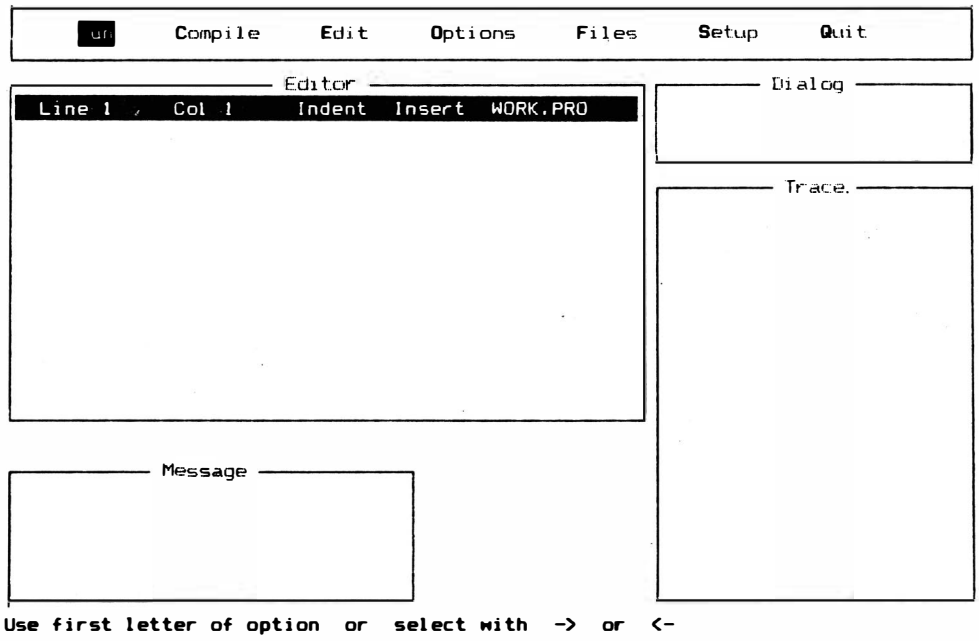


Figura 2-2 Tela principal do Turbo Prolog

TELA PRINCIPAL

Digite agora a barra de espaço em seu teclado. Verá assim a tela mostrada na Figura 2-2. Caso você seja um usuário experiente em computação, as possibilidades serão de que você já viu uma janela como esta. Caso você seja um novato, não se intimide. Isto será mais fácil do que trabalhar com muitas linguagens antigas, como a versão inicial do Turbo Pascal.

A Figura 2-3 mostra a tela principal da versão 3.0 do Turbo Pascal. Conforme mostrado pela Figura 2-4, Turbo Pascal informa-lhe qual acionador e diretório são ativos, o nome do arquivo em que está trabalhando, quanto memória tem e quais as suas escolhas de comando. Assim que estiver nesta tela em Turbo Pascal, selecione um dos comandos digitando a primeira letra do comando. Digite E, por exemplo, e estará no editor.

O ambiente do Turbo Prolog é um pouco mais fácil de compreender do que o do dBASE ou mesmo o do Turbo Pascal devido à adição de menus de múltiplos níveis e janelas. A Borland declarou publicamente que este ambiente será utilizado em todas as linguagens futuras, inclusive na nova versão do Turbo Pascal. Aprenda-a aqui, e estará pronto para muito mais além do Turbo Prolog.

Na tela principal do Turbo Prolog – referenciado em todo este livro como a tela principal – você verá quatro janelas separadas, uma linha de menu na parte superior e uma linha de informação na parte inferior (veja Figura 2-2).

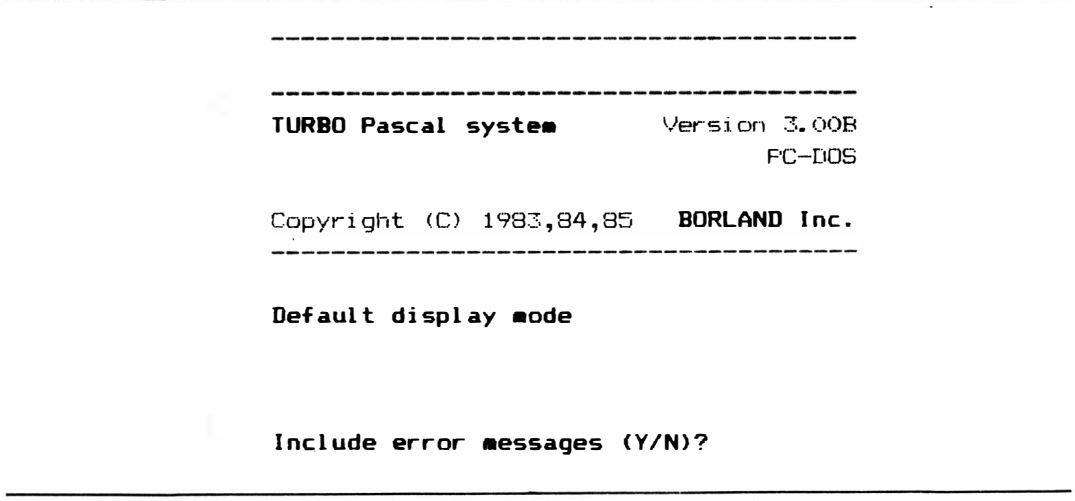


Figura 2-3 Tela principal do Turbo Pascal 3.0

```
Logged drive: B
Active directory: \

Work file:
Main file:

Edit      Compile  Run    Save

Dir       Quit  compiler Options

Text:      0 bytes
Free: 63485 bytes

>
```

Figura 2-4 Menu principal do Turbo Pascal

JANELAS

Janelas são partes da tela completa do computador que se comportam como telas completas independentes. Em um sistema sem janelas, mudar de uma tarefa para outra (por exemplo, passar da edição de um programa para o processamento do programa) significa, com freqüência, cancelar todos os elementos da tela de Edição e substituí-los pela tela de Processamento. Caso você tenha um sistema de janelas, poderá optar por ver tanto a tela de Edição como de Processamento ao mesmo tempo.

A utilização de Turbo Prolog não é a única maneira de induzir seu PC a "fazer janelas". Você poderia ainda comprar um software especial de janelas que é acrescentado ao seu sistema operacional e cria um ambiente com janelas para outros programas aplicativos. Com um programa deste tipo, seu PC reservaria uma janela para cada programa em separado que processa.

As janelas do Turbo Prolog são construídas no compilador e feitas sob medida para gerenciar os quatro aspectos principais da programação Turbo Prolog: editar, perguntar, acompanhar execução e depurar. Cada uma das janelas pode ser movimentada, reduzida ou ampliada.

JANELA DE EDIÇÃO

A janela que você vai utilizar com maior freqüência é a de edição. Na tela princi-

pal do Turbo Prolog, esta é a maior janela entre as quatro e está no canto superior esquerdo da tela (veja novamente a Figura 2-2). Logo abaixo da linha título da janela de Edição existe uma linha que vai informar-lhe o estado de seu trabalho de edição:

- A mensagem "Text:" da versão 1.0 informa quantos caracteres existem no arquivo que você está editando – o arquivo que aparece na janela de Edição.
- A mensagem "Free:" da versão 1.0 informa quanto espaço você ainda tem sobrando no arquivo. O limite é 64k caracteres.
- As mensagens "Line" e "Column" da versão 1.1 e posteriores substituíram "Text" e "Free". Informam-lhe a posição da linha e da coluna do cursor na janela de Edição.
- A mensagem "Indent" será destacada quando o editor estiver no modo Auto Indent (explicado no Capítulo 3).
- A mensagem "Insert" será destacada quando o editor estiver no modo Insert em vez de no modo Overwrite.
- Finalmente, a última mensagem da direita, depois da mensagem "Insert", é o nome do arquivo que está sendo editado. Pelo fato do nome default ser WORK.PRO, é isto o que você vai ver ali. (Os compiladores pegam *arquivos-fontes* que você escreveu com um editor e os compila em *arquivos-objetos*. Qualquer arquivo-fonte do Turbo Prolog terá a extensão .PRO em seu nome.) Caso a janela não tenha largura suficiente para mostrar tudo, o que verá será "WORK". Caso amplie a janela, puxando a sua margem direita para mais perto da borda direita da tela, verá todo o nome do arquivo.

JANELA DE DIÁLOGO

Os programas Prolog tipicamente percorrem uma lista de regras e fatos na tentativa de responder perguntas, mas também podem fazer o que todas as outras linguagens de programação têm condições de fazer: podem fazer jogos, traçar gráficos ou resolver equações. Você vai formular perguntas e obter respostas na janela de Diálogo, na parte superior direita da tela principal. A não ser que o programa especifique em contrário, é ali também que o Turbo Prolog apresenta tudo o mais que o programa lhe pede para fazer.

JANELAS DE MENSAGEM

Algumas variedades de Prolog não lhe informam quando alguma coisa está erra-

da. Caso você cometa um erro de digitação ou caso haja um problema com seu programa, o interpretador ou compilador Prolog fornece-lhe no máximo uma mensagem de erro misteriosa.

Turbo Prolog utiliza a janela de Mensagem (canto inferior esquerdo na tela principal) para informar-lhe o que está ocorrendo, certo ou errado. Quando você lhe instrui para fazer uma compilação, o computador faz uma anotação na janela de Mensagem a respeito do que está fazendo. O compilador Turbo Prolog inclui também um rigoroso verificador de sintaxe que vai alertá-lo quanto à localização e conteúdo de erros em seu programa.

Turbo Prolog não é suficientemente esperto para informar-lhe quando um programa não lhe dará o resultado esperado. Tem condições apenas de encontrar *erros que o impedirão* de continuar. Erros e mensagens de erro serão abordados com maiores detalhes mais adiante. Talvez você, queira consultar o *Manual do Proprietário* da Borland para dar uma rápida olhada nas mensagens de erro. Certamente os verá novamente durante o aprendizado da linguagem.

JANELA DE ACOMPANHAMENTO ("TRACE")

Mesmo que seu programa cumpra as regras de sintaxe oficiais do Turbo Prolog, poderá não executar exatamente o que você quer. Caso ele execute o planejado, você poderá ainda querer saber como foi do começo ao fim. Esta é a finalidade da janela de Acompanhamento (canto inferior direito da tela principal).

Você pode acrescentar comandos de acompanhamento aos programas em Turbo Prolog para forçá-los a operar passo a passo e não de uma só vez. Depois de cada passo, o programa pára e a janela de acompanhamento informa o que realizou. O acompanhamento será discutido posteriormente no livro.

MENU PRINCIPAL

Um *menu* é uma lista de comandos e funções possíveis. Os menus são populares em diversos programas de computadores porque aliviam os programadores e usuários da tarefa de lembrar todas as escolhas de comandos possíveis. Existem diversos tipos de menus; existem até menus dentro de menus denominados *submenus*, quando comandos poderosos contêm opções mais especializadas. Turbo Prolog utiliza uma linha no topo da tela principal para mostrar-lhe os sete comandos fundamentais: Run, Compile, Edit, Options, Files, Setup e Quit. Caso haja um submenu com opções mais especializadas quando você escolhe um comando, o Turbo Prolog apresenta o submenu e aguarda que você faça uma escolha adicional. Caso não haja mais escolhas a fazer, Turbo Prolog tenta executar seu

programa. Se encontrar algum problema com a compilação, você será informado com a mensagem na janela.

SELEÇÃO EM UM MENU

Você pode escolher um dos comandos principais e posteriormente uma das opções do submenu, de duas formas:

- digite a primeira letra da opção escolhida. Por exemplo, se digitar F selecionará o comando Files, e verá um submenu aparecendo abaixo da palavra "Files" na linha de menu;
- utilize as teclas de seta (teclas de direcionamento do cursor) para movimentar a parte iluminada até o comando de sua escolha. Isto funciona também nos submenus, mas você precisará das setas para cima e para baixo em lugar das de esquerda e direita. Assim que seu comando escolhido estiver iluminado, tecle RETURN.

ESCAPE

Utilize a tecla Escape (ESC, à esquerda da parte alfabética, antes da tecla 1 do teclado do PC) sempre que não lhe agradar o local em que você estiver. Caso esteja trabalhando no interior de uma janela, tecle ESC para voltar à linha do menu principal. Você poderá ver a linha de menu enquanto estiver dentro da janela, mas não poderá escolher nenhum dos comandos do menu. Se estiver em um submenu, tecle ESC para voltar ao comando do menu principal acima daquele submenu.

COMANDOS SEM SUBMENU

Quatro entre os sete comandos fundamentais não têm submenus especiais. São Quit, Edit, Compile e Run.

COMANDO QUIT

O comando Quit sai do Turbo Prolog. Ele pára o programa e volta à indicação DOS (A >, caso este esteja no disco com o qual você começou). A operação Q fará com que você saia da "via de alta velocidade" do Turbo Prolog e volte ao vagaroso mundo do DOS. Existem duas exceções a esta regra. Caso o arquivo com o qual você está trabalhando tenha sido editado desde que você o carregou, ou se é completamente novo, Turbo Prolog vai perguntar se você quer gravar o arquivo. Caso você responda "Yes", ele o aju-

dará nesta tarefa. Caso responda "No", ele desaparecerá e voltará ao DOS.

COMANDO EDIT

Quando você tecla E no menu principal, salta para a janela do Editor. Assim que estiver lá, poderá escrever um novo programa ou modificar um antigo (caso ele já tenha sido carregado utilizando o comando Files e seus submenus).

COMANDO COMPILE

Assim que você tiver editado com sucesso (escrito ou reescrito) um programa, a ponto dele estar pronto para ser experimentado, você precisará do comando Compile. Digite C enquanto estiver no menu principal, e o arquivo de programa na janela Editor será compilado em um programa pronto para ser processado na memória principal do computador.

A compilação poderá ser muito simples. Entretanto, para programas mais complexos, o Turbo Prolog contém diversas opções de compilação (no submenu do comando Options) para a compilação de um programa de diversos tipos de arquivos em disco. Contém também algumas diretrizes poderosas de compilação.

COMANDO RUN

Depois de ter editado ou escrito um programa e ter feito a sua compilação, a próxima coisa a ser feita é seu processamento. O comando Run dispara o programa que estiver na memória do computador – normalmente o programa recém-compilado na memória. Caso você tenha compilado um programa na memória, mas depois disto foi ao editor para modificar o arquivo-fonte daquele programa, o programa Run primeiro compilaria o programa (automaticamente, sem que você tenha que operar a tecla C) e depois o processa. O comando Run reconhece dois tipos diferentes de programas Turbo Prolog: aqueles com metas internas e aqueles sem. Caso um programa tenha uma meta interna, ele processará completamente aquela meta quando você aciona o comando Run. Assim que o programa tiver processado, tudo que tem a fazer é operar a barra de espaço, para voltar ao menu principal. Caso um programa não tenha uma meta interna, o programa começa a ser processado e depois espera que você entre com uma meta na janela Dialog. Algumas das teclas de função tem os comandos especiais atribuídos a elas enquanto um programa é processado. É fácil utilizar o comando Run. Opere a tecla ESC quantas vezes é necessário para ir de onde quer que esteja ao menu principal. Depois opere a tecla R. Quando se decidir a utilizar a janela Trace, a utilização de Run será um pouco mais complicada.

COMANDOS COM SUBMENUS

Os comandos Options, Files e Setup têm submenus. Você vai descobri-los ao selecionar o comando principal associado. O submenu aparece abaixo – ou é puxado – do comando principal.

COMANDO OPTIONS E SUBMENU

A Figura 2-5 mostra o submenu que é "puxado" quando você seleciona o comando Options. Todas as escolhas do Options dizem respeito à compilação do programa. São colocadas abaixo de "Options" em vez de "Compile" porque não são utilizadas com muita freqüência, ao passo que o comando Compile o é.

COMANDO FILES E SUBMENU

O comando Files inclui muitas das funções que você realizaria usando o DOS

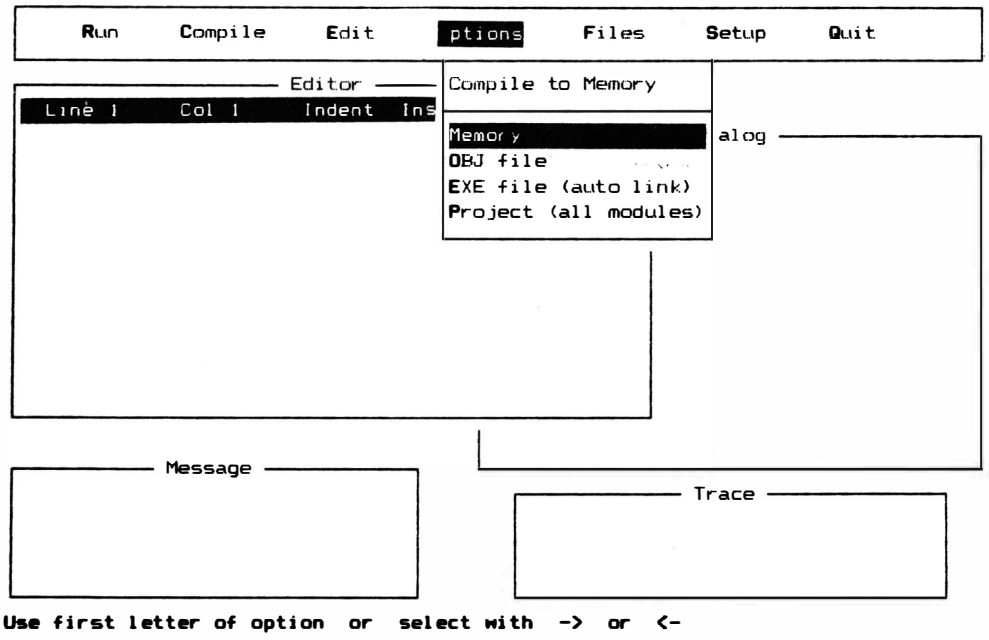


Figura 2-5 Submenu do comando Options

fora do Turbo Prolog. De fato, permite até mesmo que você vá até o DOS e volte sem tirar o Turbo Prolog da memória. A Figura 2-6 mostra o submenu abaixo do comando Files. Lembre-se: Files não faz nada até que você selecione uma das escolhas de submenu. Para fazer isto, acione a tecla da primeira letra do comando ou utilize as teclas de seta para iluminar o comando e depois tecle RETURN. Se a qualquer momento o Turbo Prolog não encontrar um arquivo que você solicitou, ele vai notificá-lo na janela Message.

COMANDO LOAD

O comando Load permite que você traga arquivos de um disco e os coloque na janela Editor para modificação ou execução. Escolha o comando Load e lhe será solicitado fornecer um nome de arquivo. Existem duas maneiras de especificar o nome do arquivo que você quer carregar:

- digitando o nome correto do arquivo e teclando RETURN. Se não colocar o ponto e a extensão do tipo de arquivo no fim do nome do arquivo, Turbo Prolog presumirá que você está operando com um arquivo Turbo Prolog padrão e acrescentará um .PRO ao nome que você forneceu;

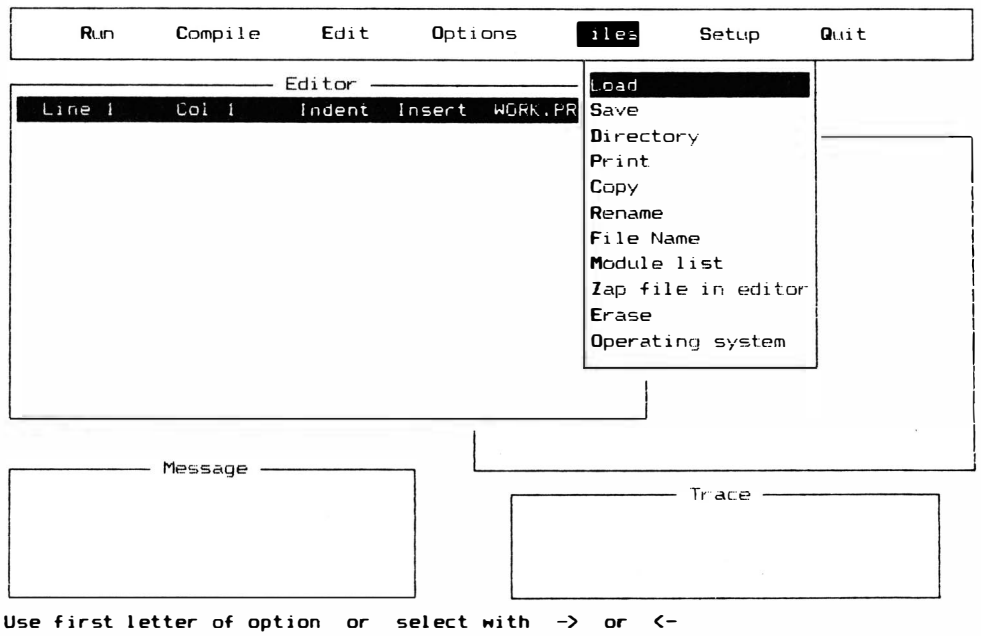


Figura 2-6 Submenu do comando Files

- escolhendo o arquivo desejado de um diretório. Os comandos Load, Save, Directory, Rename e Erase oferecem este recurso.

CONSEGUINDO UM DIRETÓRIO COM O COMANDO LOAD

Assim que tiver escolhido o comando Load, se você teclar RETURN, sem listar um nome de arquivo, verá um diretório de todos os arquivos com extensões .PRO (veja a Figura 2-7 como exemplo). Esta lista virá do diretório e acionador de disco default. Você pode alterar este default permanentemente com a escolha do diretório no submenu Files (descrito posteriormente neste capítulo), ou pode apresentar o comando Load com um padrão contendo caracteres-chaves e depois teclar RETURN.

Por exemplo, caso você responda à pergunta sobre o nome do arquivo digitando

a: e*.pro

e RETURN, verá o diretório de todos os arquivos no acionador A que começam com a letra "e", eles têm de uma a sete letras depois do "e", e terminam com uma extensão

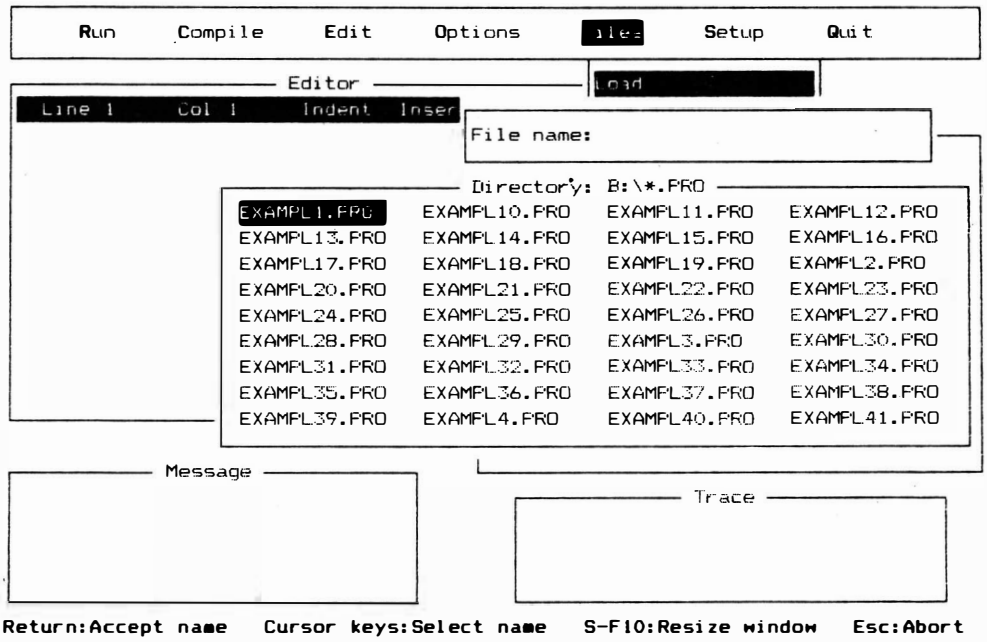


Figura 2-7 Exemplo de um diretório obtido através do comando Load

.PRO. Consulte um manual DOS para uma descrição melhor de caracteres-chaves.

Assim que tiver um diretório, ilumine o arquivo desejado usando as teclas de seta junto com HOME, END, PGUP, e PGDN e depois tecle RETURN para obter o arquivo. Você pode ainda teclar ESC de qualquer diretório para voltar à pergunta do nome de arquivo.

COMANDO SAVE

Escolha o comando Save para gravar o arquivo da janela do Editor em um disco. Caso haja outro arquivo no disco com o mesmo nome do arquivo que está tentando salvar (isto normalmente seria o arquivo cuja cópia você carregou antes de começar a edição) o comando Save vai alterar a extensão no arquivo antigo para .BAK. Desta forma, você terá uma nova versão com uma extensão .PRO e um arquivo de reserva uma geração anterior.

DIRETÓRIO

Este comando permite que você obtenha um diretório daquilo que está no disco, sem passar pelo comando Load ou Save. Utilize-o da mesmo forma que a opção Directory do comando Load. O acionador de disco e percurso escolhidos serão os default (a seleção-padrão a não ser que você as altere) para todos os comandos Files.

COMANDO PRINT

O comando Print utilizará sua impressora para fazer uma cópia impressa do arquivo na janela Editor. Ao contrário de todos os outros comandos Files, este poderá ser utilizado apenas para o arquivo que se encontra na janela Editor, e não para arquivos em disco.

COMANDO COPY

Caso você queira copiar um arquivo de um disco para outro, utilize este comando. Tudo o que tem a fazer é especificar o nome do arquivo de onde será feita a cópia e o nome do arquivo onde vai ser feita a cópia.

COMANDO RENAME

Para alterar o nome do arquivo no disco, utilize o comando Rename. Digite o nome antigo e tecle RETURN. Depois digite o novo nome e tecle RETURN.

FILE NAME

O comando File Name vai alterar o nome do arquivo na janela do Editor. Isto é útil se, por exemplo, você quiser carregar um programa, retirar uma parte dele, e depois construir um novo programa com o que sobrou. Caso você tenha mantido o nome do programa antigo, o novo programa será gravado sobre ele, relegando o programa antigo a um arquivo .BAK. Depois, você terá de utilizar o comando Rename para dar ao programa novo um nome apropriado. Utilize File Name para alterar o nome corrente, e depois, quando você escolher o comando Save, não terá de alterar o programa antigo.

MODULE LIST

Programação modular, ou a divisão de um pedaço grande de software, em módulos independentes, menores, é uma ferramenta fundamental na moderna engenharia de software. Turbo Prolog pode cuidar de programas que são divididos em módulos. Este comando permite que você diga o compilador o nome do arquivo-projeto que relaciona todos os nomes de módulos.

ZAP FILE IN EDITOR

Este comando permite que você cancele rapida e completamente o arquivo na janela Editor. Não se preocupe com uma operação acidental dele. Antes que o arquivo seja realmente "eliminado" você tem de estar certo de que realmente quer perdê-lo. O nome do arquivo cancelado continuará sendo o nome do arquivo de trabalho.

COMANDO ERASE

Erase é outro comando para manipulação de arquivo que funciona como Load, Save, ou Rename. Você terá de especificar o nome do arquivo que deseja cancelar do disco digitando seu nome ou utilizando um diretório, da mesma forma que faz com o comando Load.

OPERATING SYSTEM

Este comando permitirá que você utilize os comandos DOS sem tirar o Turbo Prolog da memória do PC. Uma vez que o escolha, você pode digitar comandos DOS como se não tivesse Turbo Prolog na memória residente. Quando terminar com DOS, digite

```
exit
```

e RETURN para voltar ao Turbo Prolog. O DOS com o qual você inicializou seu computador tem que estar disponível no disco quando você utilizar este comando, ou receberá uma mensagem de erro e terá que voltar ao Turbo Prolog.

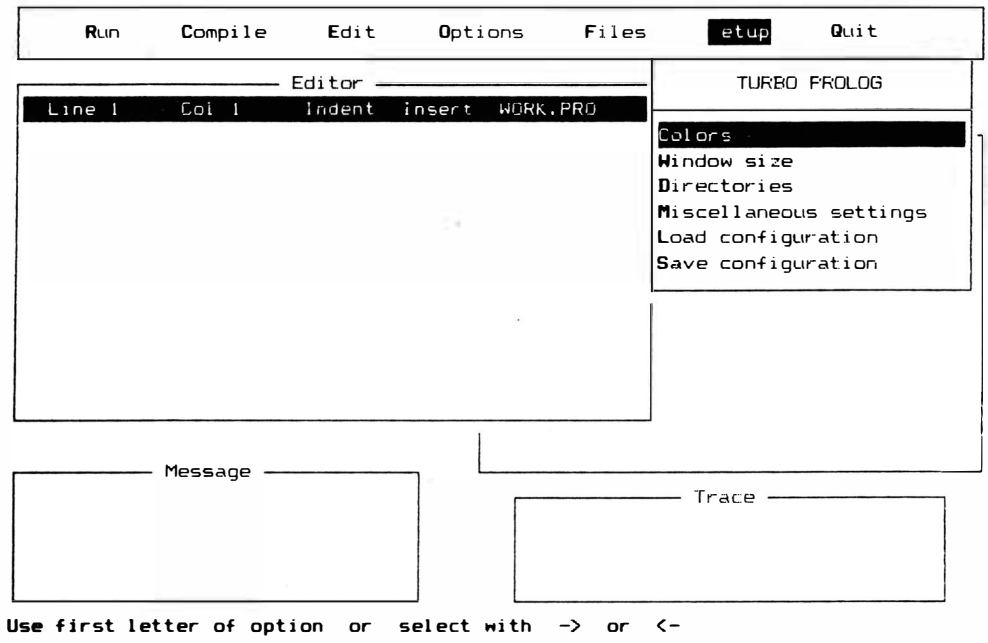


Figura 2-8 Submenu do comando Setup

COMANDO SETUP E MENU

O submenu abaixo do comando Setup permite que você inspecione, modifique temporariamente ou altere permanentemente a configuração de seu Turbo Prolog. A Figura 2-8 mostra o submenu que aparece assim que você digita o S no menu principal. As escolhas do submenu que você vai ver detalham como Turbo Prolog mostra e recebe informação.

COLORS (CORES)

Aqui, pela primeira vez, você vai encontrar outro menu por baixo de um submenu (veja Figura 2-9). Você pode utilizar este comando para definir as cores de primeiro e segundo plano de suas janelas de sistema. Mais uma vez, acione a tecla da primeira letra

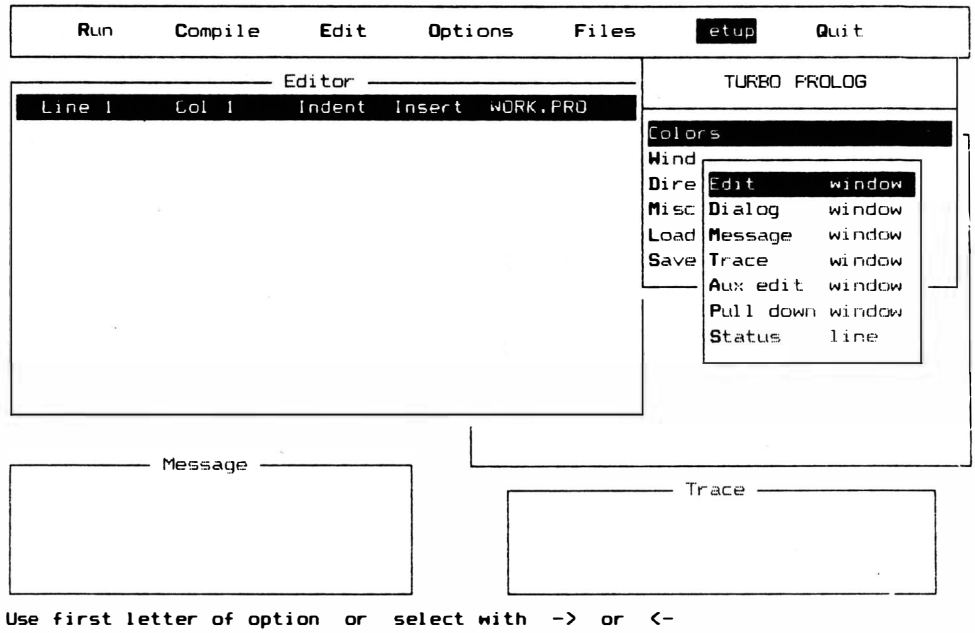


Figura 2-9 Menu do estabelecimento de Colors

da janela que deseja modificar. Depois olhe a linha inferior da tela, para ver os comandos de teclas para alteração das cores das janelas. As teclas de seta permitem que você percorra a seleção de cores. A Tabela 2-1 mostra que cores são representadas pelos atributos numéricos.

TAMANHO DA JANELA

Opere W no submenu do comando Setup, e estará pronto para alterar o tamanho, a posição ou formato de suas janelas Turbo Prolog. Escolha a janela com a qual quer trabalhar (veja a Figura 2-10) e depois utilize os comandos mostrados na parte inferior da tela para alterar o perímetro da janela. Utilize apenas as teclas de seta para alterar o tamanho da janela. Tecle CTRL ao mesmo tempo para fazer alterações maiores, com rapidez. Utilize as teclas de seta com a tecla SHIFT para mover a janela.

A informação não desaparecerá se uma janela estiver muito estreita para apresentá-la inteira. Em vez disto, ela passa para a linha seguinte da janela, ou fica escondida à direita. Você pode ver aquela informação alterando o tamanho da janela ou movimentando o cursor no interior da janela. As janelas podem também se sobrepor. A janela ativa é aquela que está sempre por cima em qualquer momento.

Da mesma forma como com as outras escolhas de Ambiente ("Setup"), você pode manter a sua nova configuração até deixar o Turbo Prolog, ou pode armazená-la permanentemente. A Figura 2-11 mostra um arranjo de janela que algumas vezes é útil. Veja as escolhas Load Configuration e Save Configuration nas seções seguintes para aprender como fazer as configurações sob medida.

Tabela 2-1 Valores de atributo de cor para telas monocromáticas e coloridas

Adaptador de vídeo monocromático

Caracteres (1º plano)	2º plano	Descrição	Valor do atributo
Preto	Preto	Vazio	0
Branco	Preto	Vídeo normal	7
Preto	Branco	Vídeo reverso	112

Notas: 1. Acrescente 1 para sublinhar caractere na cor de 1º plano.
2. Acrescente 8 para mudar de branco para branco de alta intensidade.
3. Acrescente 128 para fazer o caractere piscar.

Adaptador de vídeo gráfico/colorido

Cor de 1º plano	Atributo	Cor de 2º plano	Atributo
Preto	0	Preto	0
Cinza	8	Azul	16
Azul	1	Verde	32
Azul claro	9	Verde azulado	48
Verde	2	Vermelho	64
Verde claro	10	Rosa azulado	80
Verde azulado	3	Marron	96
Verde azulado claro	11		
Vermelho	4	Branco	112
Rosa	12		
Rosa azulado	5		
Rosa azulado claro	13		
Marron	6		
Amarelo	14		
Branco	7		
Branco de alta intensidade	15		

Notas: 1. Escolha uma cor de primeiro e uma de segundo plano e some seus números.
2. Acrescente 128 para fazer com que o objeto apresentado pisque.

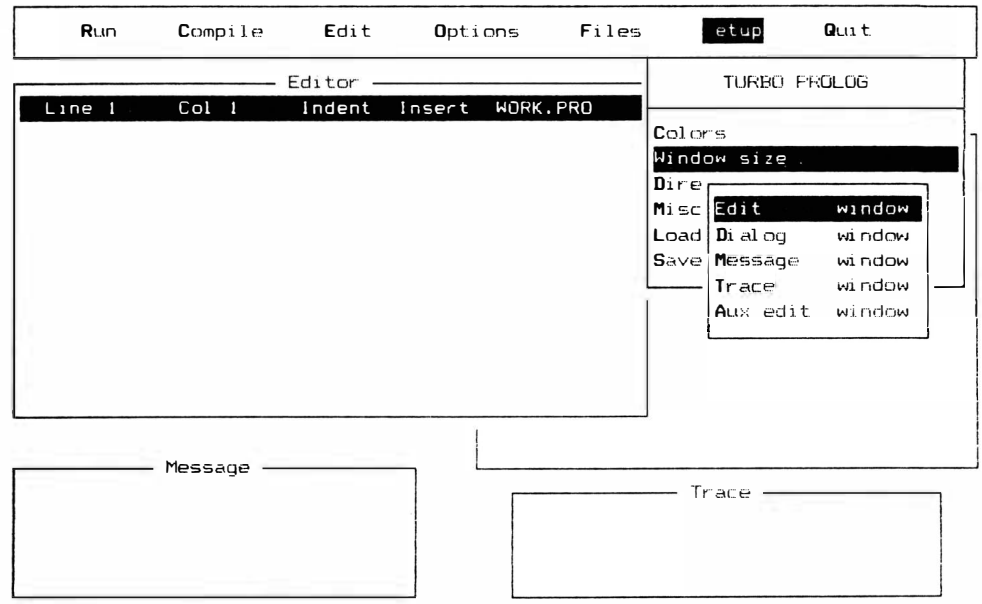


Figura 2-10 Menu para estabelecimento de tamanho de janela

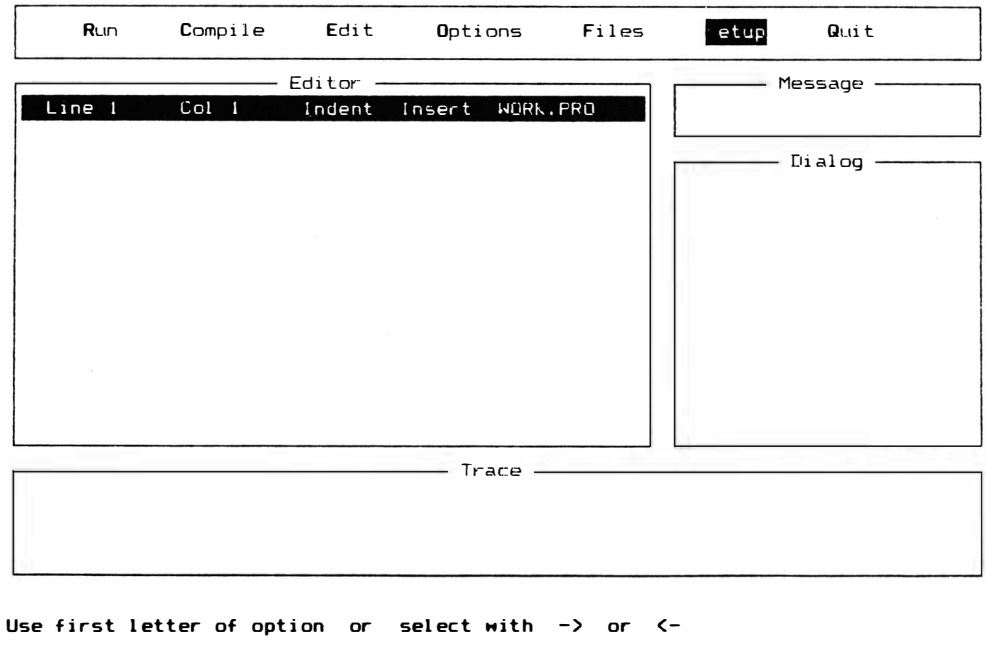


Figura 2-11 Um arranjo de janela sob medida

DIRETÓRIOS

Turbo Prolog precisa saber onde encontrar e colocar arquivos. Para evitar o trabalho de ter sempre que digitar todo o nome do percurso, ele armazena uma lista de acionadores de disco e nomes de percurso-padrão. O menu Directories (veja a Figura 2-12) permite que você inspecione ou modifique as condições de padrão atuais. Os diretórios são os seguintes:

- O diretório PRO é utilizado para toda manipulação de arquivo no submenu Files Command.
- O diretório OBJ é utilizado para arquivos .OBJ e .PRJ., e o diretório EXE é utilizado para arquivos .EXE. Ambos dizem respeito à compilação em disco e serão descritos posteriormente.
- O diretório TURBO informa onde o programa vai encontrar os arquivos PROLOG.EXE, PROLOG.ERR, PROLOG.HLP, PROLOG.SYS, e PROLOG.LIB. Estes são os arquivos principais de mensagem de erro, de auxílio, configuração de sistema e de biblioteca.

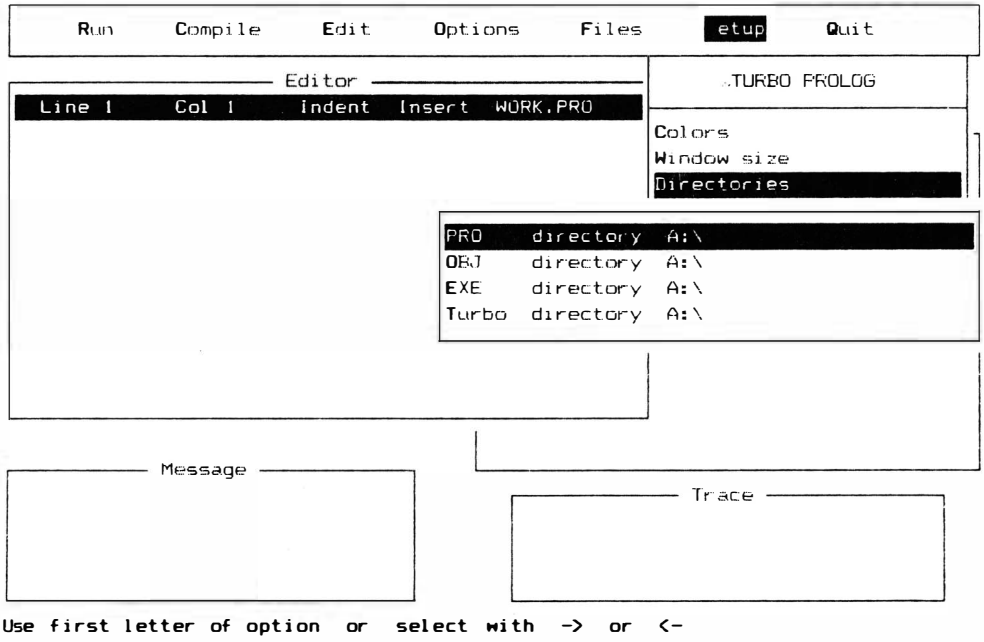


Figura 2-12 O menu de estabelecimento de Diretórios

CONFIGURAÇÕES DIVERSAS (MISCELLANEOUS SETTINGS)

Estas são as condições restantes (veja Figura 2-13). Toda vez que você operar I ou A alternará (ligá-lo se estiver desligado e desligá-lo se estiver ligado) o adaptador IBM-CGA ou a função de mensagem de Autocarregamento. Caso a escolha do adaptador IBM-CGA esteja ligada, uma sincronização especial será empregada para gerar uma melhor tela de vídeo colorida. Caso a mensagem de Autocarregamento esteja ligada, as mensagens de erro serão carregadas na memória ao ligarmos Prolog. Normalmente esta escolha será desligada, de modo que cada vez que ocorre um erro, a mensagem de erro tem que ser encontrada e carregada a partir do arquivo de disco PROLOG.ERR.

Caso tecle S, poderá alterar o tamanho da pilha, um recurso de software importante para tarefas avançadas de programação. O tamanho-padrão é 600 parágrafos de 16 bytes cada. A pilha pode ter um tamanho que varia de 600 a 4.000 parágrafos.

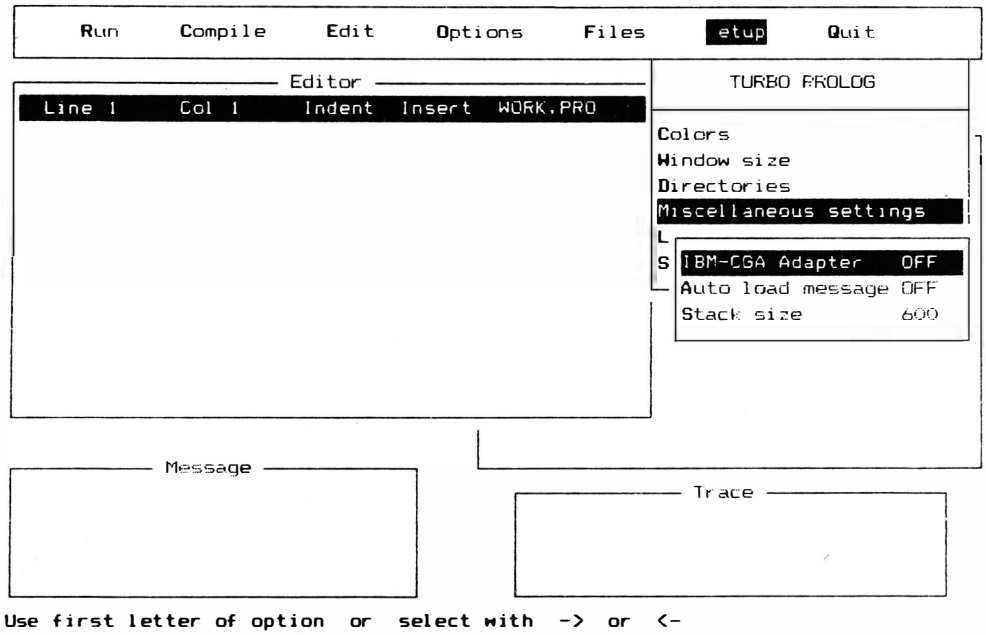


Figura 2-13 Menu de configurações diversas (Miscellaneous Settings)

GRAVAR CONFIGURAÇÃO

Assim que tiver alterado qualquer condição de configuração (utilizando o submenu do comando Setup), você pode gravar a sua nova configuração pela utilização do comando Save Configuration. Você deverá fornecer um nome de arquivo, e Turbo Prolog gravará os novos detalhes em um arquivo com uma extensão .SYS. O arquivo default, utilizado quando Turbo Prolog é inicializado, é PROLOG.SYS.

CARREGAR CONFIGURAÇÃO

Caso você tenha gravado uma configuração feita sob medida em um arquivo .SYS, conforme acabamos de descrever, poderá mudar para esta configuração usando o comando Load Configuration. Basta teclar L do submenu do comando Setup e depois dar o nome do arquivo .SYS que deseja utilizar. Seu Turbo Prolog será configurado com os valores daqueles arquivos. Para voltar ao original, basta carregar o arquivo PROLOG.SYS.

CAPÍTULO 3

O EDITOR

Um programa em Prolog começa como uma série de caracteres agrupados em palavras, frases e outras formas e armazenados em um arquivo denominado arquivo-fonte. Ao processo de escrever, cancelar, ou modificar os caracteres naquele arquivo chamamos editar o programa. A maioria dos computadores não sabe automaticamente como permitir que você edite texto. Eles precisam de um software "editor" especial para mostrar-lhe como aceitar, mostrar, mover e gravar caracteres.

Um editor é basicamente uma versão simples de um programa de processamento de palavras. Permitirá que você introduza, movimente cancele e de outra forma manipule caracteres. Muitos compiladores de linguagem de computadores antigos não incluíam um editor. Aqueles que tinham, freqüentemente apresentavam um editor com poucas funções e comandos obscuros.

Os projetistas de linguagem presumiram que se você tivesse muita programação para fazer, utilizaria seu próprio editor e processador de palavras para criar o arquivo-fonte. Assim que o fonte estivesse completo, você dispararia o compilador, dar-lhe-ia o nome de seu arquivo-fonte e esperaria o arquivo-objeto – o programa compilado e pronto para ser processado – aparecer.

Turbo Prolog tem seu editor próprio que contém a maioria das funções de processamento de palavras que você precisará para criar ou trabalhar em um arquivo-fonte Turbo Prolog. Para utilizar aquele editor com eficiência, será preciso memorizar algumas combinações de teclas que acionam comandos de edição.

Este capítulo informa-lhe como utilizar o Editor Turbo Prolog. Caso você tenha experiência em utilizar outros programas com comandos de edição similares (WordStar, MultiMate, Turbo Pascal ou SideKick, por exemplo) sentir-se-á em casa com o Editor

Turbo Prolog.

Volte para este capítulo mais tarde caso se sinta perdido com o Editor. O Editor Turbo Prolog não inclui todos os comandos de edição que você encontra em outros programas, mas tem os principais.

ENTRANDO NO EDITOR

Conforme anteriormente mencionado, assim que o Turbo Prolog estiver em processamento com o menu principal à sua frente, bastará teclar E para entrar no Editor. A janela Editor (mostrada no canto superior esquerdo da tela na Figura 3-1) será iluminada, e o cursor aparecerá na janela para mostrar-lhe onde sua digitação terá efeito.

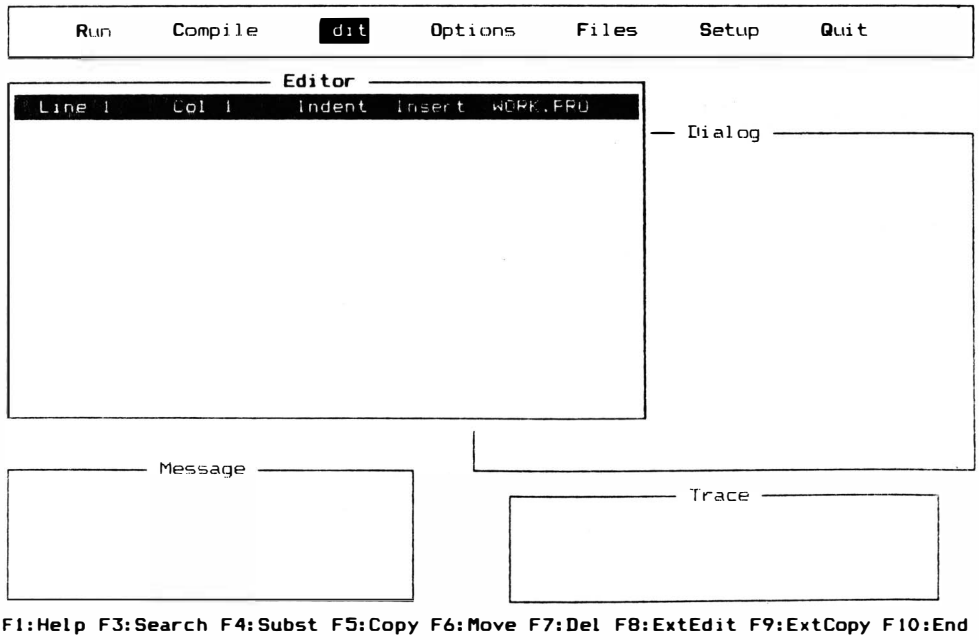


Figura 3-1 A janela Editor na tela principal

OBTENDO AJUDA

Caso as tabelas deste livro não estejam à mão, você poderá teclar F1 (na extremidade esquerda do teclado do PC IBM) para obter a informação de ajuda (Help) a respeito de edição. Os subtítulos funções de bloco, posição, teclas de cancelamento e status vão mostrar uma lista de comandos relevantes. A informação Help mostrará uma grande janela de edição, cor, predicados-padrões e predicados especiais pelos quais você pode passar com as teclas de seta. Os títulos cópia externa, edição auxiliar e edição rápida colocam-no nestas funções. A Figura 3-2, o menu principal para informação de auxílio, mostram um exemplo da ajuda de Edição.

PREPARANDO PARA INTRODUZIR O TEXTO

Se já existir algum texto na janela, cancele-o antes de continuar. Existem diversas maneiras de fazer isto, mas uma das mais rápidas (especialmente ao tratar com uma grande quantidade de texto) é deixar o Editor e utilizar uma das opções no submenu Files.

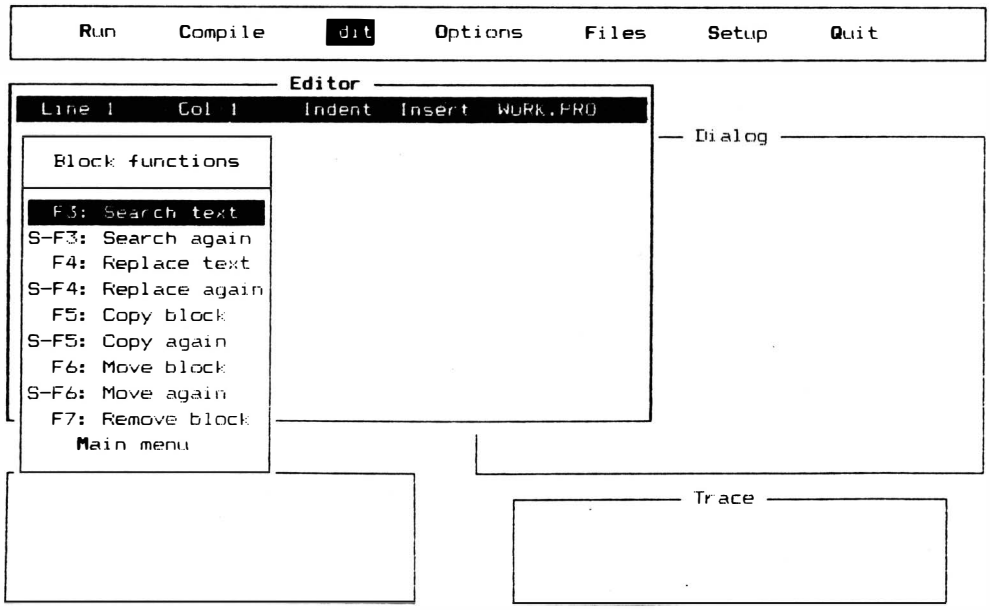


Figura 3-2 Menu principal de auxílio de edição

SAINDO DO EDITOR

Como regra geral no Turbo Prolog, quando você quiser sair de uma janela e voltar ao menu principal, tecle ESC. Isto se aplica também para deixar o editor.

Tecle ESC. Isto o levará de volta ao menu principal. Depois, tecle F, que lhe mostrará o submenu Files. Tecle Z para selecionar a opção Zap File in Editor (introduza arquivo no editor) (veja a Figura 3-3). Ser-lhe-á inquerido se você realmente quer isto. Você o quer, de modo que tecle Y de "sim". Agora você tem uma janela Editor limpa. Tecle ESC (para ter certeza de que está no menu principal), e tecle E para voltar ao Editor.

FUNDAMENTOS

Entrar, cancelar, salvar e carregar texto são os fundamentos absolutos da edição.

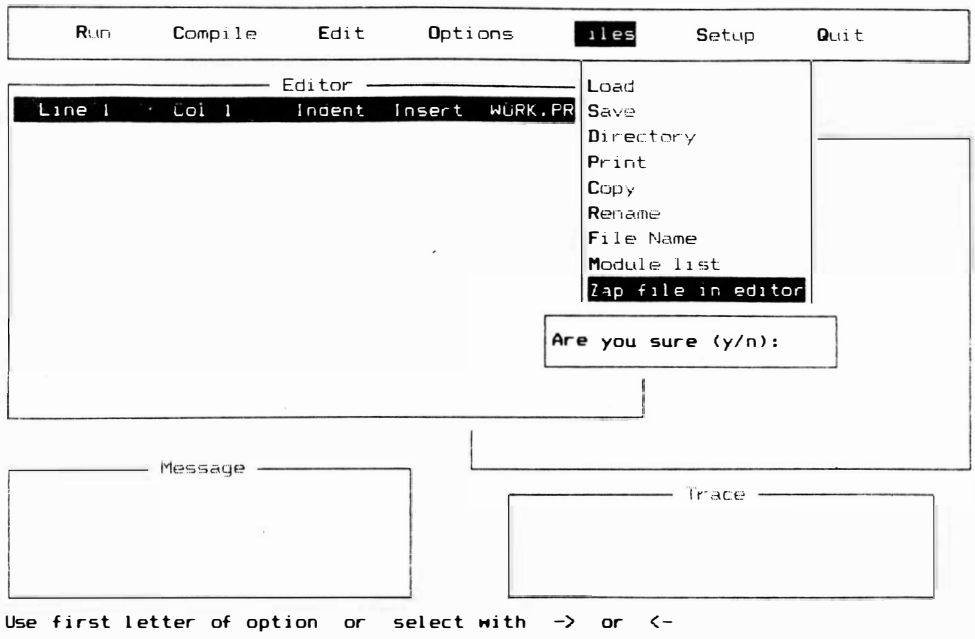


Figura 3-3 Submenu de Files com opção Zap File in Editor (introduza arquivo no Editor)

INTRODUZINDO TEXTO

Comece digitando o seguinte:

Jackie Robinson

Digite da mesma forma como faria com uma máquina de escrever, utilizando as teclas SHIFT para a obtenção de letras maiúsculas. Não se preocupe em teclar RETURN ainda. Caso já o tenha feito, tecle BACKSPACE uma vez (é a seta que aponta para a esquerda, acima da tecla RETURN). O cursor se move à medida que digita letras, ficando ao fim logo depois do "n". Esta é a maneira mais simples de acrescentar texto ao seu arquivo. Você pode ainda acrescentar texto de diversas outras formas: copiando blocos, importando texto de outros programas, ou carregando texto do disco. Não acione ainda outra tecla.

CANCELANDO TEXTO

Existem duas teclas que cancelam um caractere por vez: a tecla BACKSPACE e a tecla DEL (cancelar), que operam de forma diferente.

TECLA BACKSPACE

Pressione esta tecla para cancelar o caractere logo à esquerda do cursor. Tente-o quando o cursor estiver no fim de

Jackie Robinson

e verá

Jackie Robinso

como resultado. Além de cancelar um caractere, o cursor se move para trás, ficando logo depois do "o". Digite BACKSPACE algumas vezes, depois digite a parte faltante do nome novamente.

TECLA DEL

Com o nome completo no lugar novamente, pressione a tecla de seta para a esquerda (no teclado numérico) três vezes. O cursor ficará em baixo do "S". Tente digitar a tecla DEL (na parte inferior do teclado numérico). Você verá

Jackie Robinson

mudar para

Jackie Robinon

A tecla DEL cancela o caractere acima do cursor. Esta diferença entre as funções de BACKSPACE e DEL é usada em muitos programas de computação do PC IBM. Tente-o algumas vezes para se acostumar com ela.

GRAVANDO

Quando você editar texto de programa, ele será constantemente guardado na memória RAM do computador. Entretanto, você deverá guardá-lo periodicamente em disco. A não ser que você tenha algum tipo de alimentação ininterrupta, poderá perder todo trabalho caso ocorra uma queda de força ou oscilação em seu computador. Isto pode ocorrer em diversas formas, desde um carro batendo em um poste de energia até uma criança desligando a tomada de seu computador. Gravar é uma coisa tão rápida, vale a pena fazê-lo a cada cinco minutos quando estiver fazendo muitas alterações ou a cada quinze quando as alterações forem menores.

Para gravar o arquivo que está na janela Editor, precisará utilizar a opção Save no submenu do comando Files. Caso esteaja no Editor, tecle ESC para voltar ao menu principal. Depois tecle S para a opção Save. Ser-lhe-á solicitado um nome de arquivo. O nome deverá ser como os dos arquivos em DOS: uma a oito letras com uma extensão opcional com um ponto e uma a três letras. Preceda o nome de seu arquivo com uma indicação de acionador de disco que vai colocá-lo em seu disco Programas-Exemplo. Caso responda:

b: temp

à pergunta do nome de arquivo, seu arquivo será armazenado como TEMP.PRO (o .PRO é automaticamente acrescentado pelo Turbo Prolog) no acionador B.

CARREGANDO TEXTO DO DISCO

Se você quer voltar amanhã para trabalhar no arquivo feito hoje, terá de saber como carregá-lo novamente na janela Editor do disco. Pelo fato de ainda ter um arquivo no Editor, primeiro cancele tudo no Editor. Utilize o comando Zap File in Editor do submenu File conforme descrito anteriormente em "Saindo do Editor", neste mesmo capítulo.

Uma vez feito isto, você deverá estar de volta no menu principal. (Tecle ESC caso queira ter certeza.) Tecle F do submenu Files e L para o comando Load. Digite o nome do arquivo que deseja, como a seguir:

```
b: temp
```

Depois, tecle RETURN. Seu arquivo vai aparecer na janela Editor. Tecle ESC para voltar ao menu principal e depois E de Editor. Agora você está de volta à janela Editor, pronto para fazer algumas alterações.

MOVIMENTO DO CURSOR

Não importa o que existe na janela Editor, utilize o comando Zap File in Editor do submenu Files para seu cancelamento. Depois, volte ao Editor como acabamos de descrever. Digite

```
domains
```

e RETURN. Ele colocará o cursor no começo da próxima linha. Depois digite as seguintes linhas:

```
predicates

gola

clauses
```

Agora que tem este texto estranho em seu Editor (conforme mostrado na Figura 3-4), precisará corrigir de "gola" para "goal". Existem maneiras melhores de se fazer isto do que cancelar caracteres até o erro. Você pode mover o cursor até o problema e fazer a correção ali.

Existem diversos comandos para movimentação do cursor. São divididos nos dois seguintes grupos com muitas funções iguais: comandos de teclado numérico e comandos de controle de tecla ou teclas de função.

O teclado numérico à direita do teclado inclui um conjunto de comandos de movimentação do cursor descritos na própria tecla. As setas (para cima, para baixo, para a direita e para a esquerda nas teclas 8, 2, 6 e 4) movimentam o cursor um caractere por vez. Faça uma tentativa. As teclas PGUP e PGDN movimentam uma página inteira de texto,

rolando-a se houver necessidade (você não tem ainda texto suficiente na janela para tentar estes comandos). HOME e END levam o cursor ao começo da linha ou ao fim da linha.

Da mesma forma como em muitos programas de software, o movimento do cursor durante a edição em Turbo Prolog pode ser conseguido pela utilização de teclas de comando. Teclas de comando são teclas de função programadas para fornecer determinados comandos, ou são seqüências de teclas de controle que chamam comandos específicos. Depois de dominar o uso das seqüências de teclas de controle, alguns usuários poderão achar mais fácil utilizar a tecla única de função correspondente a um comando específico. Para acionar comandos com teclas de controle, terá que manter pressionada a tecla CTRL (controle) e alguma outra tecla ao mesmo tempo. A Tabela 3-1 relaciona os comandos das teclas de controle.

Para movimentação do cursor existem teclas de comando com formato em losango. CTRL-S movimenta o cursor um caractere para a esquerda, e CTRL-D movimenta para a direita. CTRL-E movimenta o cursor uma linha para cima e CTRL-X move-o para baixo. Estes quatro comandos se comportam exatamente da mesma forma que as quatro teclas de comando de seta do teclado numérico. Utilize os comandos das teclas de controle

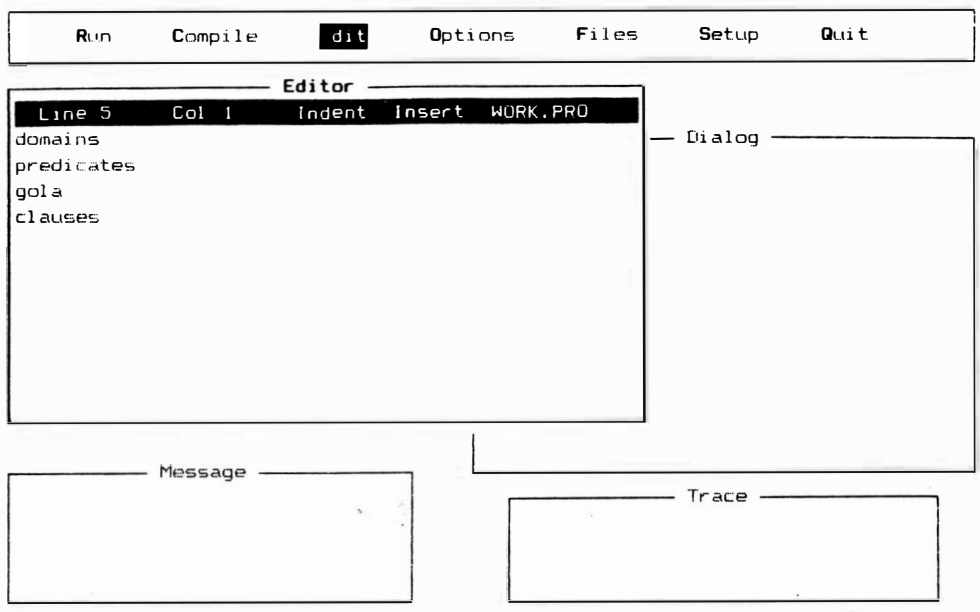


Figura 3-4 Texto no Editor

para movimentar até o "1" de "gola" no texto já colocado. Cancele o "1", mova o cursor para depois do "a" e acrescente um "1". Agora você tem a escrita correta de um programa Turbo Prolog, Grave este arquivo com o nome "Outline" (Sumário) no acionador B.

PGUP e PGDN também têm teclas de controle equivalentes. CTRL-R move o cursor uma página para a frente e CTRL-C uma para trás. Algumas vezes é necessário acrescentar mais uma tecla a um comando de tecla de controle. Tecle CTRL e Q ao mesmo tempo, e depois uma das teclas anteriormente mencionadas para a obtenção de um comando combinado "rápido" (que faz o que vários comandos individuais fariam). Você não precisa continuar a manter a tecla CTRL pressionada quando pressiona a tecla final. Por exemplo, tecle CTRL-Q e depois S para ir até o começo de uma linha ou CTRL-Q e depois D para ir até o fim. Estes comandos são semelhantes a HOME e END.CTRL-Q-R vai ao início do arquivo – o primeiro caractere na janela Editor .CTRL-Q-C vai ao fim do arquivo. Você pode também chegar a estes locais utilizando CTRL-PGUP e CTRL-PGDN.

Finalmente, você pode movimentar o cursor uma palavra por vez em vez de um caractere por vez, utilizando CTRL-F (uma palavra à direita) e CTRL-A (uma palavra à esquerda). Combinando a tecla CTRL com as setas para a esquerda e para a direita do teclado numérico tem-se o mesmo efeito.

Tabela 3-1 Comandos de edição da tecla de controle

Help	
Informação de ajuda	F1 (sempre)
Teclas de cancelamento	
Cancele o caractere sob o cursor	DEL ou CTRL-G
Cancele o caractere à esquerda do cursor	BACKSPACE
Cancele a palavra à direita do cursor	CTRL-T
Cancele a linha do cursor	CTRL-Y ou CTRL-BACKSPACE
Movimentos do cursor	
Subindo uma linha	Seta-para-cima ou CTRL-E
Descendo uma linha	Seta-para-baixo ou CTRL-X
Um caractere à esquerda	Seta-para-esquerda ou CTRL-S
Um caractere à direita	Seta-para-direita ou CTRL-D
Uma palavra à esquerda	CTRL-seta-esquerda ou CTRL-A
Uma palavra à direita	CTRL-seta-direita ou CTRL-F

Ao começo da linha do cursor	**HOME ou CTRL-Q-S**
Ao fim da linha do cursor	**END ou CTRL-Q-D**
Uma página para a frente	PGUP ou CTLR-R
Uma página para trás	PGDN ou CTRL-C
Começo de bloco marcado	CTRL-Q-B
Fim do bloco marcado	CTRL-Q-K
Começo (topo) do arquivo	CTRL-PGUP
Fim do arquivo	CTRL-PGDN
Vá a uma determinada linha	F2, digite o número da linha e pressione F2 novamente
Mostre o número da linha do cursor	SHIFT-F2 (apenas na versão 1.0)
Ligue indentação automática	CTRL-Q-I (operar de novo esta seqüência desliga a indentação)

Funções de bloco

Marque começo de bloco	CTRL-K-B (ou tecla de função especial)
Marque fim de bloco	CTRL-K-K (ou tecla de função especial)
Esconda marcações de bloco	CTRL-K-H (operar de novo esta seqüência reapresenta as marcações)
Cancele bloco marcado	CTRL-K-Y ou F7
Mova bloco marcado à posição do cursor	CTRL-K-V ou F6
Copie o bloco marcado na posição do cursor	CTRL-K-C ou F5
Repita cópia do bloco marcado	SHIFT-F5
Leia bloco do disco na posição do cursor	CTRL-K-R ou F9
Grave bloco marcado no disco	CTRL-K-W

Entrada de texto

Entre na janela Editor	E
Entre na janela Editor auxiliar	F8
Deixe Editor ou Editor auxiliar (fim da edição)	F10 ou ESC
Alterne entre inserção e sobreposição	CTRL-V ou INS

Pesquisa e Substituição

Procure uma determinada cadeia	F3 ou CTRL-Q-F
Repita a última pesquisa	SHIFT-F3 ou CTRL-L
Procure e depois substitua uma determinada cadeia	F4 ou CTRL-Q-A
Repita a última pesquisa e depois substitua	SHIFT-F4 ou CTRL-L

INDO PARA UMA LINHA DETERMINADA

As linhas de um arquivo-fonte do Turbo Prolog são numeradas do começo ao fim. Em versões de Turbo Prolog diferentes da versão 1.0, a posição atual do cursor é identificada pelo especificador Line/Col na linha de status da janela Editor. Para saber em que linha o cursor se encontra na versão 1.0, você pode pressionar a tecla SHIFT e a tecla de função F2 ao mesmo tempo. Para mover o cursor até uma determinada linha, digite F2. Depois digite o número da linha que deseja atingir e novamente F2.

MODOS INSERÇÃO E SOBREPOSIÇÃO

O Editor começa com o modo de inserção ligado. Isto significa que se você coloca o cursor sobre caracteres na tela e começa a digitar, os novos caracteres são inseridos no texto anterior sem cancelar aquele texto. Você pode desligar o modo Inserção – também conhecido como colocar o Editor no modo Sobreposição – operando INS (Inserção) (próxima à tecla DEL) ou CTRL-V. Quando o Editor estiver no modo Sobreposição, tudo o que é digitado elimina aquilo que antes estava na posição do cursor.

Para ver os efeitos dos modos Inserção e Sobreposição, veja as Figuras 3-5 e 3-6. Na Figura 3-5, o Editor estava no modo Inserção quando o cursor foi posicionado no começo da palavra "predicates" e "phillip" foi digitado. Na Figura 3-6, o Editor estava no modo Sobreposição quando ela foi feita.

CANCELAMENTO DE CARACTERE

A utilização de BACKSPACE e DEL e suas maneiras de liquidar uma letra foi discutida anteriormente. Você pode também cancelar o caractere logo acima do cursor, mesmo que seja um espaço vazio, pressionando CTRL-G.

CANCELAMENTO DE LINHA E PALAVRA

Ao pressionar CTRL-Y ou CTRL-BACKSPACE você cancelará a linha inteira em que se encontra o cursor. Caso queira eliminar apenas parte da linha, tente CTRL-Q-Y para cancelar todo texto à direita (do cursor até o fim da linha).

COMANDOS AVANÇADOS

Você passou da edição de caracteres à edição de linhas. É possível fazer também com que o computador edite blocos inteiros de texto e procure determinadas partes de um

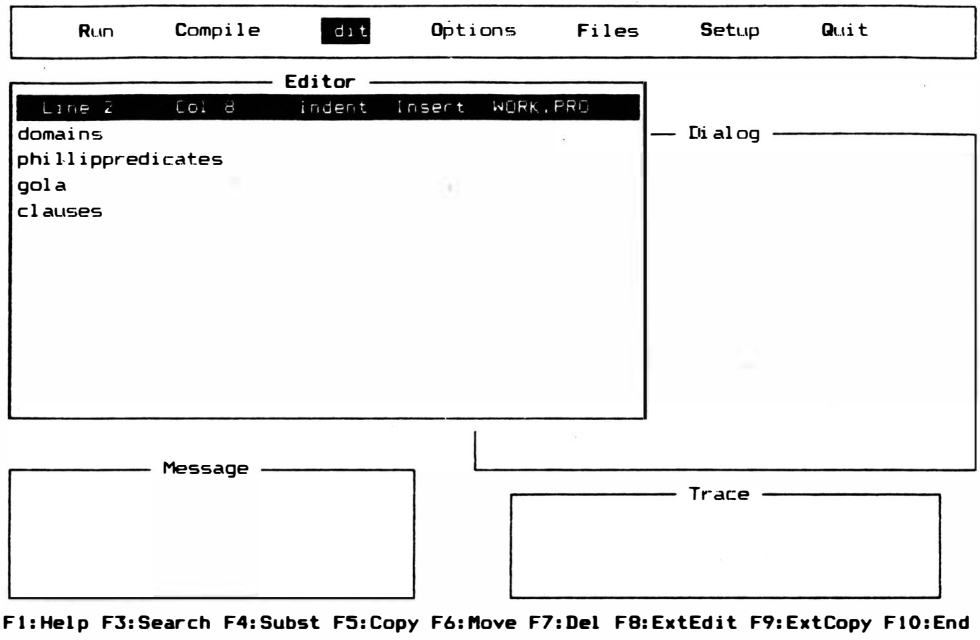


Figura 3-5 Editor no modo inserção

texto. Estas operações são muito fáceis de serem feitas e são a razão principal porque o processamento de palavras computadorizado é melhor do que a utilização da máquina de escrever ou o lápis-e-papel no processamento de palavras.

OPERAÇÕES COM BLOCOS

Você executa operações com blocos de texto identificando o bloco e depois informando ao Editor o que fazer com ele. Digite ou carregue o arquivo Outline descrito anteriormente neste capítulo (é mostrado na Figura 3-4).

MARCANDO O BLOCO

Mova o cursor para "g" em "goals". Para marcar o começo do bloco, pressione CTRL-K-B. (A combinação CTRL-K é utilizada para muitos comandos de bloco.) Depois, mova o cursor para a letra "s" em "clauses". Marque o fim do bloco digitando CTRL-K-K. Você verá o bloco marcado mudar de intensidade na tela. (Caso não, ajuste o

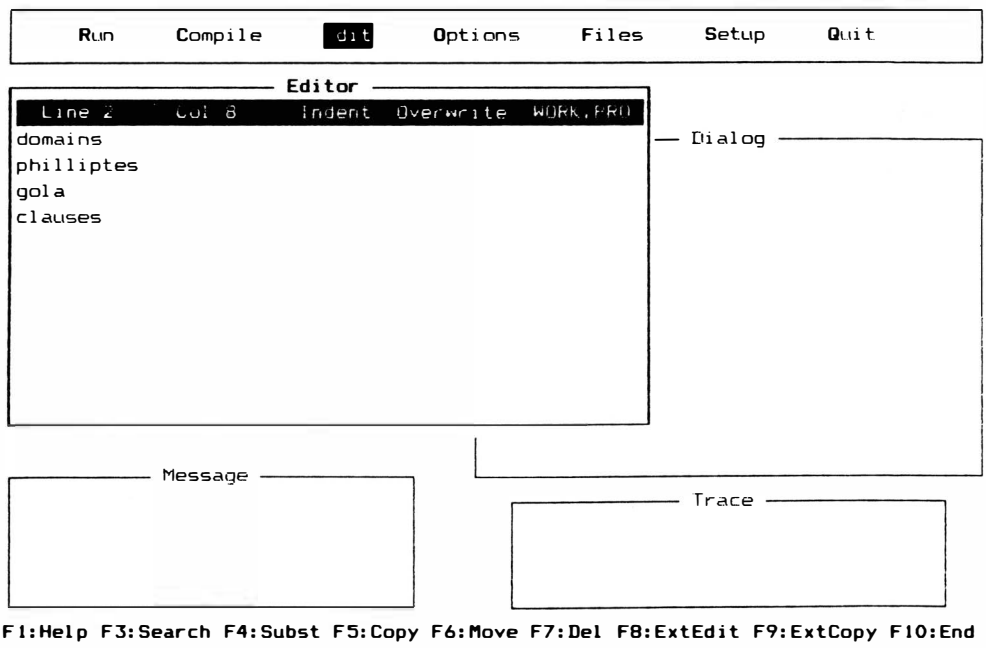


Figura 3-6 Editor no modo Sobreposição

contraste e brilho da tela até que aconteça.) Assim que o bloco estiver identificado ou marcado, poderá movê-lo, copiá-lo ou cancelá-lo.

Para movimentá-lo, mova o cursor ao local na janela do Editor onde você quer que o bloco inicie – talvez no "d" de "domains". Depois digite CTRL-K-V ou a tecla de função F6 (veja adiante). O bloco desaparece de sua posição antiga para aparecer na nova. Para copiar o bloco, mova o cursor de volta à sua posição original (depois do "s" de "predicates") e digite CTRL-K-C ou a tecla de função F5 (veja a seguir). Você verá uma segunda versão do bloco aparecer neste ponto. Cancelar o bloco é igualmente fácil. Utilize CTRL-K-Y ou F7 a qualquer momento para cancelar um bloco marcado.

Caso deseje, pode utilizar as teclas de função especial tanto para marcar os blocos como para manipulá-los. Por exemplo, para movimentar um bloco, poderá mover o cursor para o começo do bloco, teclar F6 e depois mover o cursor para o final e teclar F6. Finalmente, mova o cursor para a nova posição do bloco e tecle F6 uma terceira vez. Isto também funciona com a tecla F5 (para cópia) e a tecla F7 (para cancelamento).

Enquanto um bloco está marcado e iluminado, você pode utilizar CTRL-Q-B para ir ao começo do bloco ou CTRL-Q-K para ir até o fim do bloco.

CTRL-K-H vai alternar a intensidade do bloco marcado, isto é, vai passar um bloco iluminado para não-iluminado e um não-iluminado para iluminado. Um bloco marcado e iluminado poderá ser copiado, movido ou cancelado. Um bloco marcado não-iluminado não poderá. Os comandos para ir ao fim do bloco e ao começo do bloco funcionam em blocos iluminados e não-iluminados.

LENDO UM BLOCO DO DISCO

Isto poderá parecer uma operação obscura, mas muito útil em programação. Você poderá querer um pedaço de código de programa de outro arquivo no arquivo com o qual está trabalhando. Mova o cursor para o local no qual quer colocar o bloco. Caso não esteja seguro, pode colocar o cursor no fim do arquivo. Digite CTRL-K-R ou F9. A opção Load do submenu Files aparece. Digite o nome do arquivo de onde quer tirar um bloco. O arquivo vai aparecer em uma janela (como, por exemplo, na Figura 3-7). Encontre o bloco que deseja pela movimentação do cursor com comandos de Editor, e depois marque o começo e fim do bloco com CTRL-K-B (ou F9) e CTRL-K-K (ou F9).

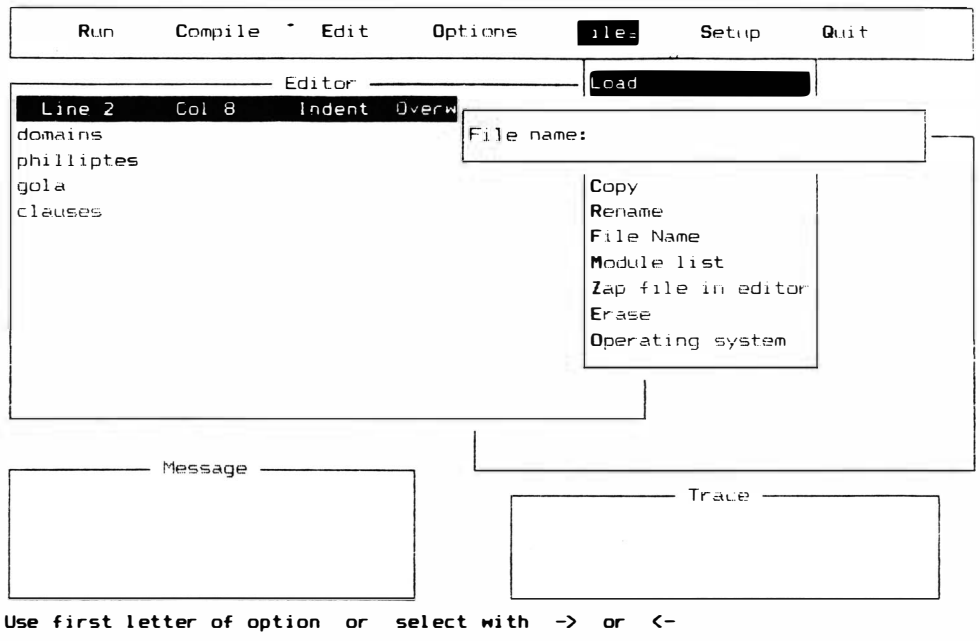


Figura 3-7 Carregando um arquivo em disco em um programa

COMANDOS DE PESQUISA E SUBSTITUIÇÃO

O Editor contém diversos comandos que lhe permitem encontrar ou substituir uma determinada parte de seu arquivo. Uma cadeia de caracteres é simplesmente uma série de caracteres em uma linha. Turbo Prolog pode procurar qualquer cadeia que você determine que tenha no máximo 25 caracteres de comprimento e pode substituí-la. Ele pode levar o cursor à cadeia ou substituí-la por qualquer outra cadeia de até 25 caracteres.

PROCURA

Digite F3 ou CTRL-Q-F. Em seguida, Prolog lhe perguntará qual cadeia de caracteres deseja encontrar. Digite a cadeia e depois execute o comando pela operação de F3 (caso o comando tenha sido iniciado com ele) ou RETURN (caso tenha utilizado CTRL-Q-F). O cursor vai aparecer na primeira ocorrência do arquivo, da cadeia escolhida. Caso queira encontrar a ocorrência seguinte daquela cadeia no arquivo, opere SHIFT e F3 ou CTRL-L.

SUBSTITUIR

Para substituir uma cadeia de caracteres, digite F4 ou CTRL-Q-A. Depois digite a cadeia que deseja substituir. Em seguida digite F4 (caso tenha iniciado a operação com ele) ou RETURN (caso tenha utilizado CTRL-Q-A para começar). Depois, digite a nova cadeia e em seguida F4 ou RETURN (como feito anteriormente).

Neste ponto, você terá que dizer se deseja uma procura ou substituição local ou global (veja Figura 3-8). A opção substitui apenas a primeira ocorrência da cadeia anterior pela nova cadeia. A opção Global substituirá todas as ocorrências no arquivo da cadeia anterior pela nova.

Em seguida você terá de dizer se deseja verificar cada substituição. Caso apenas queira que seja feito rapidamente, tecle N. Caso queira, no entanto, verificar cada ocorrência, tecle Y. Neste caso, o computador achará uma ocorrência da cadeia anterior e lhe pedirá um "sim" ou "não" (Y/N). Caso queira que esta ocorrência em especial seja substituída, tecle Y. Ela será substituída e a próxima ocorrência da cadeia anterior será encontrada.

Caso não queira que esta ocorrência em particular seja substituída, responda N, e o Editor vai conservá-la e procurar a próxima ocorrência. Com uma procura e substituição global, isto continua até o final do arquivo ou até que você pressione a tecla ESC. ESC vai interromper as operações de pesquisa e substituição em qualquer ponto.

Para repetir uma operação de procura e substituição, pressione as teclas SHIFT e CTRL-L.

INDENTAÇÃO AUTOMÁTICA

Você observará, quando estiver praticando a edição de um arquivo, que o cursor nem sempre volta à coluna mais à esquerda da janela Editor. Quando a característica Auto Indent estiver ligada e você digitar a tecla RETURN no fim de uma linha, o cursor voltará à esquerda até a primeira coluna da linha recém-terminada. Isto ajuda muito os programadores, porque muitos arquivos de texto de programa são mais fáceis de serem lidos e melhor organizados quando se utiliza indentação para a identificação de subseções, tanto quanto se faz em esquemas.

Para se mover até outra coluna, basta utilizar os comandos de movimentação do cursor. Para desligar a opção Indentação Automática (Auto Indent), digite CTRL-Q-I. Digite estas teclas novamente quando desejar ligá-lo novamente.

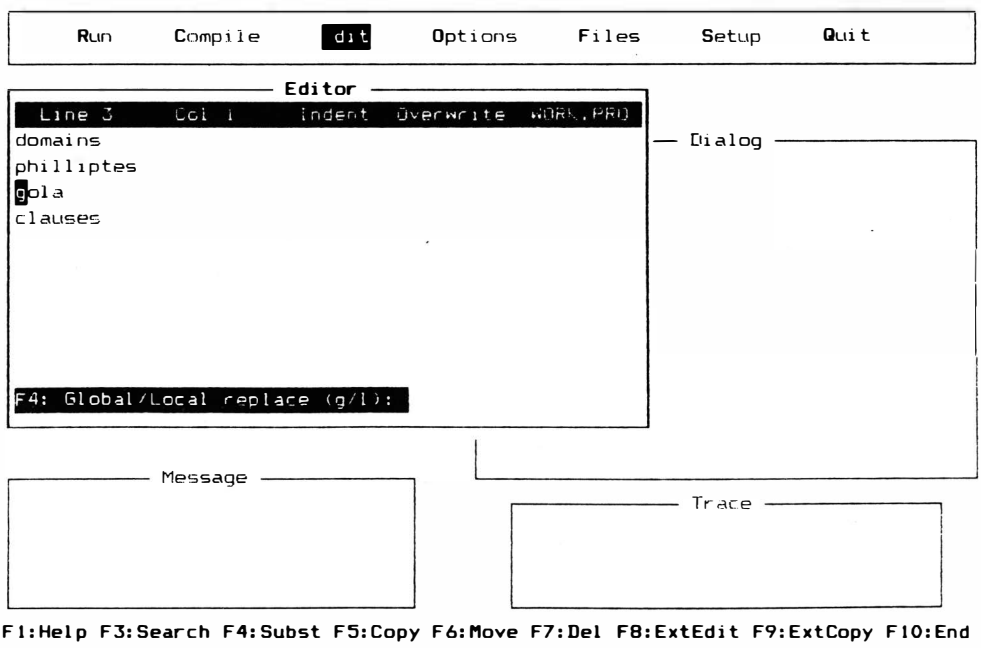


Figura 3-8 Pesquisa e substituição global

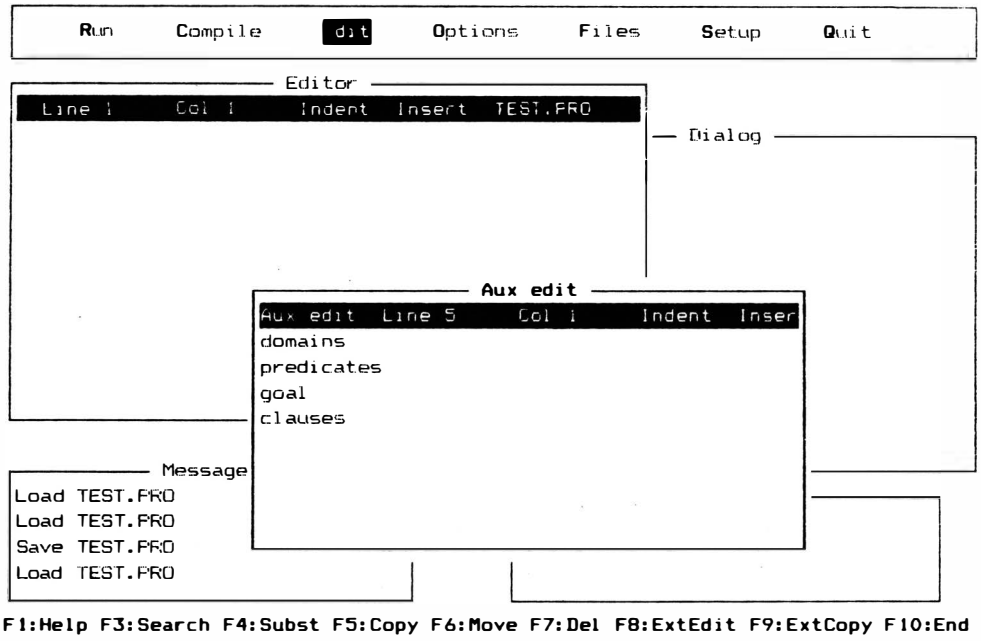


Figura 3-9 Exemplo da janela Editor auxiliar

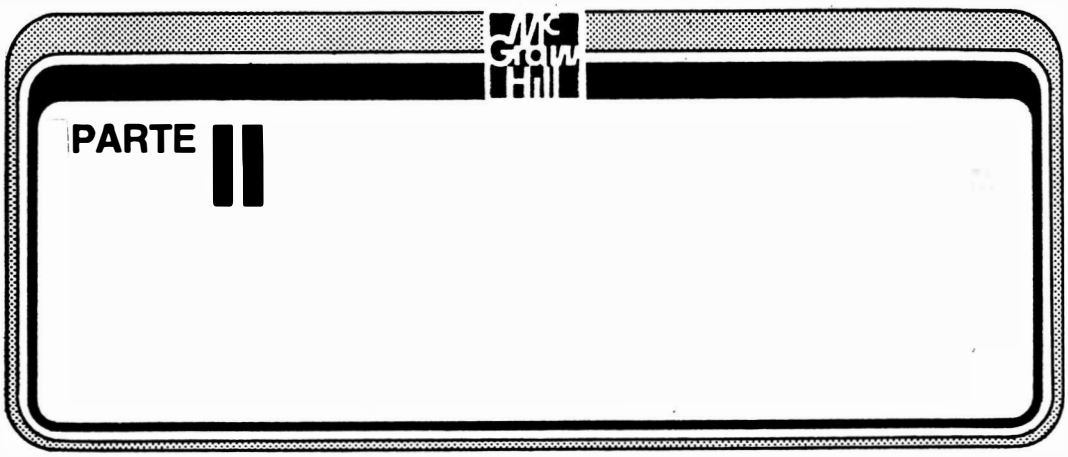
EDITOR AUXILIAR

Turbo Prolog vai fornecer-lhe outra janela de edição para aquelas ocasiões onde quiser modificar algum texto antes de movê-lo para o Editor principal. Pressione F8 para ver esta janela (veja a Figura 3-9). Você poderá utilizar os mesmos comandos de edição nesta janela.

A janela auxiliar de edição pode mesmo ser dimensionada, da mesma forma que a outra janela. Isto foi explicado no Capítulo 2. Dê uma olhada na escolha Window Size do submenu Setup. Ao terminar o trabalho na janela do Editor auxiliar, tecle F10.

TERMINANDO UMA EDIÇÃO

Ao terminar a edição de um arquivo, poderá digitar ESC para voltar ao menu principal, ou F10. Se estivesse editando o arquivo porque tentou fazer sua compilação e Turbo Prolog lhe informou que havia um erro de sintaxe para corrigir, F10 não iria apenas sair do Editor, mas também tentar compilar automaticamente o programa recém-editado.



Todo programa em Turbo Prolog é construído em torno de alguns conceitos básicos. A Parte II explora estes conceitos, começando com fatos e regras. Depois, passa para tópicos como retrocesso, aritmética e estruturas (functores, listas e cadeias de caracteres). Ao terminar esta parte, você estará escrevendo programas Turbo Prolog muito poderosos.

CAPÍTULO 4

FATOS: OBJETOS E RELACIONAMENTOS

Caso você já se sinta à vonade com as janelas, os menus e o Editor do Turbo Prolog estará pronto para a linguagem em si. Este capítulo cuidará do primeiro passo na utilização do Prolog: entender sentenças sobre objetos e relacionamentos.

Não acredite no mito de que Prolog é útil apenas para Inteligência Artificial (IA) como linguagens naturais e sistemas especialistas. Você pode fazer com Prolog virtualmente tudo o que pode fazer com qualquer outra linguagem de programação, incluindo jogos, contabilidade, gráficos e simulação. Não é sempre a linguagem mais prática ou eficiente para algumas aplicações, mas consegue realizá-las. Estar aprendendo Prolog não significa estar aprendendo IA. Prolog é uma linguagem popular para muitas tarefas e aplicações de IA, mas processamento de linguagens naturais e sistemas especialistas, por exemplo, poderiam ser construídos em BASIC ou mesmo COBOL. Prolog não é uma chave mágica. Apenas torna algumas destas tarefas de programação mais fáceis e rápidas.

Seguramente você não deve ignorar este capítulo. Se você já tem conhecimento sobre objetos, relacionamentos, fatos, predicados, argumentos e metas do Prolog, poderá apenas estudá-lo rapidamente e fixar-se nas seções denominadas "Sintaxe", "Predicados" e "Domínios". A sintaxe do Turbo Prolog é muito parecida com a versão Prolog de Clocksin e Mellish e inclui controles semelhantes aos do Turbo Pascal.

PROCEDURAL CONTRA DECLARATIVO

A maioria das linguagens que têm sido utilizadas em microcomputadores – BASIC, Pascal, Modula-2 – é procedural. Estas linguagens permitem que o programador in-

forme ao computador o que fazer, passo a passo, procedimento por procedimento, para chegar a uma conclusão ou executar uma função.

Prolog não é procedural. É *declarativa*. Caso você não tenha utilizado uma linguagem declarativa antes, o aprendizado de Prolog lhe dará mais trabalho do que apenas aprender mais uma linguagem procedural. Mas o esforço, sem dúvida, valerá a pena, porque uma linguagem declarativa faz muito do trabalho de programação para você. Enquanto a linguagem procedural exige que você introduza a receita e os ingredientes, uma linguagem declarativa pede apenas os ingredientes e a finalidade. Você declara a situação com a qual quer trabalhar e declara onde quer chegar. A linguagem em si – o compilador ou interpretador – faz a maior parte do trabalho de decidir como atingir a meta.

Por exemplo, se você é um programador BASIC que deseja conhecer um pouco de Pascal, logo descobrirá muitos elementos semelhantes entre as linguagens. Você reconhece declarações de atribuição, escolhas IF-THEN, laços FOR-NEXT e coisas semelhantes. Mas se estiver passando de BASIC para Prolog, terá de dar um passo maior. Não reconhecerá muitas das estruturas de texto. Um programa Prolog poderá parecer apenas um banco de dados de fatos – muitos parênteses em linhas não relacionadas, sem sentido. Mas não se preocupe. Com um pouco de prática você achará sua leitura tão fácil quanto BASIC. Além do mais, na medida em que avançar, descobrirá alguns comandos práticos muito semelhantes às declarações avançadas do Pascal ou outra linguagem procedural. Ficará surpreso ao descobrir tudo o que pode fazer com umas poucas linhas de programa.

OBJETOS E RELACIONAMENTOS

Prolog em seu plano mais simples cuida de objetos e o relacionamento entre eles. É, inclusive, chamado de linguagem orientada a objetos. Um objeto não será necessariamente algo tangível. Pode ser qualquer coisa que você possa representar simbolicamente no computador.

Durante alguns anos, Prolog foi mais popular na Europa do que nos Estados Unidos. Foi inventado na França e muito utilizado da Hungria à Inglaterra. A ligação européia poderia ser o motivo pelo qual membros da família real são freqüentemente utilizados como objetos nos textos Prolog. Eis alguns exemplos de objetos:

```
charles
philip
diana
elizabeth
```

Eis alguns relacionamentos úteis ao considerar estes objetos:

king	(rei)
queen	(rainha)
prince	(príncipe)
princess	(princesa)
mother	(mãe)
father	(pai)
son	(filho)

Você deve ter observado que nenhuma destas palavras está em maiúsculas. Não é nenhum engano; devem começar com letra minúscula. Começar uma palavra com letra maiúscula tem um significado especial em Prolog, que será explicado posteriormente com a discussão das variáveis. Por ora, permaneça na digitação de letras minúsculas.

Caso você não seja monarquista, eis outro conjunto de objetos e relacionamentos. Os objetos são:

cow	(vaca)
bat	(morcego)
iguana	(iguana)
redwood	(jacarandá)
fern	(samambaia)
ibm pc	(ibm pc)
apple macintosh	(apple macintosh)

e os relacionamento são:

animal	(animal)
mineral	(mineral)
vegetable	(vegetal)
mammal	(mamífero)
reptile	(réptil)
computer	(computador)

Agora você está pronto para escrever algumas sentenças Prolog que utilizam estes objetos e relacionamentos. Mas antes de iniciar esta tarefa, deverá primeiro considerar a sintaxe.

SINTAXE

A *sintaxe* de uma linguagem de programação são as regras que governam as palavras aceitáveis, as posições nas quais estas palavras devem ser escritas, onde é colocada a pontuação, e assim em diante. Mesmo que você compreenda a teoria da linguagem, não irá

a lugar nenhum se não se prender à sintaxe. Existem diversos tipos de sintaxe no Prolog. Implementações da linguagem por empresas de software diferentes, algumas vezes parecem ser muito diferentes. Mesmo assim, muitas das construções da linguagem funcionam da mesma forma, mesmo que apareçam em roupagem diferente.

Turbo Prolog contém a maioria das características e a sintaxe do Prolog descritos no livro popular de W. F. Clocksin e C. S. Mellish, *Programming in Prolog* (New York, Springer-Verlag New York Inc., 1984). É uma boa leitura caso esteja interessado em aprender mais a respeito de Prolog, depois de terminar este livro e a introdção no *Manual de Proprietário* de Turbo Prolog da Borland. Turbo Prolog segue aproximadamente o Prolog Clocksin e Mellish. O texto Clocksin e Mellish utilizou um determinado estilo do Prolog escrito, com um determinado conjunto de pontuações e soletramentos. Este estilo tornou-se assim um padrão "não-oficial" para muitos usuários de Prolog. Pelo fato do livro ser tão admirado e tão popular, o estilo está o mais próximo possível de um padrão.

FATOS

A primeira maneira de combinar um objeto e um relacionamento é utilizando-os para definir um *fato*. A sintaxe do Turbo Prolog quer que você escreva fatos da seguinte maneira:

```
relação (objeto)
```

Observe que o objeto está no interior de um parênteses, e o relacionamento o precede. Este "objeto" tem aquele "relacionamento". Ao ser escrito desta forma, o relacionamento é conhecido como *predicado* e o objeto é conhecido como *argumento*. Você poderia pegar alguns objetos e relacionamentos das duas listas anteriores e construir os seguintes fatos:

```
mammal (bat)          [mamífero (morcego)]
prince (charles)      [príncipe (charles)]
```

O segundo fato significa que o objeto "charles" tem o relacionamento "prince". A tradução em inglês é necessariamente vaga porque a lógica do fato Prolog não especifica se "charles era um príncipe", "charles é um príncipe" ou "charles viu um príncipe". Na medida em que você utiliza Prolog, tem que manter sob controle o que os relacionamentos representam – Prolog não pode fazer isto por você. O seu programa somente fará sentido se você for consistente durante um programa inteiro no que diz respeito ao significado de um dado relacionamento. Algumas vezes utilizar palavras de relacionamento que mais se aproximam daquilo que você quer dizer ajuda. Por exemplo, se você quiser indicar o fato de que charles é um príncipe, poderá utilizar este relacionamento com ele como objeto:

is_a_prince (charles)	[é_um_príncipe (charles)]

Os travessões indicam ao computador e compilador que isto é uma única palavra longa para um relacionamento. A velha abreviação usada em computação, GIGO (lixo que entra, lixo que sai), definitivamente se aplica ao Prolog. O compilador não tem meios de determinar se um fato é verdadeiro ou falso no mundo real. Se você colocar o fato

king (diana)	[rei (diana)]

no programa, Prolog vai aceitá-lo e utilizá-lo em trabalho futuro. Tudo que você informa ao Turbo Prolog é um fato; ele aceita como um fato. Você tem que ser responsável por verificar os seus fatos sempre que for necessário. Existem casos em que você pode querer utilizar declaradamente fatos falsos. Se você está tentando decidir o que pode acontecer em uma determinada situação, por exemplo, poderá querer utilizar fatos que não são necessariamente provados, demonstrados, ou mesmo próximos da verdade atual.

INTRODUZINDO O PRIMEIRO PROGRAMA-EXEMPLO

O que segue são mais alguns fatos da segunda lista de objetos e relacionamentos mencionados anteriormente:

animal (cow)	animal (vaca)
animal (bat)	animal (morcego)
animal (iguana)	animal (iguana)
vegetable (redwood)	vegetal (jacarandá)
vegetable (fern)	vegetal (samambaia)
computer (ibm_pc)	computador (ibm_pc)
computer (apple_macintosh)	computador (apple_macintosh)

Novamente, o travessão é utilizado para fazer um único objeto de várias palavras.

Estes fatos podem ser os fundamentos de um programa Prolog. Antes de introduzi-los, porém, existem mais duas coisas a saber: como fazer um comentário e como dividir um programa Turbo Prolog.

COMENTÁRIOS

O compilador Turbo Prolog transforma seu arquivo-fonte em um arquivo-objeto – um programa que pode ser processado diretamente; converte cada bit de texto em códi-

go de linguagem de máquina. Algumas vezes, você vai querer que o compilador ignore certas partes do texto. Por exemplo, você poderia desejar começar o texto de um arquivo de programa-fonte com as palavras

```
Exemplo bobo de programa Turbo Prolog.
```

Se introduzir apenas estas palavras, o compilador ficará confuso quando tentar interpretá-las. Transforme-as em um comentário colocando-as entre *marcas de comentário* como as seguintes:

```
/*Exemplo bobo de programa em Turbo Prolog*/
```

Quando uma parte de texto é precedida e sucedida pelas marcas barra-asterisco, o compilador a ignora. Isto permite que se acrescente a um programa um texto que vai explicar o que é, o que faz, e como o faz.

PARTES DO PROGRAMA

A maioria dos programas Turbo Prolog será organizada em quatro partes principais:

- Cláusulas
- Predicados
- Domínios
- Meta

Apesar de nem todas as partes precisarem estar presentes em todos os programas, você deverá familiarizar-se com todas.

CLÁUSULAS

Os fatos que você constrói a partir de seus objetivos e relacionamentos estão descritos na seção de cláusulas. As cláusulas também contêm regras e outros elementos que serão discutidos mais adiante.

PREDICADOS

Predicados são os relacionamentos. O termo "predicado" é emprestado da lógica formal, que é uma das raízes do Prolog. Sempre que o programa for utilizar um determinado predicado nas cláusulas, será preciso declará-lo formalmente na seção de predicados.

DOMÍNIOS

Turbo Prolog precisa de mais um nível de explicação antes que se diga que o programa está completo. É preciso informar os argumentos que o predicado vai utilizar. Para manter mínima a utilização da memória do computador e tornar os programas mais fáceis de ser depurados, Turbo Prolog tem uma série de verificações de *tipos* com as quais os programadores Pascal estão familiarizados. Ele precisa saber adiantado que "tipo" de coisa um argumento pode ser.

META

Esta é a seção do Turbo Prolog que informa o que você quer descobrir ou o que quer que o computador faça com as informações fornecidas nas três seções anteriores. Normalmente, um programa tem ao menos os predicados e as cláusulas, mas é possível ter um programa que seja apenas uma seção de meta. Com a seção meta, o Turbo Prolog difere dos Prolog Clocksin e Mellish. Clocksin e Mellish mantêm a meta fora do programa. Pelo fato do Prolog poder ser utilizado interativamente, é comum processar um programa e depois esperar que ele peça uma meta. Você digita a meta em um terminal e depois espera para ver o que o Prolog vai lhe entregar.

Turbo Prolog também pode funcionar desta maneira, mas fornece uma seção de meta que lhe permite processar programas sem interação. Você pode colocar a meta no código original, de modo que assim que processar o programa poderá estar operando em direção à solução desejada.

O PRIMEIRO EXEMPLO DE PROGRAMA

Dispare Turbo Prolog, vá ao Editor, e digite a primeira linha:

```
/*UTP Exemplo 1 — Animal, Vegetal, ou Computador*/
```

Depois digite RETURN. A tela pode rolar enquanto você digita a primeira linha, mas vai voltar à posição inicial quando você digitar RETURN. A Figura 4-1 mostra o que você deveria ver.
Depois digite as palavras:

```
domains
predicates
goal
clauses
```

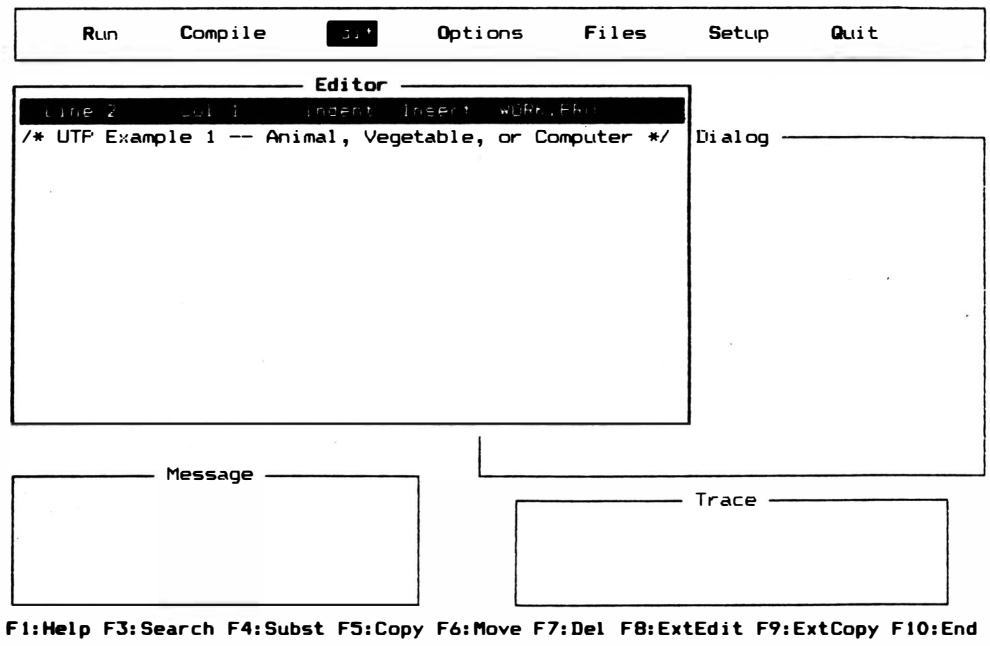


Figura 4-1 Exemplo de uma linha de comentário na janela Editor

digitando RETURN duas vezes entre cada palavra. A Figura 4-2 mostra o resultado. Você tem o esqueleto do programa Turbo Prolog.

Mova o cursor para a linha depois da palavra "clauses" e tecle TAB. Depois introduza os fatos conforme mostrado na Figura 4-3, terminando cada fato individual com um ponto e um retorno (digite RETURN). O ponto informa ao compilador que esta cláusula está terminada e que deve se preparar para outra cláusula ou para o fim do arquivo. Digitando RETURN você levará o cursor para a linha seguinte, e se Auto Indent estiver ligado, verá o cursor parar na mesma posição horizontal como no começo da linha anterior. Caso Auto Indent não pareça estar ligado, tecle CTRL-Q–I para conseguir fazê-lo.

Agora que tem algumas cláusulas, precisará informar ao Turbo Prolog quais os predicados (relacionamentos) que está utilizando nelas. Mova o cursor para a linha logod/ depois da palavra "predicates" e tecle TAB. Depois digite

animal (thing)	[animal (coisa)]
vegetable (thing)	[vegetal (coisa)]
computer (thing)	[computador (coisa)]

em três linhas, conforme indicado aqui. As declarações de predicado não precisam terminar com um ponto como o fazem as cláusulas. A sua tela Turbo Prolog deve agora estar parecida com a da Figura 4-4.

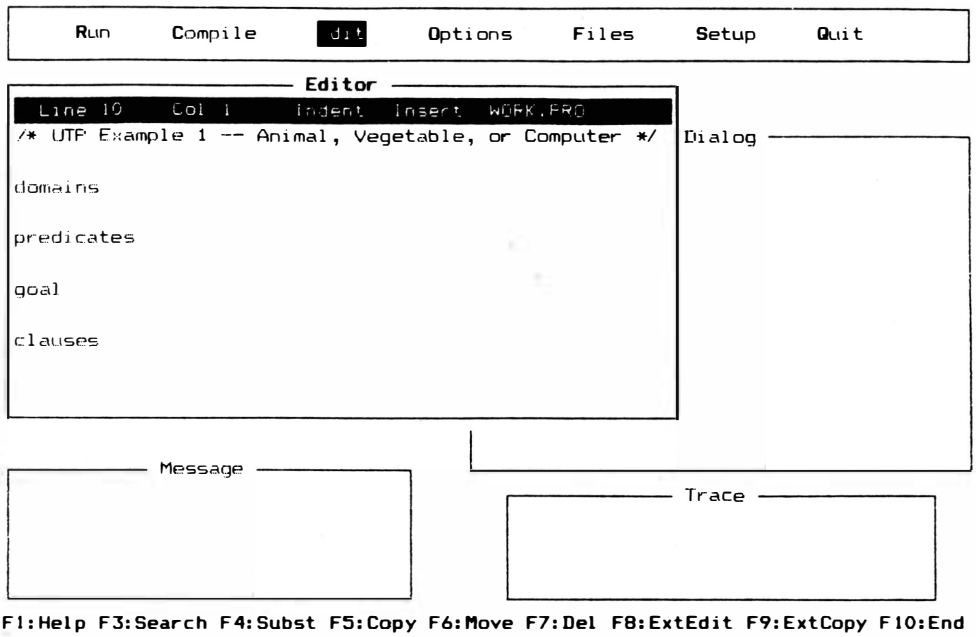


Figura 4-2 Esqueleto do programa Turbo Prolog

Suas declarações de predicado informaram ao Turbo Prolog que os predicados animal, vegetal e computador dizem respeito a "coisas". Não sem surpresas o compilador agora quer saber que tipo de coisas. É para isto que serve a seção domínio.

Mova o cursor para a linha logo depois da palavra "domains", tecle TAB, e digite a linha

```
thing = symbol
```

com um espaço depois de "thing" e outro espaço depois do sinal de igual. Agora terminou de digitar.

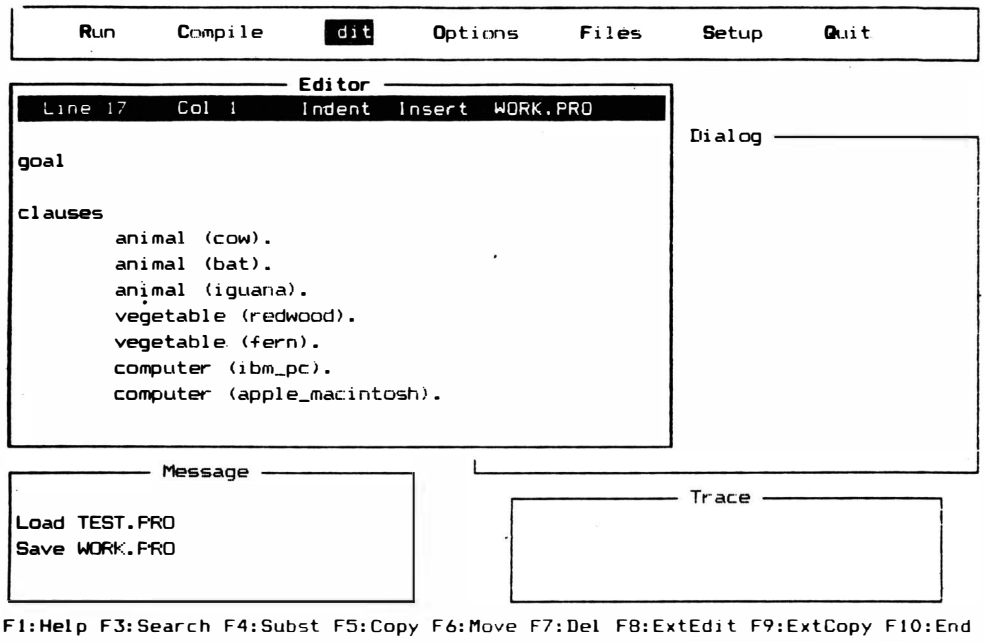


Figura 4-3 Fatos colocados em todas as cláusulas com pontos no final

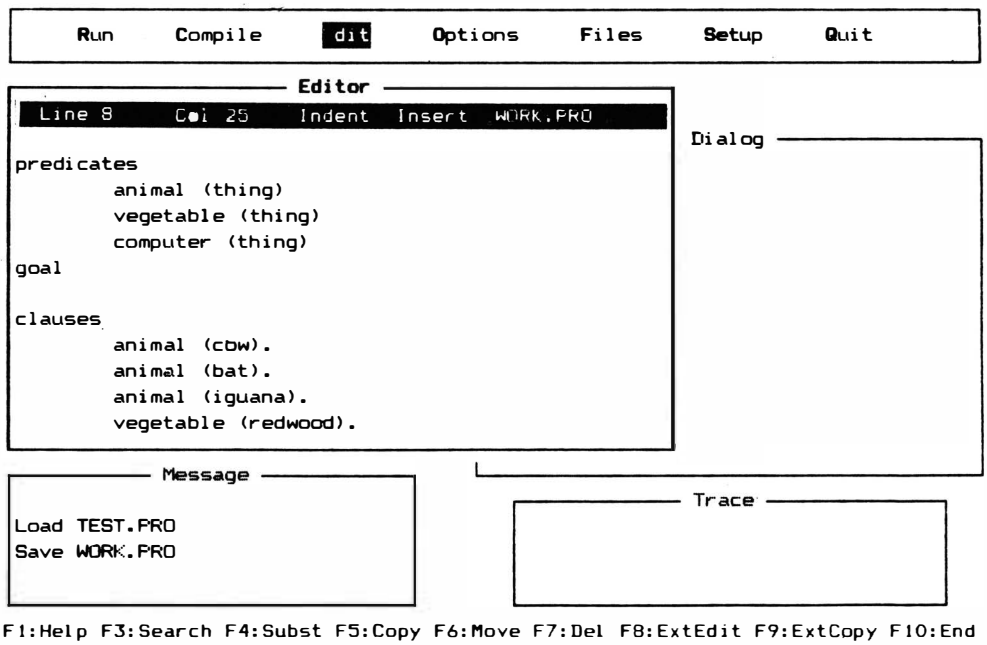


Figura 4-4 Declarações de predicado (sem pontos)

GRAVE O QUE TEM ATÉ AGORA

Antes de seguir, proteja-se contra falhas de rede ou do ataque de uma criança que adora teclados, gravando o arquivo-fonte em que está trabalhando. Tecle ESC para ir até o menu principal e depois F para obter o submenu Files. Escolha S de "Save", e digite o nome do arquivo. Utilize "utpexl" por "Usando Turbo Prolog Examplo 1". Isto grava o arquivo no acionador A (caso este seja seu acionador pré-selecionado). Você verá uma mensagem como

```
Save utpexl.PRO
```

na janela Message, confirmando a ação.

COMPILANDO O PRIMEIRO PROGRAMA-EXEMPLO

Agora você está pronto para compilar o programa. Tecle ESC o número suficiente de vezes para voltar ao menu principal. (Caso você tenha acabado de guardar o arquivo, já estará no menu principal e ESC não fará nada.) Tecle C de Compilação. Verá uma informação de erro na parte inferior da janela Editor, conforme indicado na Figura 4-5.

VERIFICAÇÃO DE SINTAXE

Pelo fato da seção meta nada ter nela, o compilador pensou que fosse outra declaração de predicado na seção de pedicado. O compilador também posicionou o cursor no problema encontrado: a palavra "goal". A verificação interna de sintaxe é de grande auxílio durante a programação e depuração.

Cancele a linha "goal" e digite F10. Isto vai automaticamente recompilar o arquivo-fonte corrigido. Em alguns programas você poderá assim encontrar outro erro e voltar novamente à edição, mas neste caso você deveria ver a mensagem

```
Compiling utpexl.PRO
```

na janela de mensagem sem outras mensagens de erro. Isto significa que tem um programa compilado. A utilização de F10 depois de um ero de sintaxe em edição permite que você combine as funções ESC e Compile (teclando C) com uma única tecla.

PROCESSANDO O PRIMEIRO PROGRAMA

Agora, o programa está compilado na memória, esperando para fazer alguma

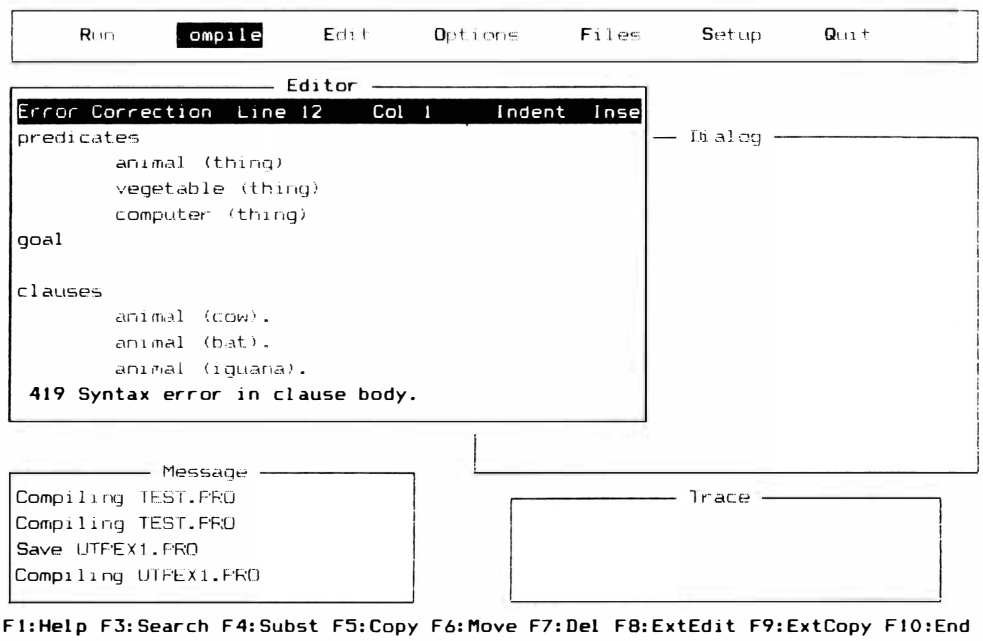


Figura 4-5 Mensagem de erro para um problema encontrado durante a compilação

coisa. Tecle R para processar o programa do menu principal. Na janela surgirá Dialog, conforme mostrado na Figura 4-6. Uma meta será solicitada. Agora o programa está em processamento. Lembre-se de que pelo fato do Prolog ser uma linguagem lógica e declarativa, basta inserir uma meta lógica, não uma descrição longa de como chegar lá.

METAS E PERGUNTAS

Apesar das metas estabelecidas para seus programas Prolog poderem eventualmente ser tão complexas quanto a decisão de um tratamento médico ou mostrar uma simulação gráfica de um vôo, as metas deste primeiro exemplo terão de ser menos exaltadas. Não existe a informação suficiente na seção cláusulas para que Prolog possa tirar muito do programa.

Programas posteriores deste livro terão metas internas. Este trata apenas de metas externas – metas que você coloca na janela Dialog. Esta janela agora está ativa e permitirá que você digite informação utilizando muitos dos mesmos comandos de edição que a janela Editor utiliza.

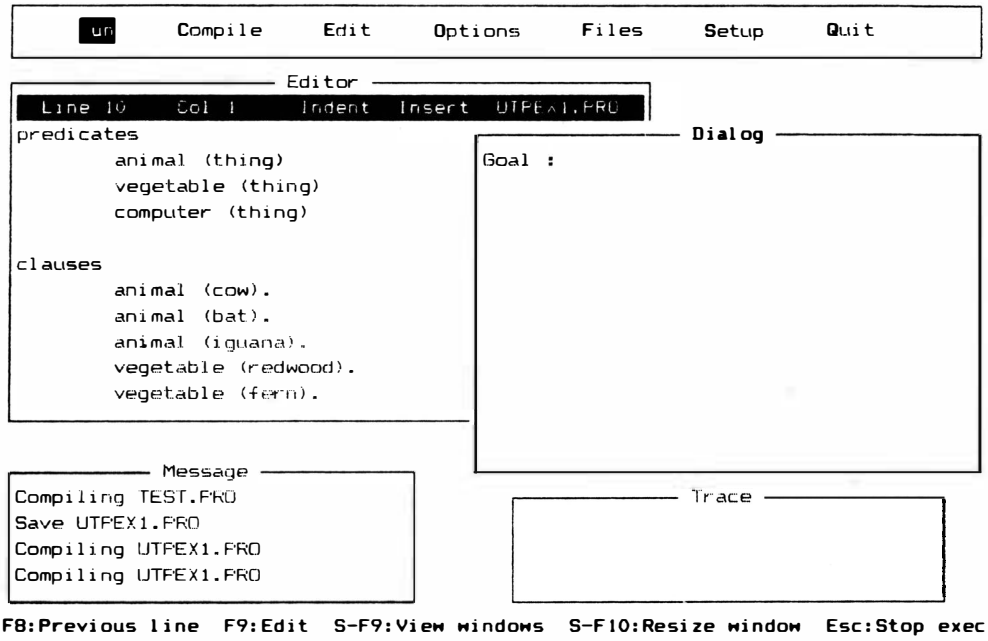


Figura 4-6 Pedido de meta na janela Dialog

VERIFICANDO FATOS

A meta menos complexa apenas verifica um fato. Caso você coloque um fato na janela Dialog, Turbo Prolog vai encará-lo como uma pergunta e lhe informará se aquele fato está em sua lista – na seção cláusulas. Para este programa-exemplo, digite

```
animal (bat)                    [animal (morcego)]
```

e RETURN. Você, perguntou, "Um morcego é um animal?", ou mais precisamente na lógica, "Será que o objeto morcego tem o relacionamento animal?" Imediatamente verá a resposta – a palavra "True" (verdadeiro) aparece, como mostrado na Figura 4-7.O programa continuará sendo processado e depois pedirá outra meta. Tente

```
computer (ibm_pc)               [computador (ibm_pc)]
```

e verá a resposta: "True".

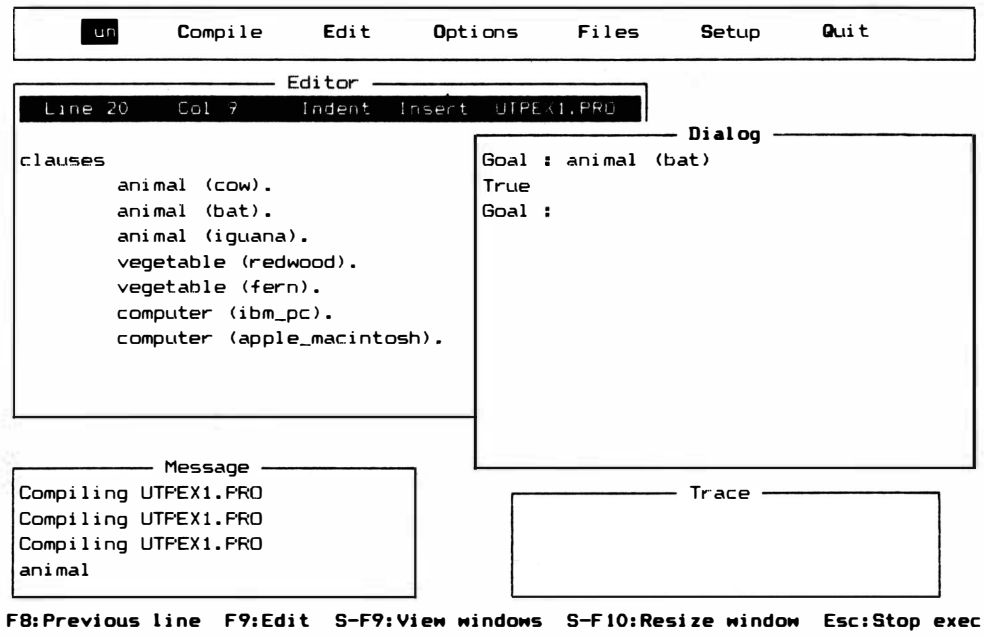


Figura 4-7 Resposta Turbo Prolog à meta externa simples

PARANDO O PROGRAMA

Caso queira parar o programa e sair da janela Dialog, bastará teclar ESC. Isto fará com que volte ao menu principal. Você vai trabalhar mais com o programa, de modo que se o parou agora, dispare-o novamente com um R a partir do menu principal. Em seguida, tente a seguinte meta:

vegetable (ibm_pc) [vegetal (ibm_pc)]

Você verá Prolog responder "False".

Nem "True" nem "False" indicam que alguma coisa seja exatamente verdadeiro ou falso no mundo externo. Indicam apenas que Turbo Prolog foi capaz de procurar em suas cláusulas e obter informações que produziram estes resultados. Caso a informação do Turbo Prolog esteja errada, suas respostas estarão erradas.

BUSCA DE INFORMAÇÃO

Prolog procura nas cláusulas de cima pra baixo e da esquerda para a direita. Nos

casos de "animal (bat)" e "computer (ibm_pc)", encontrou fatos que combinam exatamente com a meta. No caso da meta "vegetable (ibm_pc)" procurou em todo banco de dados sem encontrar nenhuma concordância exata. Portanto, no que diz respeito a este programa, não há meios de saber se um ibm_pc é um vegetal. Poderia ser, logicamente. Apenas pelo fato de um objeto ter um relacionamento – como em computer (ibm_pc) – não se restringe apenas àquele relacionamento.

VARIÁVEIS

Existe outra maneira de fazer perguntas. Caso conheça o relacionamento mas não o objeto, poderá utilizar *variáveis*. Qualquer palavra-objeto que inicie com letra maiúscula é considerada, uma variável. Depois da maiúscula, poderá ter qualquer número de letras (maiúsculas ou minúsculas), como algarismos e travessões. A maneira mais rápida de ver o que uma variável pode fazer é interrogando o programa. Tente

```
animal (X)
```

como uma meta. Brevemente verá o que está mostrando na Figura 4-8.

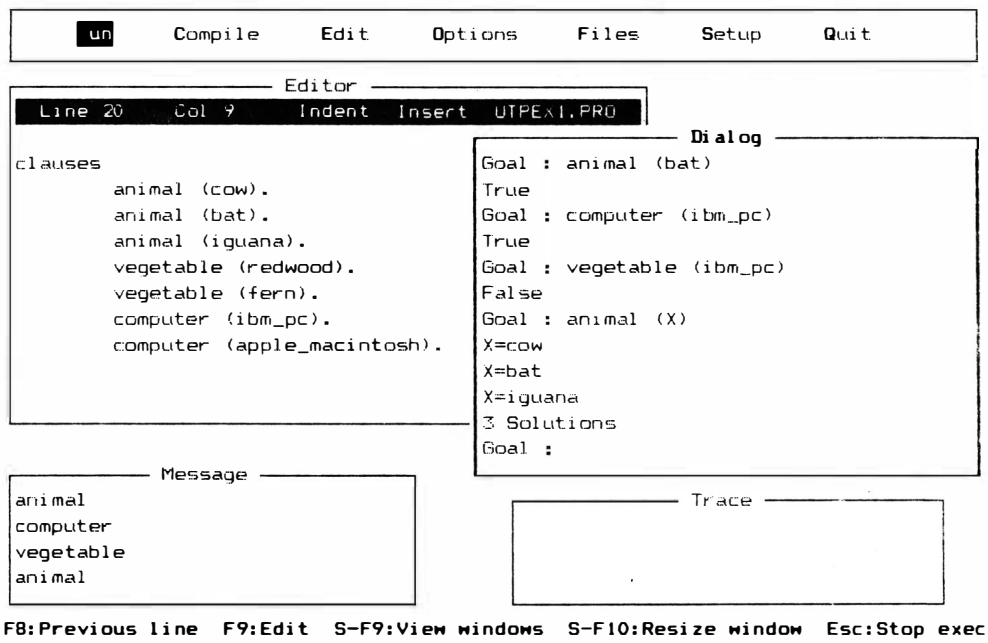


Figura 4-8 Resposta do Turbo Prolog à pergunta de variável a respeito de animal

Turbo Prolog toma a meta com a variável e a compara às cláusulas. Primeiro verifica se o predicado é o mesmo da primeira cláusula. Se é, então testa para ver se a meta e o predicado têm o mesmo número de argumentos. Se uma dessas coisas não for verdadeira, Turbo Prolog desistirá na primeira cláusula e prosseguirá para testar a meta contra a segunda cláusula. Se ambos forem verdadeiros – mesmo predicado e mesmo número de argumentos – Prolog verificará se os argumentos são os mesmos.

Neste caso, o argumento meta é uma variável, e pode significar qualquer coisa, de modo que compilador "instancia-a" ao valor na cláusula; em outras palavras, o compilador assume, por um momento, que tem o valor do argumento na cláusula. Na linguagem do Turbo Prolog, isto é chamado *ligar* um valor a uma variável. Agora, a meta e a cláusula estão completamente de acordo, e Turbo Prolog pode informar-lhe, na janela Dialog, que encontrou um caso do fato que você pediu. Informa-lhe o fato, juntamente com o valor que temporariamente estabeleceu para a variável, e depois volta às cláusulas.

Pelo fato de ter utilizado uma variável, Prolog vai procurar todas as cláusulas que possam coincidir com a meta ou tornarem a meta verdadeira. Ele vai liberar o "X" do

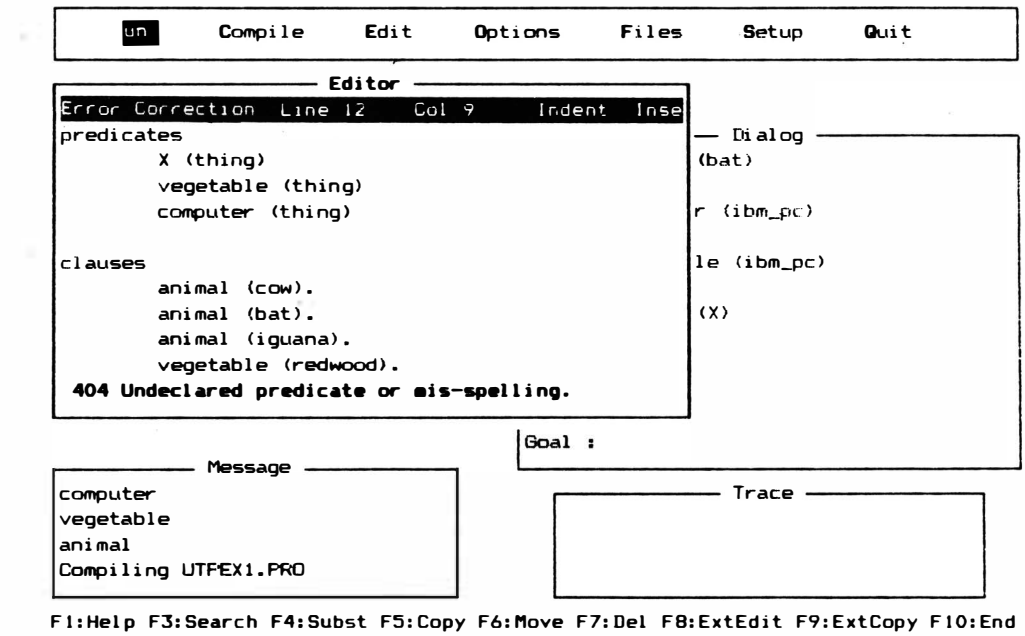


Figura 4-9 Erro provocado pela utilização de uma variável como predicado

primeiro valor e continuará para o próximo abaixo (ou mais à esquerda). Vai inspecionar esta cláusula da mesma forma que fez com a primeira. Prolog vai acabar mostrando todos os casos que permitam que a meta seja verdadeira. O exemplo atual apresentará três instâncias em que a meta é verdadeira. Lembre-se de que se tentar utilizar uma variável como predicado (como um predicado "X(coisa)"), obterá um erro, conforme mostrado na janela Editor da Figura 4-9. Turbo Prolog espera que predicados sejam utilizados e declarados no programa.

CAPÍTULO 5

REGRAS E RETROCESSO ("BACKTRACKING")

Turbo Prolog tem muito mais para oferecer do que apenas os fatos elementares e variáveis descritos no Capítulo 4. Este capítulo vai apresentar fatos com argumentos múltiplos, metas compostas, variáveis vazias, regras, retrocesso e acompanhamento. Ao terminar este capítulo, você verá que estes conceitos são mais simples do que pensou.

Caso você já tenha determinados conhecimentos a respeito de Prolog e conheça regras, retrocesso e metas compostas poderá apenas folhear grande parte deste capítulo para ver como Turbo Prolog os implementa. Mas é aconselhável dar uma olhada na seção de acompanhamento para saber como utilizar a janela de acompanhamento inerente ao Turbo Prolog.

TERMOS

Termos são os blocos básicos dos programas Prolog. Um termo pode ser uma constante, uma variável ou uma estrutura. No capítulo anterior você trabalhou principalmente com constantes, como

```
bat
cow
ibm_pc
apple_macintosh
```

Você também usou um pouco as variáveis, dependendo apenas do sempre popular "X". Uma constante representa um objeto determinado ou um relacionamento. Uma variável (que sempre começa com letra maiúscula) representa um objeto cujo nome você não conhece quando o programa está sendo definido. Posteriormente vamos estudar estruturas.

ARIDADE: FATOS COM ARGUMENTOS MÚLTIPLOS

A *aridade* de um predicado é o número de argumentos que aquele predicado tem. O Capítulo 4 empregou apenas predicados com um único argumento, ou aridade de um. Mas você verá que a maioria dos programas Prolog depende muito de predicados com aridades muito maiores. Por exemplo, você poderá querer utilizar o predicado

```
shark (leg, human)                    [tubarão (perna, homem)]
```

para descrever o relacionamento (predicado) entre os dois objetos (argumentos) "leg" e "human". Isto poderia ser apresentado como "Sharks eat the legs off of humans" ["Tubarões comem pernas de homens"]. Mas lembre-se de que o significado é aquele que você determina. Esse mesmo predicado poderia significar "Legs are the parts of humans that attract sharks" ["Pernas são a parte do ser humano que atraem tubarões"]. Você pode escrever comentários em seu programa para registrar o que quer que o predicado signifique, mas o comentário ajuda apenas o próximo leitor do programa a compreender seu significado. Não ajuda o compilador a compreender a sua intenção.

Você pretenderá utilizar um predicado com uma aridade grande para descrever seu sistema de compilação. Por exemplo.

```
computer (ibm_pc, _512K, _2_floppy, _DOS_2, _2serial, _1parallel, _Hercules, _Princeton)
```

descreveria as características básicas de seu sistema. Você poderia mesmo ter um predicado sem argumentos, mas isto é um caso que será visto mais tarde. (Observe que, no exemplo acima, a maioria das entradas estão com letras minúsculas. Letras maiúsculas serão tratadas como variáveis, a não ser que sejam precedidas por um travessão. Números devem ser evitados ou precedidos, também, por um travessão.

METAS COM ARGUMENTOS MÚLTIPLOS

Depois de fazer um pequeno programa com predicados com um único argumento no Capítulo 4, você interrogou aquele programa com metas de um só argumento. Programas que utilizam predicados com aridades maiores também poderão trabalhar com metas mais complexas. O que segue são algumas cláusulas para um exemplo simples:

```
watches (bill, bob)          [Observa (bill, bob)]
watches (john, jane)         [Observa (john, jane)]
watches (fred, felicia)      [Observa (fred, felicia)]
watches (mitch, bill)        [Observa (mitch, bill)]
```

watches (brenda, greg)	[Observa (brenda, greg)]
watches (bob, bob)	[Observa (bob, bob)]
watches (fred, greg)	[Observa (fred, greg)]
watches (bill, phillippe)	[Observa (bill, phillippe)]

Vá até a janela Editor e digite as linhas que está vendo na Figura 5-1. São a linha de comentário título, as principais partes e cláusulas maiores deste programa. Lembre-se de colocar pontos no final dos fatos na seção de cláusulas. Fatos não precisam ter pontos quando aparecem sozinhos, mas a seção de cláusulas precisa tê-los para separar um fato de outro. A seção de metas foi deixado de fora porque você vai utilizar metas externas com este programa.

O predicado "watches" significa que o objeto na primeira posição observa o objeto na segunda posição. Em outras palavras,

watches (bill, bob)

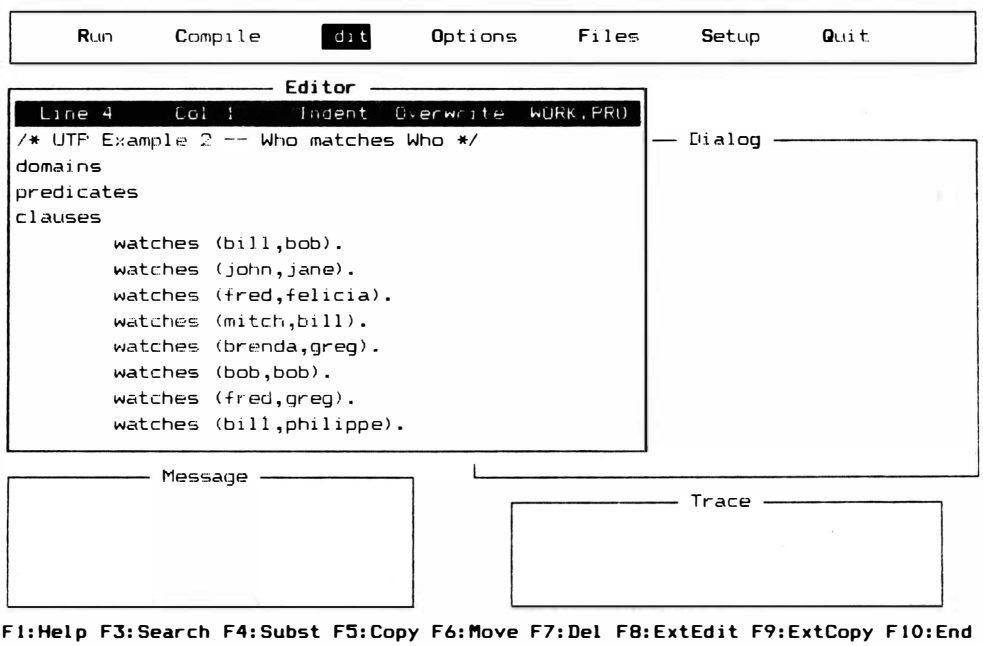


Figura 5-1 Partes iniciais e cláusulas em programas-exemplo

significa "Bill watches Bob" ["Bill observa Bob"], e não "Bob watches Bill" ["Bob observa Bill"].

Agora você pode acrescentar alguns predicados rápidos e declarações de domínio, como mostrado na Figura 5-2. Eles seguem o mesmo padrão utilizado no Capítulo 4. Todos os argumentos são do mesmo tipo e podem estar no mesmo domínio ("symbol" ["símbolo"] é o melhor domínio para nomes de coisas ou pessoas). Lembre-se de que "person" é apenas nosso termo para eles. A palavra "symbol" sempre significa alguma coisa quando utilizada na seção de domínio do Turbo Prolog.

Você deveria cancelar a linha "goal" também. Poderá causar problemas quando nenhuma meta for listada.

Agora grave seu trabalho como "utpex2" (utilize ESC para sair do Editor, F para obter o menu Files e S de "Save", gravar; digite o nome do acionador de disco e arquivo, tecle RETURN e pronto). Na janela Message você poderá ver que o arquivo está realmente gravado como "utpex2.PRO". Não é preciso acrescentar a extensão .PRO –

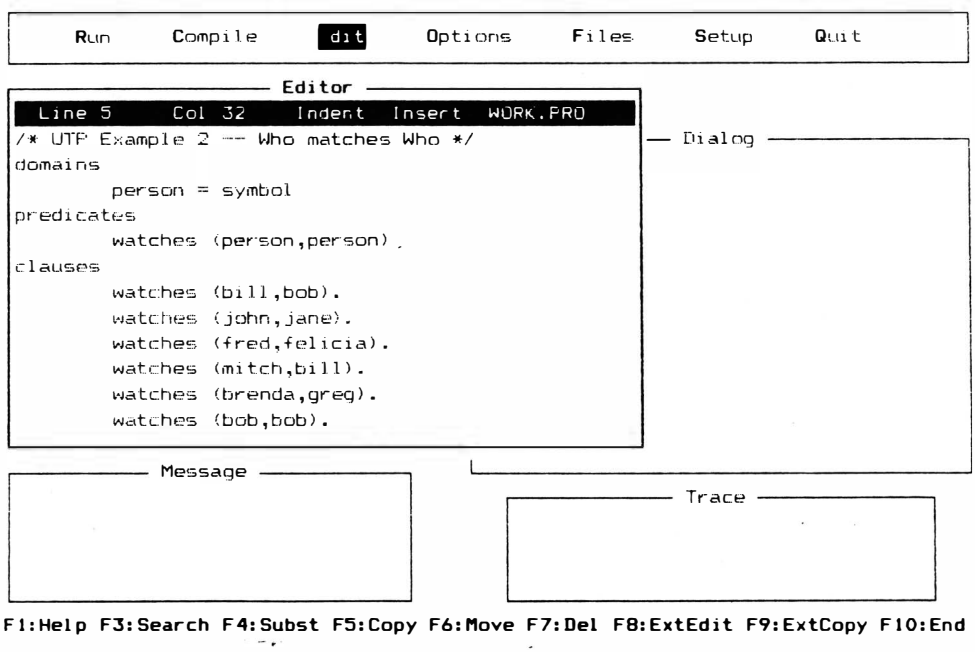


Figura 5-2 Declarações de predicado e domínio

Turbo Prolog faz isto para você se não for fornecida uma extensão para o nome do arquivo.

Agora compile e processe o programa, teclando C e depois R. Você vai ver a palavra "Goal" aparecer na janela Dialog. Agora você está pronto para experimentar metas mais complexas.

Vamos começar com algo simples. Digite

watches (fred, greg)

e RETURN. Turbo Prolog vai procurar de cima a baixo pela lista de fatos para encontrar um que coincida com a meta que você apresentou. Primeiro tenta combinar o predicado "watches". O primeiro fato utiliza aquele predicado. Depois, Turbo verifica se o fato tem o mesmo número de argumentos que a meta (dois). Todos o fazem, de modo que esta concordância ainda é uma possibilidade. Depois tenta combinar o primeiro argumento do predicado com o primeiro argumento da meta, mas não encontra nenhuma concordância.

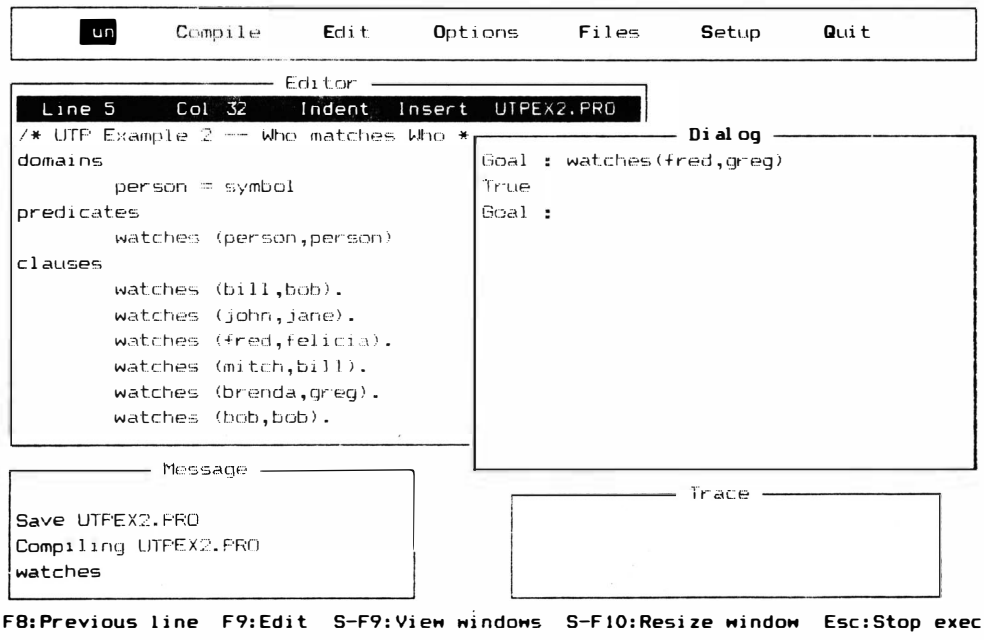


Figura 5-3 Metas com argumentos múltiplos

Neste ponto, Turbo prossegue para o próximo fato e tenta combinar com ele, passando por todo processo descrito anteriormente. Eventualmente, encontrará exatamente o mesmo fato, e responderá conforme indicado na Figura 5-3.

Agora tente utilizar uma variável na meta de argumento múltiplo. Tente a meta

watches (fred, Who)

para ver o que acontece (veja a Figura 5-4). Você pode tentar acionar a tecla F8 para digitar esta meta em particular. A tecla F8 repete à linha de meta anterior, deixando a você apenas o trabalho de mudar "greg" para "Who". "Who" é uma variável porque começa com letra maiúscula.

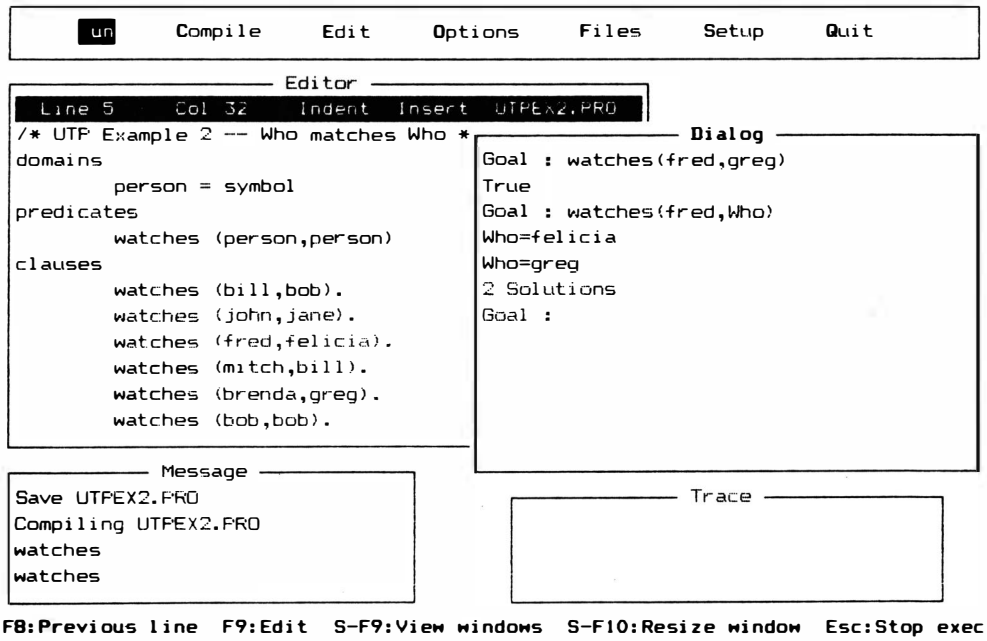


Figura 5-4 Meta com argumentos múltiplos e uma variável: parte 1

Ocorre que Fred observa duas pessoas diferentes, de modo que você obtém duas soluções em sua resposta. Você poderia ter utilizado uma variável na posição da outra "pessoa", conforme indicado na Figura 5-5. Nenhuma regra impede que um objeto apareça nas duas posições. Você deveria ter definido o predicado de modo que aquilo não fizesse sentido, mas continuaria a ser sintaticamente legal em Turbo Prolog.

Agora tente utilizar duas variáveis em uma única meta. A meta

watches (Who, Who2)

vai produzir a tela apresentada na Figura 5-6. Na verdade, esta é uma lista de todos os fatos no banco de dados da cláusula. Se o programa também incluísse predicados diferentes de "watches", isto poderia ser uma operação útil para ver quantos fatos "watches" estariam no programa.

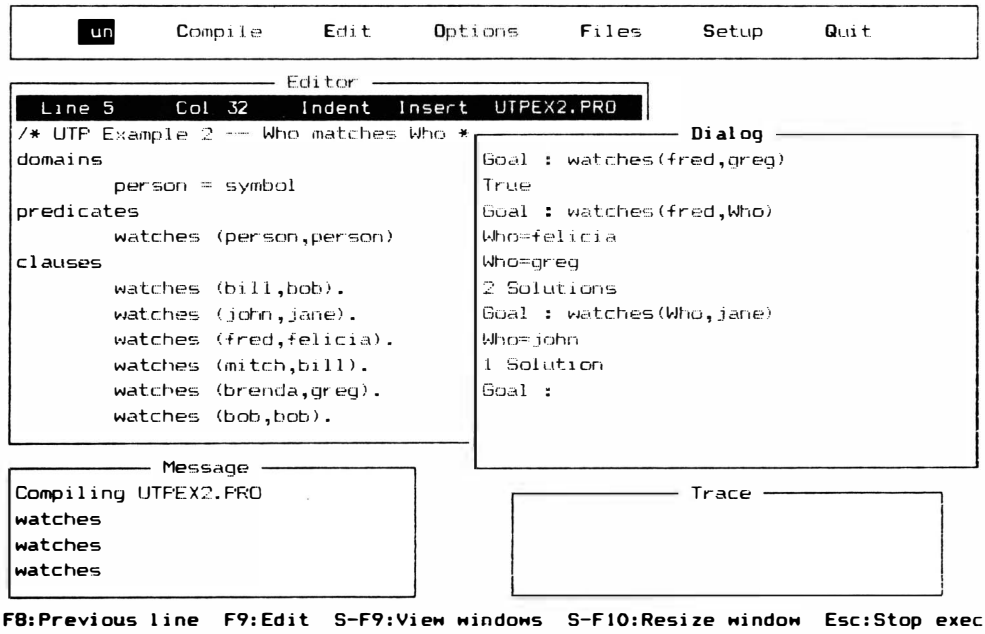


Figura 5-5 Meta com argumentos múltiplos e uma variável: parte 2

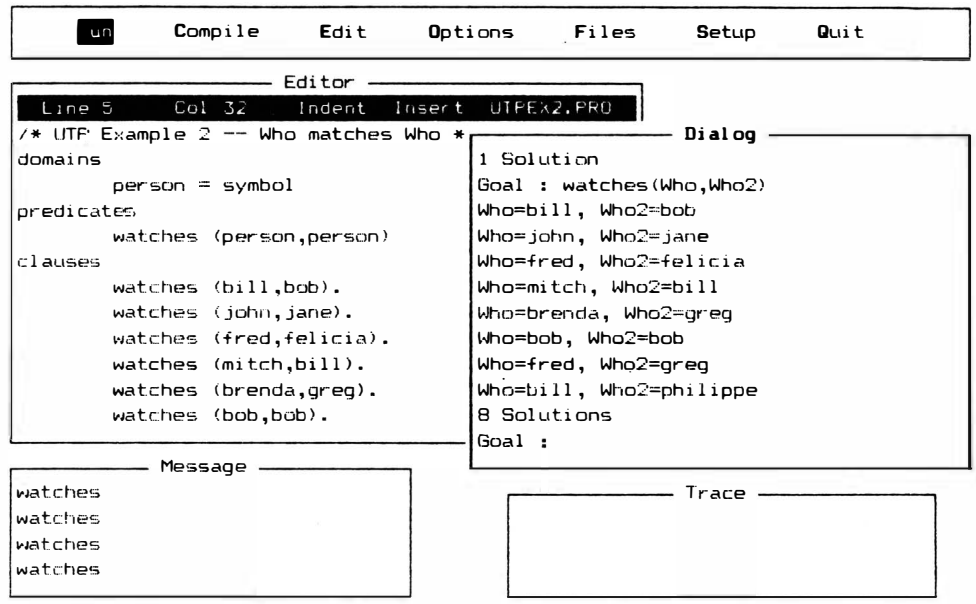


Figura 5-6 Meta com duas variáveis.

VARIÁVEL ANÔNIMA OU VAZIA

Outro tipo de variável conveniente em algumas situações é a variável anônima. Outros Prologs algumas vezes chamam isto de variável "vazia". É escrito apenas como um travessão. A variável anônima é utilizada nos mesmos locais em que são utilizadas as variáveis-padrão, mas nunca é inicializada com um valor determinado.

Aqui está um exemplo de uma meta que utiliza a variável anônima. Vamos supor que você tenha o programa utpex2, e queira saber se Brenda observa alguém. Você poderá colocar a meta

```
watches (brenda,_)
```

e ver que é "Verdade": Brenda observa alguém (veja Figura 5-7). Mas, pelo fato de ter utilizado a variável vazia, não já informação de quem ela observa. Turbo Prolog apenas procura um fato que tem o predicado "watches", dois argumentos ("brenda" como primeiro argumento e qualquer coisa como segundo argumento). Ele não precisa saber mais,

de modo que não se preocupou em perguntar e não lhe forneceu nenhum nome.

Caso tente a meta

```
watches(_,_)
```

saberá que alguém observa alguém (veja Figura 5-8); se tentar

```
watches(fred,_)
```

verá a resposta "True", significando que "fred" observa alguém ou que há alguém sendo observado por "fred".

A variável anônima também pode ser utilizada em fatos. Caso você acrescente a linha

```
watches(_, brenda)
```

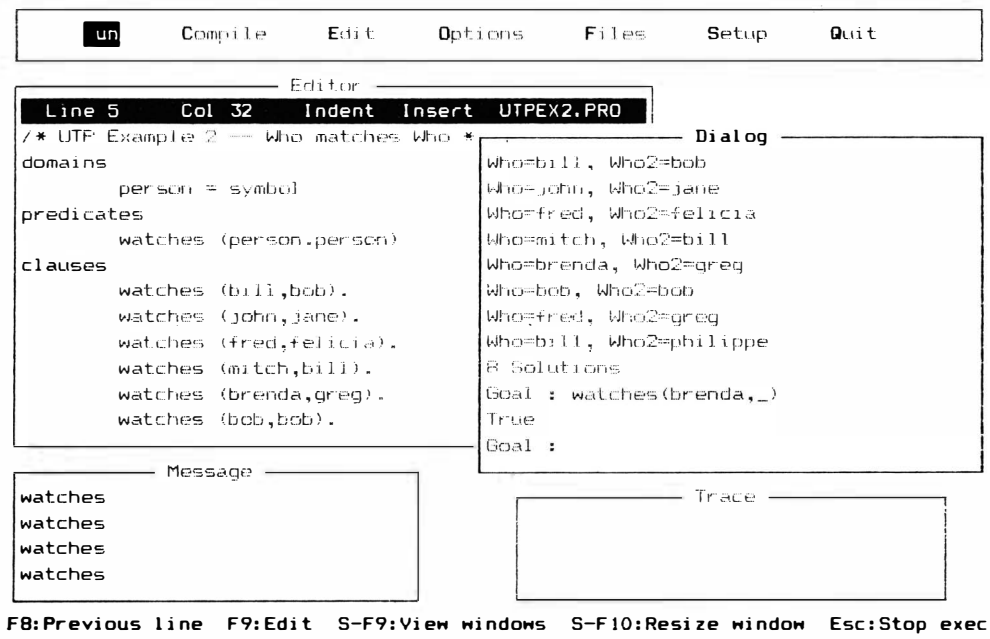


Figura 5-7 Meta com variável anônima

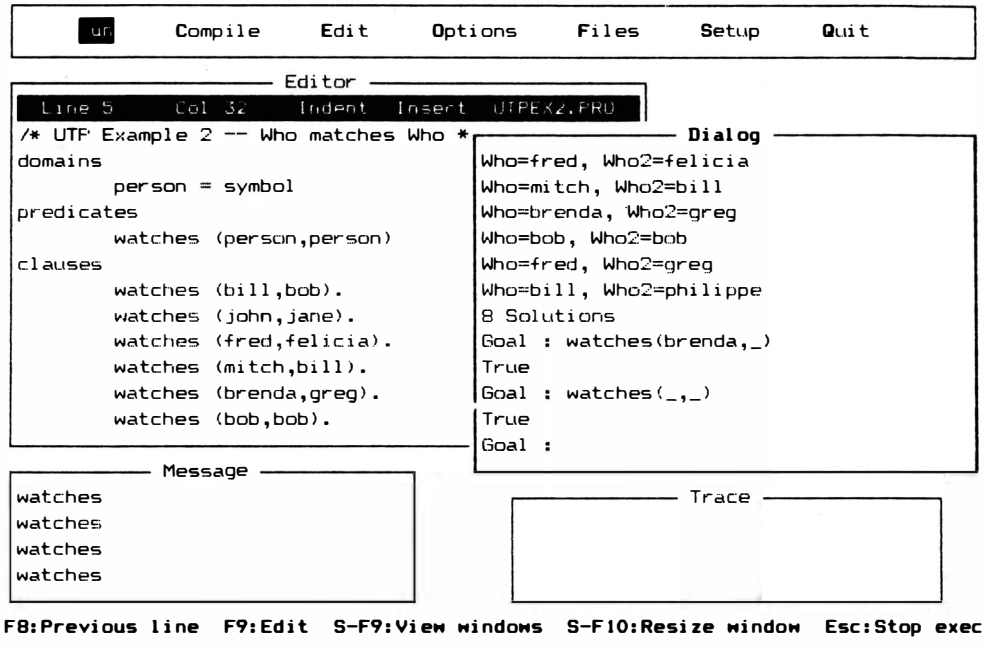


Figura 5-8 Meta com duas variáveis anônimas

à seção cláusula, estará dizendo que "Everyone watches brenda" ["Todo mundo observa brenda"], ou que "We don't care who is watching brenda" ["Não importa quem observa brenda"]. Não acrescente isto ao programa, uma vez que vai apenas confundir o que vem depois.

METAS COMPOSTAS

As metas podem ser ainda mais complexas. Por exemplo, as metas também podem lidar com fatos múltiplos.

O OPERADOR AND

Caso você proponha a meta

watches (bill, bob) and watches (john, jane)

estará perguntando "É verdade pelas cláusulas que Bill observa Bob AND(e) John obser-

va Jane?" Esta meta composta será considerada verdadeira apenas se as duas metas individuais forem verdadeiras. Faça uma tentativa (Figura 5-9).

Você não precisa teclar RETURN para digitar uma meta com este tamanho. As palavras na janela Dialog automaticamente continuam na linha seguinte.

O Turbo Prolog tenta primeiro aplicar a primeira submeta. Assim que isto estiver feito, guarde o fato de que a primeira submeta é satisfeita e tente aplicar a segunda submeta.

Agora tente

watches (bill, bob) and watches (greg, brenda)

O resultado será falso, porque as duas metas individuais precisam ser verdadeiras para que a meta composta seja verdadeira, e

watches (greg, brenda)

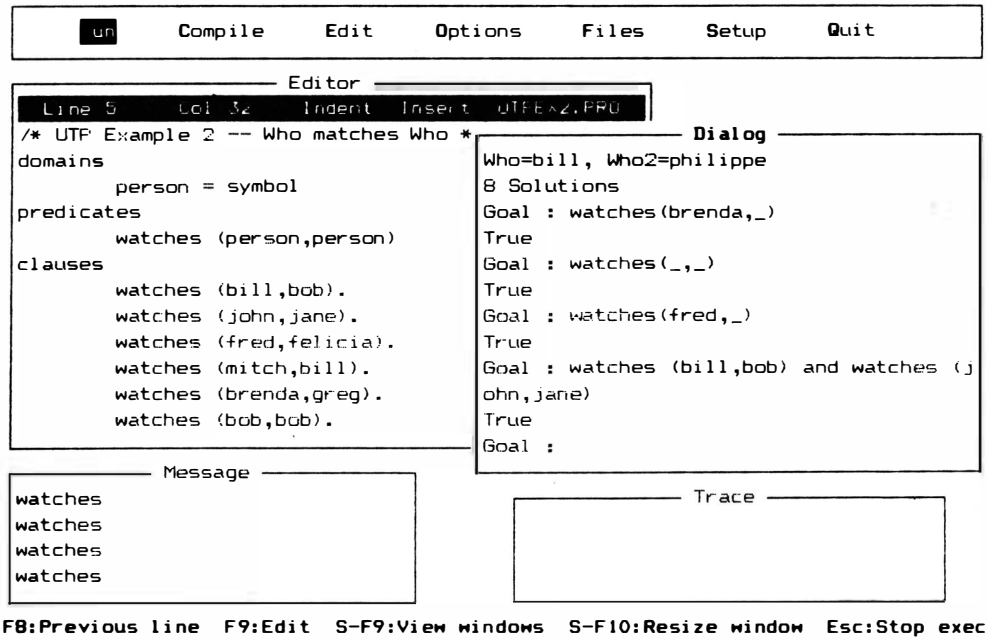


Figura 5-9 Meta composta

é uma meta falsa. Não se aplica aos fatos na seção cláusulas. O fato

```
watches (brenda, greg)
```

não é o mesmo, porque os valores estão em posições diferentes. Pelo fato de muitas outras implementações Prolog utilizarem uma vírgula para significar AND, Turbo Prolog também oferece esta opção, de modo que a meta composta acima poderá ser escrita como

```
watches (bill, bob), watches (greg, brenda)
```

OR E OUTROS OPERADORES

Você pode construir também metas compostas, denominadas *conjunções* com outras palavras de ligação. A meta composta feita com duas metas ligadas pela palavra "or" será verdadeira se uma das metas individuais for verdadeira. (Da mesma forma que uma vírgula pode substituir AND, um ponto-e-vírgula pode tomar o lugar de OR.) A outra meta individual pode ser verdadeira ou falsa, que não vai afetar a verdade da meta composta. A utilização dos operadores AND e OR é fundamental para muitas áreas da ciência de computação.

Existem também diversos outros operadores que você pode colocar em metas compostas, incluindo os sinais maior do que e menor do que. Você não está limitado a utilizá-los com números. Um "símbolo" pode ser comparado com outro "símbolo", letra por letra, com as letras colocadas em ordem alfabética. Para ver como isto funciona, tente a seguinte meta, conforme ilustrado na Figura 5-10:

```
watches (bill, Who) and Who > bob
```

Você obterá a resposta

```
philippe
```

porque existe alguém que "bill" observa cujo nome vem alfabeticamente depois de "bob".

O operador NOT é discutido posteriormente neste capítulo.

REGRAS

Fatos não são os únicos habitantes da seção cláusulas. Você pode colocar regras

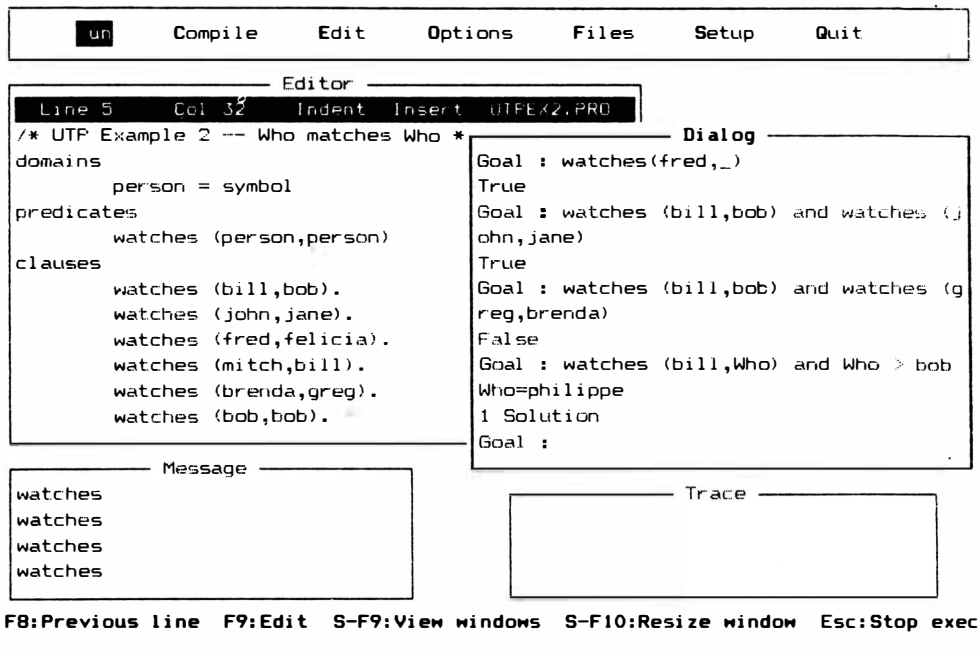


Figura 5-10 Metas compostas utilizando o operador >

em seu banco de dados. Uma regra típica diz que alguma coisa é verdadeira (uma meta será bem-sucedida) se algumas outras coisas são verdadeiras. Regras leva Prolog além do estado de um mero dicionário de pesquisa ou banco de dados até chegar a uma máquina lógica, pensante.

Vamos acrescentar algumas regras ao programa utpex2. Digite a linha

```
happy (greg) if watches (brenda, greg)          [feliz (greg) se observa (brenda, greg)]
```

e a linha

```
nervous (Who) if watches (bill, Who)          [nervoso (Who) se observa (bill, who)]
```

ao final dos fatos previamente listados (veja a Figura 5-11). Assegure-se de colocar pontos no fim das regras.

Agora é preciso acrescentar algumas declarações de predicados novos, antes de

tentar compilar o programa expandido. Tente as declarações mostradas na Figura 5-12. Você não precisa de novos domínios, porque nenhum argumento novo foi utilizado. Todos os predicados se referem a "person".

Compile e processe o novo programa. Agora, para tentar uma série de metas, digite

```
happy (greg)
```

primeiro. Você vai obter a resposta "True".

Turbo Prolog vai tentar satisfazer (ou aplicar) a meta "happy (greg)". De cima para baixo, ele vai procurar pelas cláusulas, procurando um predicado que combine com o predicado "happy". Este predicado é encontrado perto do fim da lista. Depois, você verifica se aquele predicado tem a mesma aridade – um. Se tiver, o processo continua. Turbo Prolog encontra "if" e entende que isto é uma regra, e não um fato.

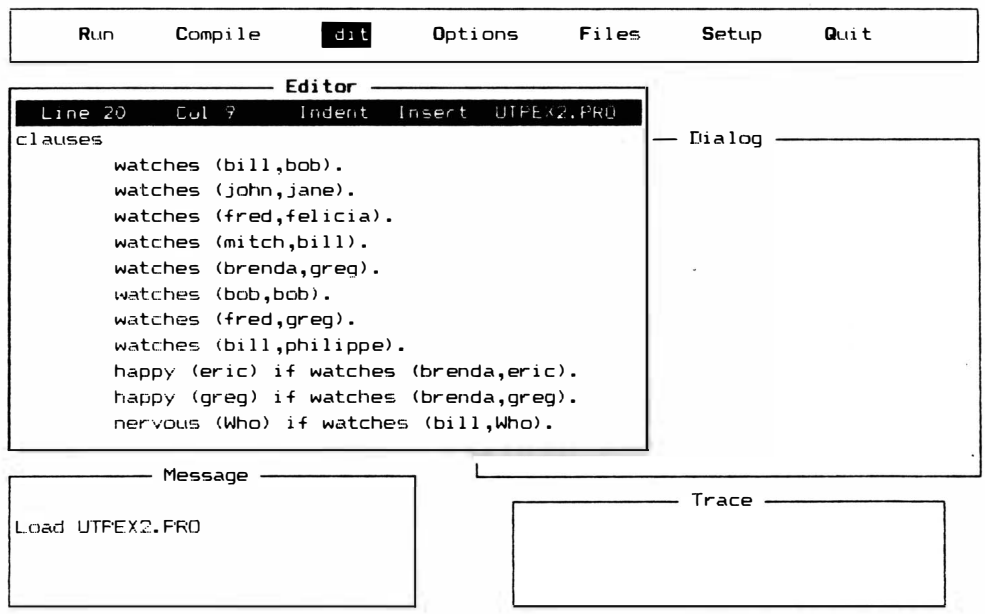


Figura 5-11 Acrescentando regras a um programa-exemplo

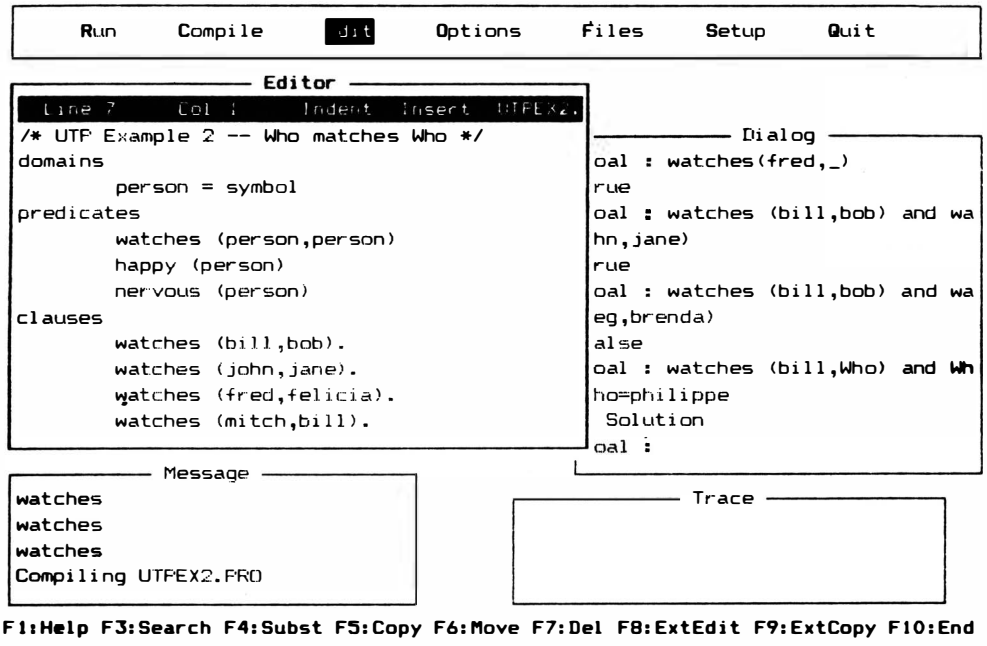


Figura 5-12 Declarações de predicados para regras

Para tornar o lado esquerdo da regra "True" de modo que possa ser utilizado para combinar com uma meta, o lado direito da regra tem de ser verdadeiro. Turbo Prolog marca seu lugar e adota "watches (brenda, greg)" para satisfazer a nova meta. Agora poderá voltar para atender a meta original.

RETROCESSO ("BACKTRACKING")

Neste ponto, Turbo Prolog volta à primeira cláusula e começa a pesquisar a lista. A isto denominados *retrocesso* e é uma característica importante do Prolog. Sempre que uma submeta tem de ser satisfeita, Prolog retrocede pelo banco de dados, sempre procurando de cima para baixo e da esquerda para a direita, para encontrar uma concordância. O retrocesso poderá, facilmente, ficar mais complexo quando as regras e as metas forem mais complexas.

Procurando pelos fatos, Turbo Prolog rapidamente encontra o fato "watches (brenda, greg)". Aplica a submeta àquele fato e volta à regra "happy (greg) if watches (brenda, greg)", onde sabe que o lado esquerdo da regra, "happy (greg)", pode ser utiliza-

do para concordar com a meta original, porque o lado direito da regra era "True" (verdadeiro).

INSTANCIAMENTO E LIGAÇÃO

Se você tentar a meta

```
happy (Who)
```

obterá a resposta "greg" depois de uma pesquisa muito parecida. A única diferença é que variável "Who" tem que sofrer "instanciamento" para "greg". Turbo Prolog vai encontrar o predicado "happy", verificar o arity, fazer o instanciamento da variável "Who" à constante "greg", recolocar a meta do lado direito da regra, combinar a nova meta e voltar à regra. Depois vai colocar o lado esquerdo da regra em "True", aplicar o lado esquerdo à meta, informar que a meta foi bem-sucedida, e mostrar o valor ligado da variável.

A palavra "ligação" é utilizada para estabelecer o valor de uma variável porque a atribuição de valor é freqüentemente temporária. Ao contrário do BASIC, onde uma declaração LET vai atribuir a uma variável um determinado valor até que outra declaração altere aquele valor, Prolog apenas liga valores a variáveis, desde que uma meta seja bem-sucedida. Caso uma meta falhe, mesmo temporariamente na lista de fatos e regras, a variável simples pode ser ligada e desligada a uma série de valores. Sempre que um valor é encontrado e que funciona, a variável está ligada àquele valor. Quando o valor falha no teste seguinte, ou regra, a variável é desligada deste valor, de modo que algum outro possa ocupar seu lugar.

Por exemplo, acrescente a regra

```
happy (eric) if watches (brenda, eric)
```

ao banco de dados. Coloque-o acima da regra

```
happy (greg) if watches (brenda, greg)
```

conforme indicado na Figura 5-13. (A ordem das linhas é importante porque Prolog procura de cima a baixo). Agora tente a meta

```
happy (Who)
```

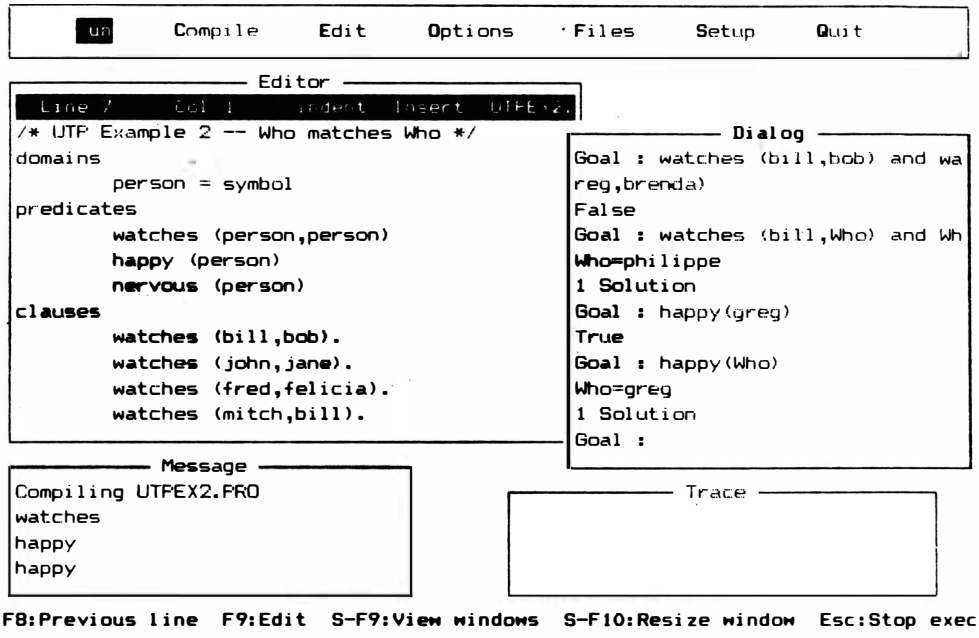


Figura 5-13 Retrocesso e regras

para ver quem está feliz e quem não está. Você vai ter a resposta "Who=greg" imediatamente. Mas como é que Turbo Prolog obteve esta resposta? O padrão de pesquisa é um pouco mais complicado desta vez, e em vez de fazer a sua descrição em detalhes, vamos explorar como você pode seguir o processo.

ACOMPANHAMENTO ("TRACING")

Quando você quiser dar uma espiada no progresso passo a passo de seu programa, em vez de ver apenas o resultado, poderá acompanhá-lo. Isto faz com que o ambiente de desenvolvimento pare o programa depois de cada passo e informe o que fez, onde está e quais são os valores das variáveis.

Uma das vantagens do Prolog como linguagem de programação é a sua enorme facilidade para acompanhar programas. Turbo Prolog tem outra vantagem que muitos Prolog anteriores não tinham: retenção dos nomes de variáveis. Outros Prolog, algumas vezes, vão converter seus nomes de variáveis (como "Who" e "Whom") para "A" e "B", "X" e "Y". Turbo continua com os nomes das variáveis utilizadas no programa, tornando o acompanhamento mais fácil de ser seguido.

Existem diversos métodos para seguir os passos que Turbo Prolog dá ao processar um programa. Alguns métodos exigem que você utilize comandos de compilação diferentes, mas estes não serão discutidos até um capítulo posterior. A janela Trace permite um acompanhamento fácil, que poderá ajudá-lo na compreensão do retrocesso e o sucesso ou falha de uma meta.

Acrescente as palavras

```
trace
```

ao seu programa (utpex2), acima da linha "domains" (veja Figura 5-14).

Isto vai informar ao compilador que você deseja processar o programa no modo Trace (acompanhamento). Depois processe e compile o programa.

Tente a meta

```
happy (Who)
```

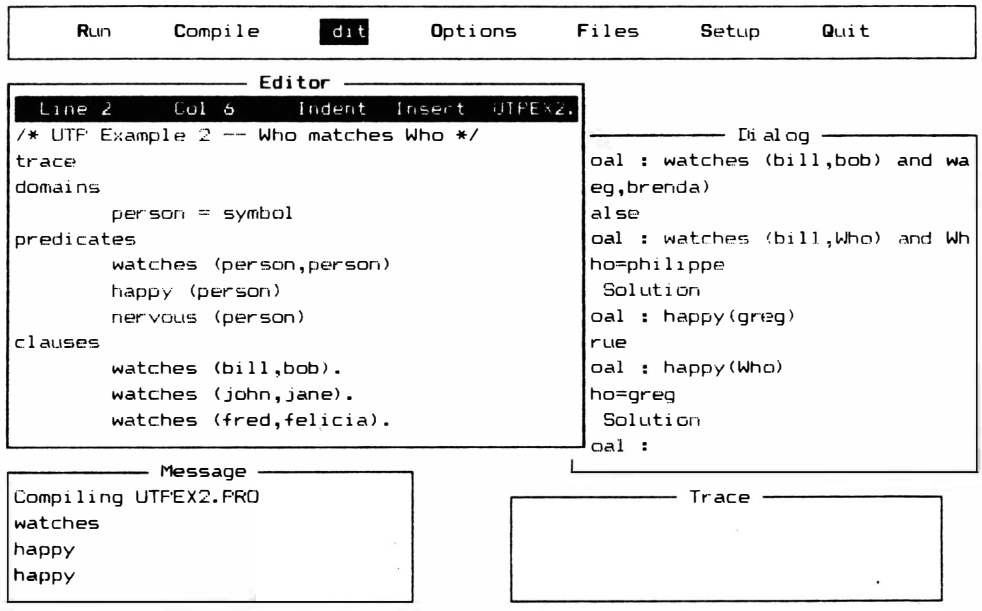


Figura 5-14 A declaração Trace (acompanhamento)

e observe o que acontece na janela Trace. Você verá (como na Figura 5-15) esta linha:

CALL: happy(_)

Isto significa que o primeiro passo que Prolog toma para satisfazer a meta é procurar um predicado "happy" com qualquer variável. Observe a variável anônima como argumento. Você deveria observar também que o cursor está no "h" de "happy" na janela Dialog. É onde a ação acontece neste momento, definindo a meta. Você verá o cursor se mover, durante a investigação, para mostrar aonde a lógica do Turbo Prolog está indo.

Acione, agora, a tecla de função especial F10 para ver o próximo passo. F10 sempre executa esta função quando você está no modo Trace. O cursor vai até o primeiro predicado que combina com a declaração CALL, que é a linha de regra

happy (eric) if watches (brenda, eric)

Tecle F10 novamente para ver o próximo passo. O cursor vai se mover para "w" de "wat-

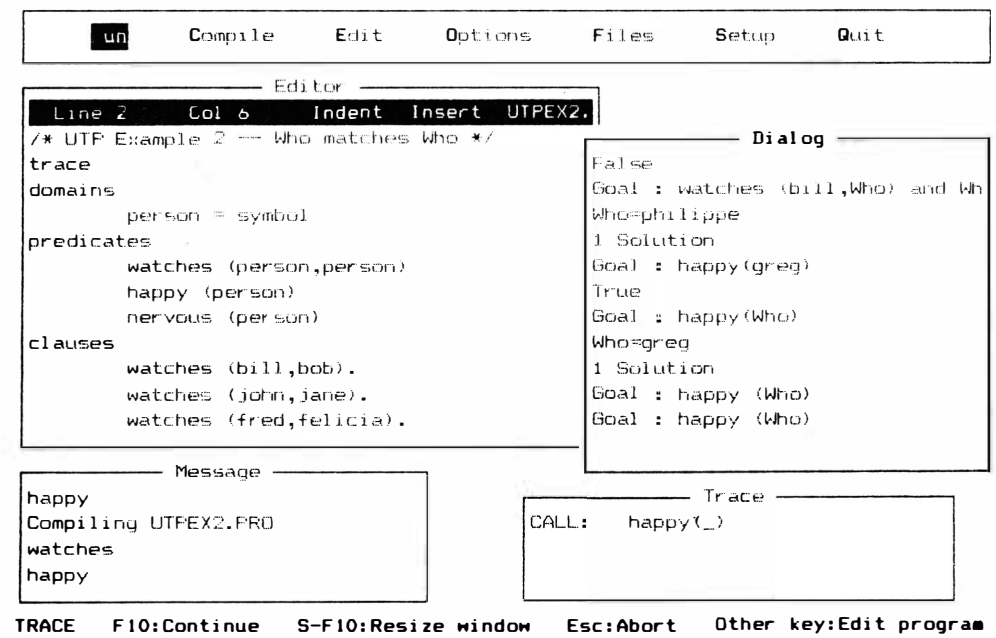


Figura 5-15 Utilizando a janela Trace (acompanhamento)

ches" porque Trubo Prolog combinou o lado esquerdo da regra com a meta original e agora sabe que tem de satisfazer o lado direito da regra. Na janela Dialog aparecerá (conforme indicado na Figura 5-15) a frase

CALL: watches ("brenda", "eric")

As palavras serão divididas porque a janela é muito pequena; Turbo Prolog continua as palavras na próxima linha, automaticamente. Quando estiver fazendo uma grande quantidade de acompanhamento, provavelmente vai querer usar o submenu Setup no menu principal para aumentar a janela Trace.

A linha "CALL" é a nova meta, temporariamente. Turbo Prolog agora vai tentar satisfazê-la procurando pelo banco de dados de cima para baixo, da esquerda para a direita, para combiná-lo com um fato conhecido ou outra regra que possivelmente possa ser satisfeita.

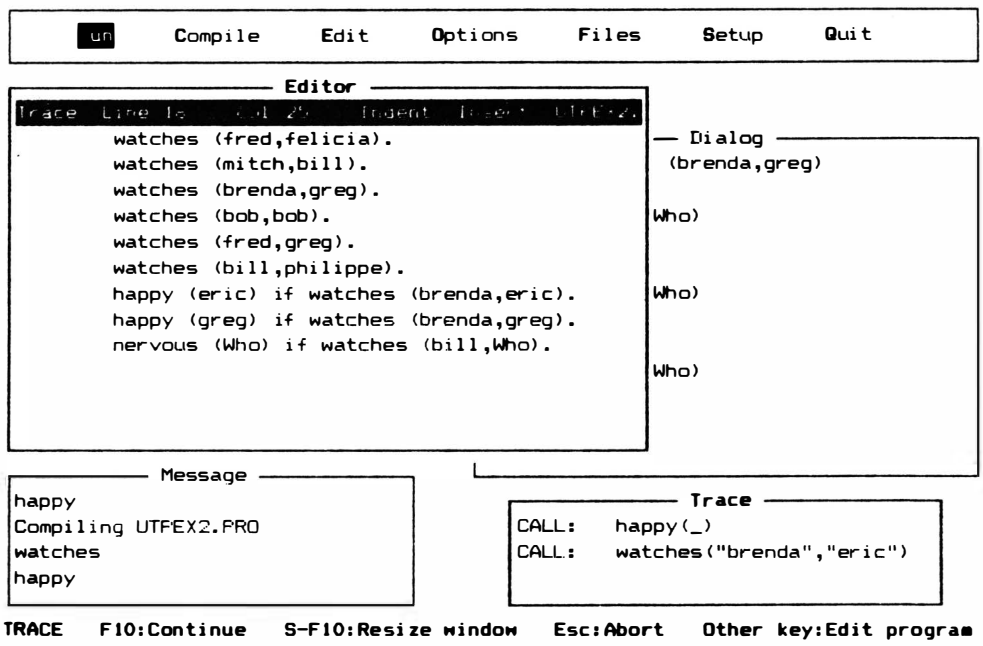


Figura 5-16 Estabelecendo uma meta nova,temporária

Tecle F10 novamente. O cursor volta ao primeiro fato no banco de dados. Tecle novamente. Aparecerá a tela indicada na Figura 5-17) com a linha

REDO: watches ("brenda", "eric")

na janela Trace. Turbo Prolog tentou combinar a nova meta contra o primeiro fato. Apesar do predicado e aridade estarem corretos, os valores dos argumentos na meta diferiam daqueles do fato. Para combinar, os argumentos têm de ser exatamente os mesmos. Turbo Prolog sabia que deveria tentar novamente para combinar a nova meta. Teria que refazer (REDO) o teste com outro fato ou regra.

Tecle F10 diversas vezes mais e observe que a mesma coisa ocorre. Turbo Prolog verifica cada linha para ver se combina com a meta; o cursor vai até a linha conferida, e a janela Trace lhe informa o que está acontecendo.

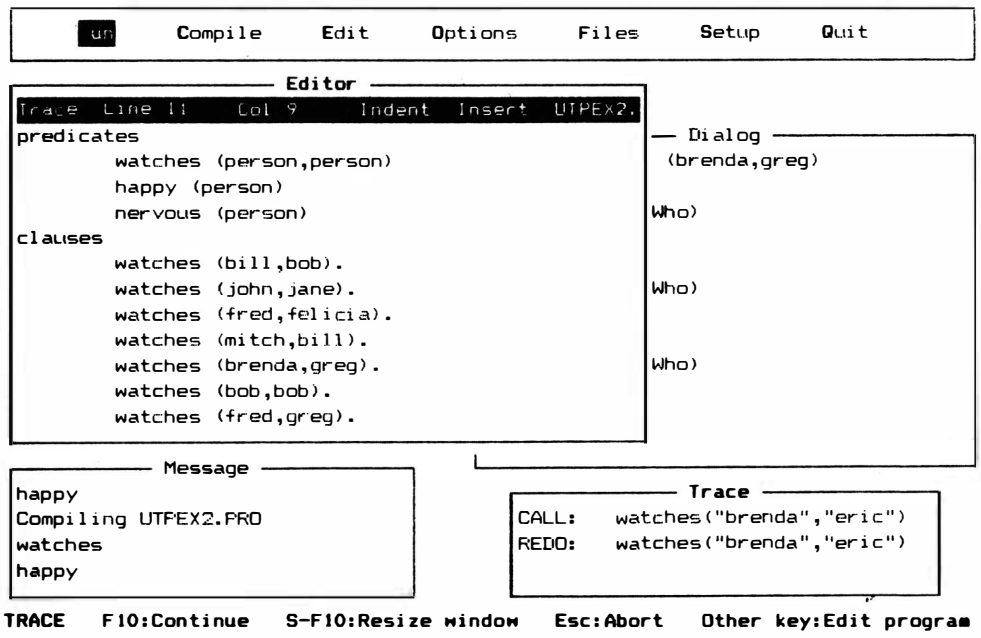


Figura 5-17 Tentando combinar a nova meta temporária, em acompanhamento

Finalmente, quando você teclar F10 verá a tela indicada na Figura 5-18. O retrocesso chegou ao estágio final. Turbo Prolog ficou sem fatos e regras a combinar com a nova meta de

watches (brenda, eric)

Ele percebeu que a nova meta falhou. Nesse ponto, voltou à regra

happy (eric) if watches (brenda, eric)

e descobriu que o lado esquerdo, "happy (eric)", também falhou, porque o lado direito não podia ser satisfeito. De modo que a meta temporária de

watches (brenda, eric)

foi abandonada e Turbo Prolog retrocedeu à meta original de

happy (Who)

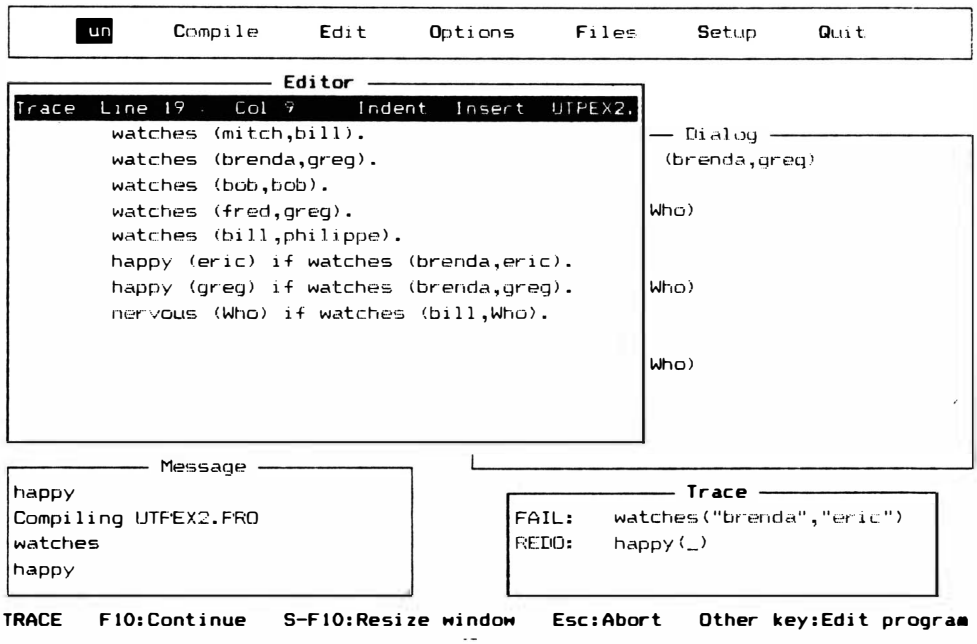


Figura 5-18 Retrocesso quando a meta temporária falha

Turbo Prolog marcou que esta meta foi testada contra todos os fatos e regras anteriores à regra que acabou de falhar, de modo que passou para o próximo fato ou regra. Encontrou

happy (greg) if watches (brenda, greg)

e começou a combiná-lo com a meta. Tecle F10 repetidamente e observe o progresso desta pesquisa. Turbo Prolog tomará

watches (brenda, greg)

como uma nova meta numa tentativa de satisfazer a regra toda. Pesquisará pelos fatos, tentando satisfazer à nova meta, e desta vez encontrará um fato que trará sucesso à nova meta. Neste ponto (veja Figura 5-19) verá

RETURN: watches ("brenda", "greg")

na janela Trace. Turbo Prolog encontrou um fato que combina com a meta temporária e

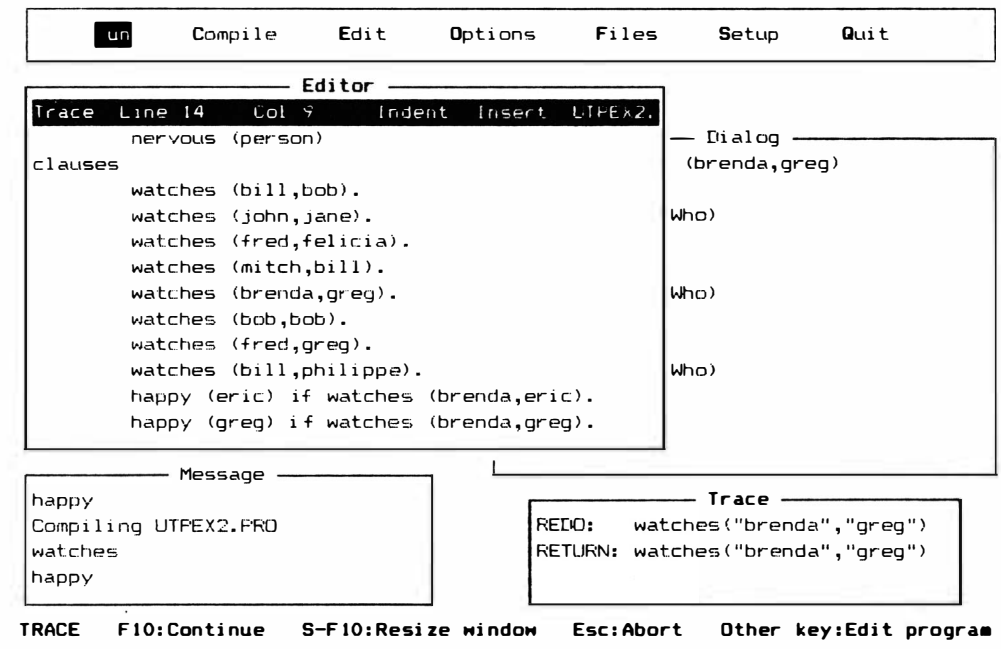


Figura 5-19 Satisfazendo a meta temporária

volta com o fato à linha de regra que incorpora a meta temporária. Com o lado direito da regra satisfeita, liberará a meta temporária, voltará ao lado esquerdo da regra, e fará o instanciamento da variável "Who" da meta original com o valor "greg". Com a meta original satisfeita, mostrará o resultado na janela Dialog, informará quantas soluções encontrou e estará pronto para uma nova meta.

O OPERADOR NOT

Outro operador que você encontrará em muitas regras Prolog é NOT. Acrescente ao seu programa a regra

```
happy (fred) if not (watches (fred, felicia))
```

depois da cláusula

```
happy (greg) if watches (brenda, greg)
```

conforme indicado na Figura 5-20. O parênteses adicional em torno do fato "watches (fred, felicia)" ajuda Turbo Prolog a saber que todo este fato tem o "not" aplicado a ele. Ainda, se você simplesmente colocar esta linha no final das cláusulas, obterá um erro 415, indicando que Turbo Prolog quer todos os predicados similares agrupados.

Cancele a palavra "Trace" no topo do programa (desta vez você não quer o acompanhamento). Depois, compile e processe o programa, e tente a seguinte meta:

```
happy (fred)
```

Obterá a resposta "False", porque Fred poderá ser feliz apenas se não estiver observando Felicia. Em outras palavras, o lado esquerdo da regra, "happy (fred)", é apenas verdade se o lado esquerdo, "watches (fred, felicia)", não for verdade. Mas o lado esquerdo é verdadeiro. Um fato mostra que é verdadeiro, e Turbo Prolog vai encontrar o fato. Portanto, o lado direito do fato falha como meta temporária; depois, o lado esquerdo falhará como uma concordância possível da meta original; pelo fato de nenhuma outra concordância ser encontrada para a meta, Turbo Prolog vai dizer que a meta original falha ou é "False".

O SÍMBOLO :-

Muitos programas Prolog, incluindo Turbo Prolog, também prmitem a utilização

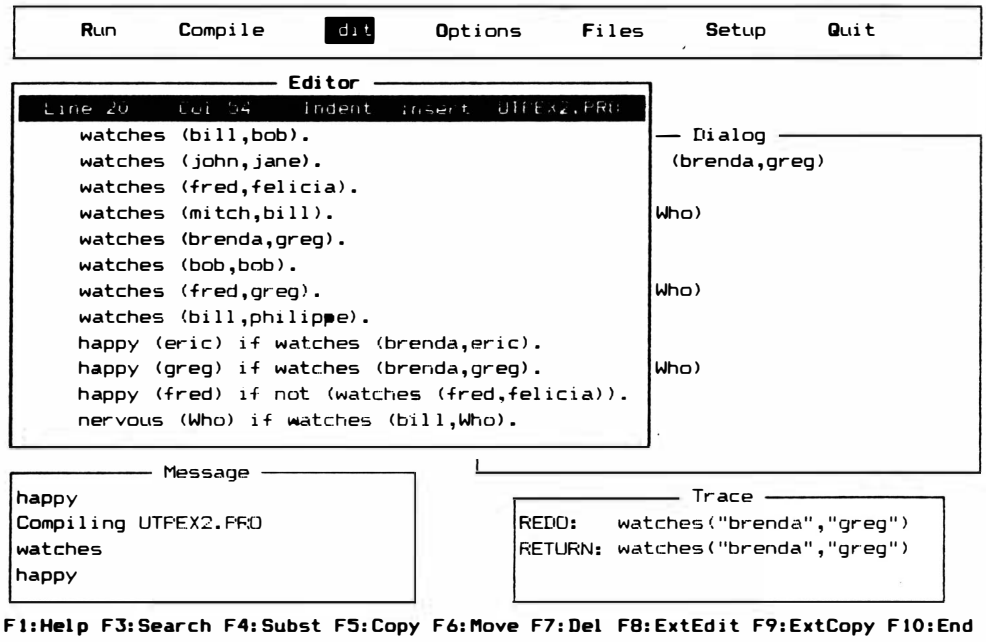


Figura 5-20 Utilizando o operador NOT em uma regra

do símbolo ":–" com o significado de "if". O que segue mostra uma das regras que você utilizou antes com o novo símbolo em seu lugar.

```
happy (eric) :– watches (brenda, eric)
```

CAPÍTULO 6

CONTROLE DE RETROCESSO: CUT (!) E FAIL

Turbo Prolog tem uma série de predicados intrínsecos, bem como comandos, que o tornam uma linguagem de programação de uso geral e não apenas uma implementação da lógica. Este capítulo mostra duas das mais importantes funções de controle de programa, disponíveis em muitas implementações de Prolog – "fail" e "cut" (!). No decorrer da explanação você vai conhecer os programas-exemplo que são parte do pacote do Turbo Prolog, os predicados procedurais para a apresentação de informação e metas externas *versus* internas. Se você já carregou e deu uma olhada nos programas do disco Biblioteca/Exemplo da Borland, poderá ignorar esta seção, mas mesmo que esteja familiarizado com os funcionamentos e utilizações dos predicados "fail" e "cut" (!), leia-a. Saber o que estes comandos são não é suficiente. Para criar programas Prolog práticos, será necessário incluir estes comandos nas posições certas. Você aprenderá também as diferenças entre metas internas e externas e dará uma primeira olhada nos predicados procedurais de apresentação, como "write". Estes serão abordados com maiores detalhes em um capítulo posterior. Finalmente, você utilizará a recorrência mesmo em programas mais simples, e não apenas em cálculos matemáticos. A recorrência é fundamental em aplicações de tratamento de listas e de linguagem natural.

PROGRAMAS-EXEMPLOS DA BORLAND

Turbo Prolog vem em dois discos: o disco Sistema/Programa e o disco Biblioteca/Exemplo. Até o presente momento você tem utilizado o disco Sistema/Programa neste livro. Este é o disco que contém o compilador principal, juntamente com as mensagens de erro, as informações de auxílio e coisas semelhantes.

O disco Biblioteca/Exemplo contém a demonstração de geografia em linguagem

natural, GeoBase, juntamente com 64 exemplos Prolog, mencionados no *Manual do Proprietário*. É aconselhável utilizar os Programas-Exemplos com freqüência, enquanto se aprende Prolog: carregue-os, leia-os modifique-os, estabeleça metas, siga a ação, e, algumas vezes, grave suas novas versões sob um nome diferente. Coloque comentários em seu código-fonte modificado, de modo a saber o que fez e por que.

A discussão a seguir mostra como carregar um programa do disco Biblioteca/Exemplo. Utilize o arranjo padrão – o disco Sistema/Programa do Turbo Prolog no acionador A e o disco Biblioteca/Exemplo no acionador B de um PC de dois floppies.

Do menu principal, tecle S de Setup e depois D de Directories.
A linha

```
PRO          directory A:\
```

deverá ser iluminada. Tecle RETURN, e depois digite

```
b:\
```

O "a:\ " que estava no quadro desaparece quando você começa a digitar o novo diretório padrão para o arquivo de trabalho. Agora, seus comandos Load chamam automaticamente o acionador B, a não ser que haja instrução em contrário. Tecle RETURN uma vez e ESC duas vezes. Isto colocará o novo diretório padrão na configuração ativa e depois o devolverá ao menu principal. Lembre-se de que a configuração ativa desaparecerá para sempre quando você deixar Prolog, a não ser que a grave em um arquivo .SYS (utilizando a opção Save Configuration do menu Setup).

Tecle F de Files e L de Load. Tecle RETURN sem digitar nenhum nome de arquivo em particular. Aparecerá um diretório dos arquivos no acionador B, que termina com a extensão .PRO. Eles são os programas-exemplos. Um deles, EXAMPL1.PRO, estará iluminado. Para carregar um arquivo iluminado, basta teclar RETURN. Você poderá movimentar a iluminação, utilizando as teclas de setas no teclado numérico.

Tecle RETURN para carregar EXAMPL1.PRO. Aparecerá na janela Editor, conforme mostrado na Figura 6-1. Acione a tecla E e estará também na janela Editor, pronto para editar o código.

CONTROLE DE RETROCESSO

Conforme discutido no Capítulo 5, retrocesso é um elemento essencial do Pro-

log, quando se pede a um programa para atender uma meta, procura de cima para baixo e da esquerda para direita pelas cláusulas que combinam com a meta. Caso encontre um beco sem saída, o programa retrocede o suficiente nas cláusulas para encontrar outro ramo que possa ser pesquisado.

Esta não é sempre a melhor forma de um programa trabalhar. Alguns programas consomem muito tempo recuperando informação desnecessária durante a procura daquilo que importa. Estes programas necessitam da ajuda do programador para diminuir este universo enorme de procura. Os computadores são vulneráveis à "explosão combinatória" que pode acontecer em Prolog com apenas alguns níveis de metas e submetas. A física atômica fornece um exemplo ilustrativo do que pode acontecer. Uma bomba atômica é baseada na reação em cadeia de uma desintegração nuclear: um único neutron rompe um átomo, resultando em dois neutrons, que rompem dois átomos, resultando em quatro neutrons, que rompem quatro átomos, resultando em oito neutrons, que rompem oito átomos e assim por diante. Este tipo de cadeia progride geometricamente. Seu programa de computador também poderá se tornar uma bomba devido ao aumento geométrico ou logarítmico de sua

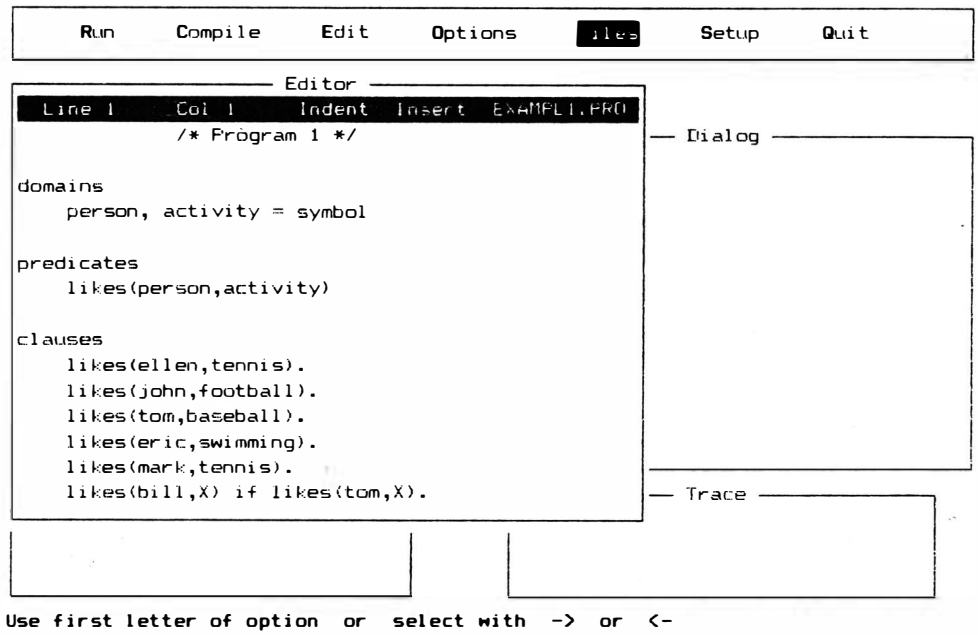


Figura 6-1 O programa EXAMPLE1.PRO carregado do disco Biblioteca/Exemplo

área de pesquisa. Se cada meta leva a duas submetas e cada submeta, por sua vez, leva a mais duas submetas, e assim em diante, o número de metas a serem satisfeitas pode rapidamente se multiplicar a níveis impraticáveis.

Este tipo de cadeia poderá rapidamente sobrepujar um computador e levar a tempos de pesquisa excessivamente longos. Imagine o número de tentativas de combinações que Prolog terá de fazer se uma meta contiver diversas cláusulas, cada uma das quais tendo de ser testada contra cada linha em um grande banco de dados. Depois, multiplique o tempo que isto vai levar, porque a procura vai, possivelmente, encontrar diversas regras e, portanto, submetas que, por sua vez, têm de ser comparadas com todo o banco de dados. Brevemente você estará aguardando que Prolog pare de procurar.

Como programador, você poderá ajudá-lo. Primeiro, poderá escrever as cláusulas em uma ordem que torne as pesquisas mais eficientes. Alguns exemplos simples disto serão apresentados neste livro, mas você deveria saber que uma compreensão detalhada de como organizar as cláusulas para a melhor eficiência é uma arte, e não uma ciência. Segundo, por ser um programador, você é mais inteligente que seu computador logo, pode dizer ao Prolog quando pular parte desnecessária de uma pesquisa. Você apenas insere alguns comandos especiais no programa. Estes podem direcionar a lógica e ganhar tempo de cálculo.

CUT (!)

Este comando (ou predicado especial) é denominado "cut" mas é escrito nos programas apenas como um ponto de exclamação. O comando "cut" se comporta essencialmente como um predicado inerente, sem argumento. Você o verá em diversos programas; ajuda muito a reduzir a árvore de pesquisa a um tamanho manipulável. A terminologia de "árvore" é amplamente utilizada na ciência de computação. Quando um padrão de procura cresce de uma única meta para submetas e subsubmetas, a metáfora de uma raiz principal, produzindo ramos e raminhas, é bastante natural.

Prolog utiliza o tipo de pesquisa em árvore denominado *pronfundidade-primeiro.* Prossegue da meta principal à primeira submeta, depois à primeira submeta abaixo desta, e assim por diante, até confirmar o nível inferior de submeta ou ter certeza de que a submeta falhou. Depois, sobe um nível de submeta (retrocedendo) e tenta comparar a próxima submeta no mesmo nível. Se a primeira série de submetas é parte de uma meta maior, que é um beco sem saída, Prolog não saberá disto antes de passar pelas submetas.

Para compreender de outra forma este tipo de procura, pense em uma árvore genealógica típica. Para simplificar a analogia, imagine que a sua família é formada apenas

por crianças fêmeas, e que cada uma tem duas meninas. Uma procura do tipo profundidade-primeiro, que começa com uma de suas bisavós, vai primeiro ao nível de seus dois filhos, uma delas sua avó. Prolog cuidaria dela primeiro. Depois, antes de continuar com a irmã dela, continuaria a se "aprofundar" um nível na árvore e trabalharia com um de seus filhos – sua mãe. Novamente, antes de continuar com a irmã de sua mãe, Prolog continuaria para outro nível de sua geração. Caso o seu nível ou geração fosse o último disponível (isto é, se não houvesse ramos abaixo de você), Prolog continuaria de você até sua irmã. Neste ponto, tendo terminado com a parte mais profunda de sua árvore em seu sub-ramo, Prolog retornaria à geração de sua mãe e continuaria com a irmã dela. Lembre-se, novamente, de que Prolog desceria até os descendentes de sua tia, passando por eles antes de voltar à sua tia e depois à sua avó. Mas o que aconteceria se você soubesse que as únicas informações úteis viriam de sua mãe e dos filhos dela? Caso não quisesse nenhuma das informações que poderiam vir de uma procura que pudesse vir das irmãs de sua mãe ou de sua avó, porque permitir que Prolog perca seu tempo ali? Você pode interromper este desperdício com "cut".

O comando ! ("cut") evita o retrocesso, funcionando como uma válvula unidirecional. A procura pode passar por ele por meio da cláusula mais à esquerda mas não pode voltar a tratá-la de novo vindo da direita. Quando encontramos !, ele congela as variáveis ligadas de uma determinada procura de meta ou submeta nos valores atuais. Isto significa que pode funcionar em uma única regra ou mesmo entre regras. Caso outras cláusulas procuradas pela meta provoquem sua falha, Prolog não pode retroceder ao ponto antes do !, na procura da meta, e procurar outros valores para aquelas variáveis – mesmo que aquelas outras variáveis permitam que a meta seja bem-sucedida.

Se você escrever uma regra com cláusulas múltiplas como

```
A if B and C and ! and D
```

o ! faz o Prolog parar de encontrar valores múltiplos para "B" e "C". Vamos ver como isto funciona.

Na procura de uma meta, Prolog desce pela seção de cláusula até chegar na cabeça ("A") da regra. Para satisfazer "A", terá que satisfazer primeiro o lado direito da regra. Primeiro Prolog toma "B" e procura encontrar uma concordância para ele. Caso Prolog não encontre uma concordância, toda a regra falha (esta é uma regra "and", em que todas as cláusulas da direita têm de ter sucesso). Caso encontre uma concordância para "B", ligue as variáveis de "B" às da concordância e depois continue para "C". Poderão existir outras possibilidades nas cláusulas (como outros fatos e regras) que possam

satisfazer a "B". Isto não importa, porque Prolog quer apenas uma satisfação – um conjunto de variáveis ligadas – por ora.

Chegando a "C", Turbo Prolog procura uma concordância para ele. Novamente, se não encontrar uma concordância, a regra toda falha, inclusive "A". Caso encontre uma concordância, ligará as variáveis em "C" com o valor concordante e continuará até a cláusula "!". Tanto as variáveis em "B" como as variáveis em "C" agora estão ligadas aos primeiros valores encontrados que satisfariam às cláusulas "B" e "C".

A cláusula "!" automaticamente tem sucesso. Não tem nenhum argumento a ser considerado, e não precisa ser combinada com nada. Automaticamente, é verdadeira. Mas tem mais um efeito: as variáveis ligadas a "B" e "C" estão presas a seus valores atuais. Logo veremos o que isto faz.

Turbo Prolog continua para "D". Vai procurar todas as possibilidades de "D", procurando encontrar alguma coisa no banco de dados que concorde com "D". Se o fizer, a regra será bem-sucedida, porque todas as cláusulas que vieram antes ("B", "C" e "!") foram bem-sucedidas. Caso não encontre uma solução para "D" a regra toda falhará, porque "!" impede que ela retroceda para encontrar outros valores de "B" e "C".

Lembre-se de que, com retrocesso normal, Turbo Prolog poderia observar o próximo valor de "C" e depois voltar para encontrar um valor que funcionasse para "D". Poderia encontrar um valor "D" deste tipo, uma vez que o valor de "C" fosse diferente. Mas o "!" não permite esta possibilidade. Esta regra precisa parar com a primeira solução que encontrar para "B" e "C".

Dê uma olhada na Figura 6-2. Você pode ver nesta figura duas coisas que são diferentes de figuras anteriores. A janela Editor foi aumentada, utilizando o menu Setup, de modo que você possa ver todo o programa. Ainda, a regra "is_a_duck" utiliza indentação para mostrar as submetas em seu lado direito.

Este programa permitirá que você descubra o que "is_a_duck" (é um pato) é quando utilizar a meta

```
is_a_duck (animal)
```

procurando em seu banco de dados para satisfazer a condição composta da regra "is_a_duck". Entretanto, se você acrescentar "cut"(!) a ele, conforme mostrado na Figura 6-3, obterá apenas o primeiro nome de pato. A regra não será capaz de retroceder à primeira submeta ("sounds_like_a_duck") depois de passar pelo "cut". Neste caso, a posição de "cut" não faz muita diferença. Você poderia tê-lo colocado depois de qualquer uma das

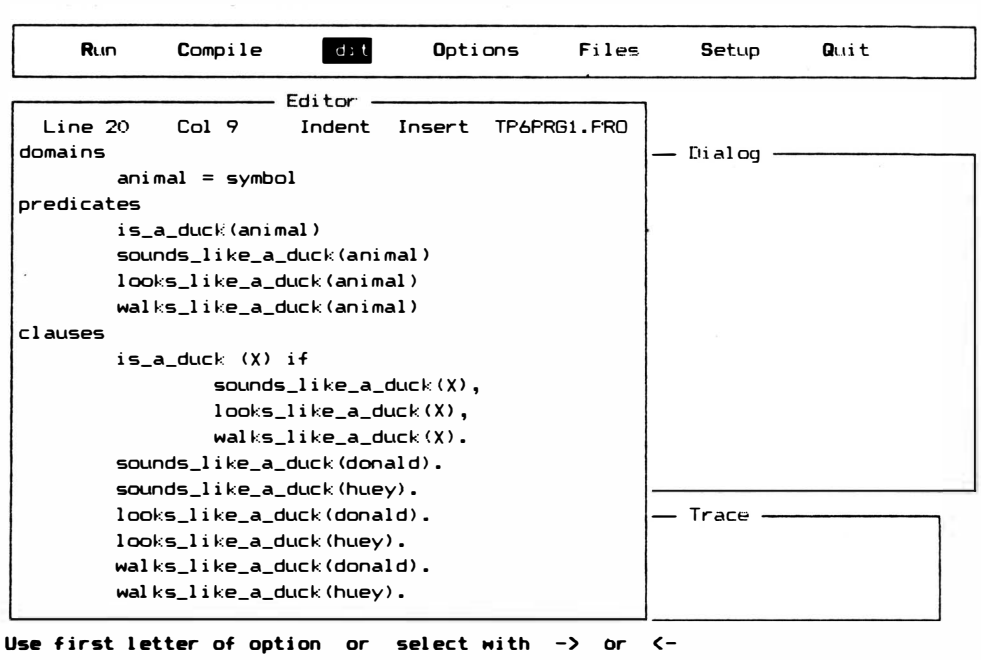


Figura 6-2 Uma janela Editor aumentada e programa-amostra com uma regra indentada

submetas da regra "is_a_duck" e continuaria descobrindo apenas o nome de uma única ave.

Se você colocar antes de todas as submetas (como mostrado na Figura 6-4), obterá os dois nomes do pato, porque Turbo Prolog poderá livremente retroceder nos três "sounds_like_a_duck", "looks_like_a_duck", "walk_like_a_duck". Poderá encontrar uma resposta ("donald") para o primeiro, e depois confirmar que é verdadeiro também para o segundo e terceiro, e depois informar-lhe o nome na janela Dialog. Pelo fato de estar no final da regra, Turbo Prolog retrocede para fazer o mesmo com o outro nome. Uma vez que não precisa retroceder pelo "cut", as duas soluções estão corretas.

Observe as Figuras 6-5 e 6-6. Este programa Prolog é feito para uma grande festa que está acabando. Você quer saber quem pode ir para casa com segurança e quem precisará de arranjos especiais. Um computador on-line contém a informação a respeito de endereços, licença para dirigir e registro do carro, bem como uma lista das pessoas que estão na festa e aqueles que são amigos. Na medida em que as pessoas chegam à porta da frente, você faz um teste de sangue para detecção de álcool para ver-se estiveram beben-

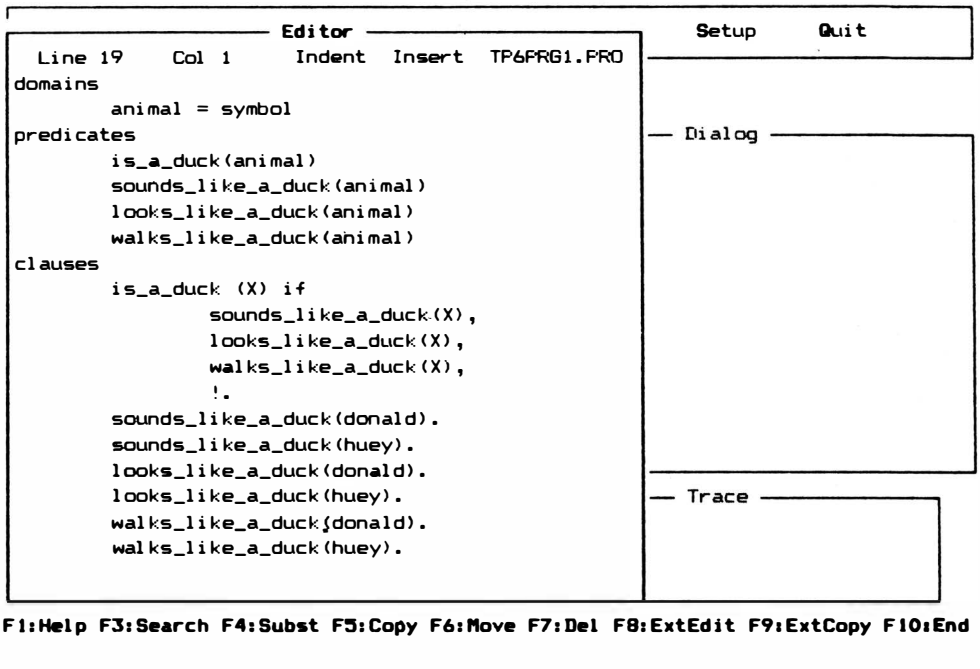


Figura 6-3 Acrescentando um "cut (!)" a um programa

do. Depois, coloca seus nomes e o resultado do teste no computador que está processando este programa, e (ignorando todos os problemas que poderiam ocorrer para a obtenção deste banco de dados fictício) espera por uma resposta de quem pode ir para casa. Observe que as submetas da regra principal

```
can_get_home(Name1) if
        safe_to_drive(Name1),
        own_car(Name1);
        know_safe_driver(Name1);
        live_close_enough_to_walk (Name1).
```

incluem diversos ponto-e-vírgula. Este símbolo pode ser utilizado em lugar da palavra "or". Alguém pode ir para casa ("can_get_home"– caso seja um motorista cuidadoso e possua um carro, ou se conhece um motorista seguro ou se mora o suficientemente perto para poder ir a pé. Pelo fato de tudo não caber na janela Editor visível, será apresentado em duas partes. Tente compilar e processar este programa.

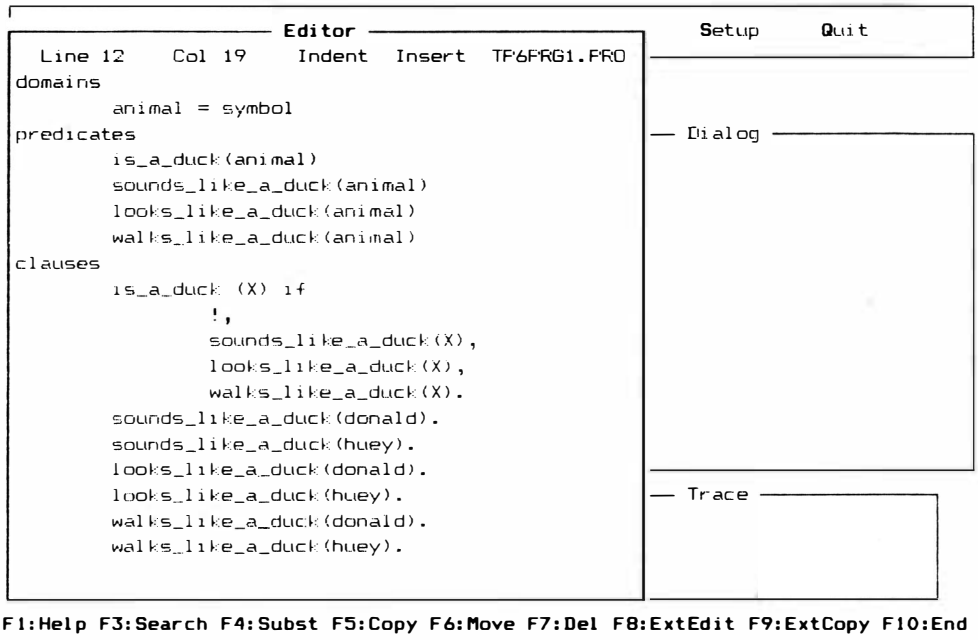


Figura 6-4 Cláusulas de retrocesso, utilizando "cut"

Se você colocar o nome de uma determinada pessoa como meta, obterá o tipo de resultado indicado na Figura 6-7. Você será informado se esta pessoa pode ou não ir para casa sem ajuda adicional. Se você utilizar uma variável (conforme mostrado na Figura 6-8), poderá obter uma lista com todas as pessoas que podem sair. Isto é mais rápido do que processar uma pessoa por vez.

Mas há um problema: "freda" aparece duas vezes, porque ela tem duas maneiras diferentes de ir para casa. Isto não é um problema com uma amostra tão pequena, mas imagine que você estivesse fazendo uma lista de algum programa semelhante e estivesse utilizando um banco de dados muito grande. Por exemplo, se estivesse fazendo uma mala direta, poderia gastar dinheiro em excesso e mesmo aborrecer algumas pessoas atingidas, caso seus nomes aparecessem várias vezes em sua lista.

"Cut" aparecerá para salvá-lo. Você não quer retroceder pelas submetas da regra "can_get_home". Veja a Figura 6-9 para saber onde poderíamos colocar "cut", e depois tente novamente a variável na meta, como mostrado na Figura 6-10. Lembre-se dos atalhos: você pode utilizar F10 para recompilar e reprocessar depois da edição, e poderá depois utilizar F8 para escrever a mesma meta que a anterior.

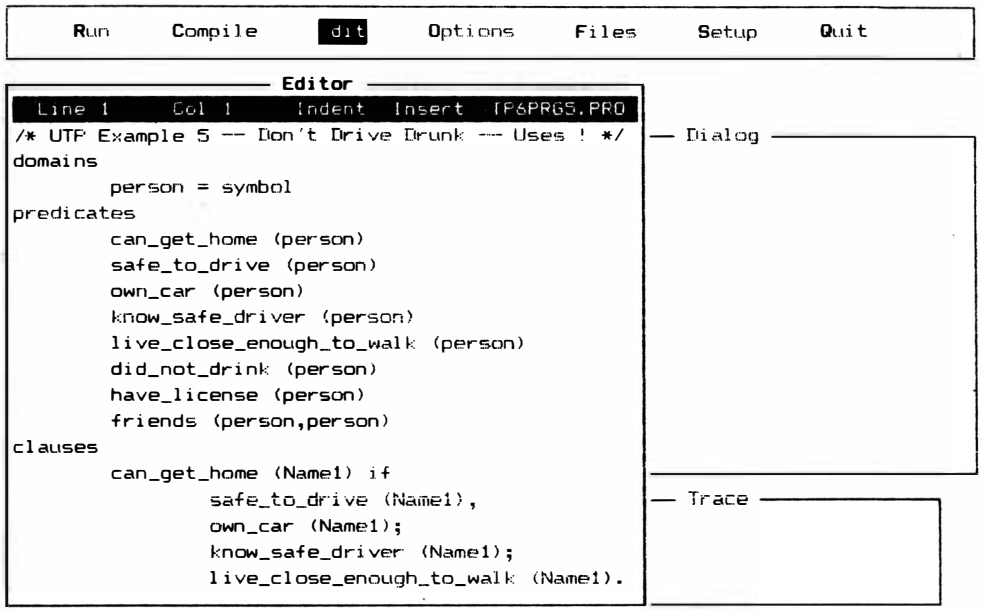


Figura 6-5 Programa amostra "Don't Drive Drunk" (Parte 1)

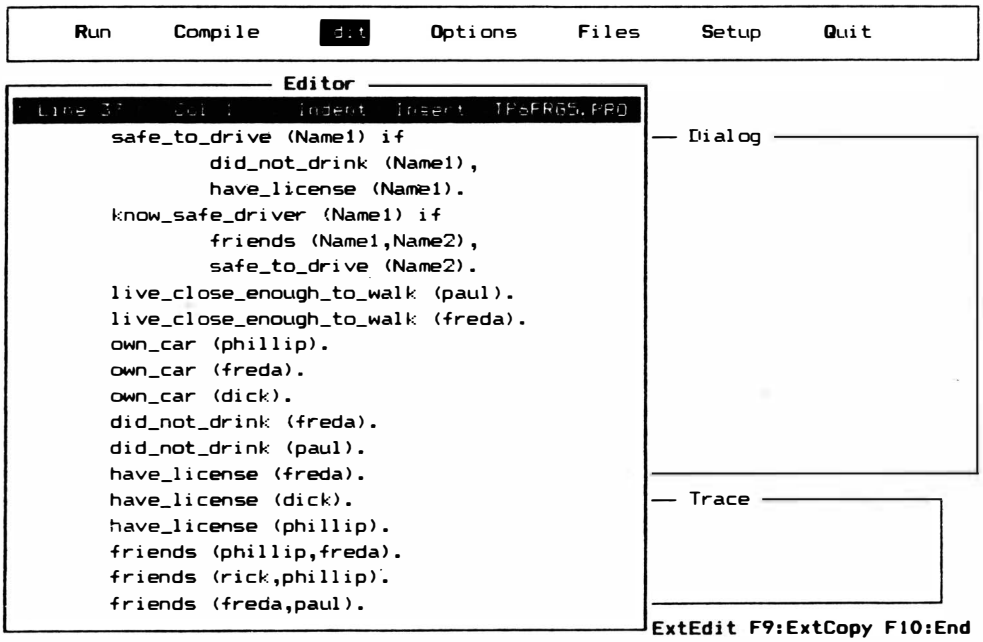


Figura 6-6 Programa-amostra "Don't Drive Drunk" (Parte 2)

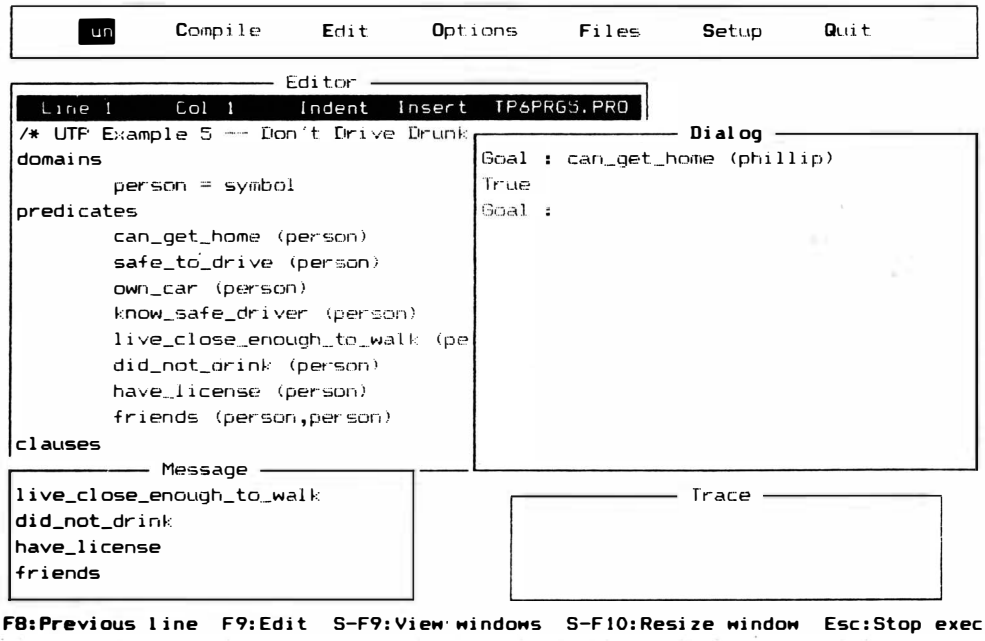


Figura 6-7 Resultados típicos do programa "Don't Drive Drunk"

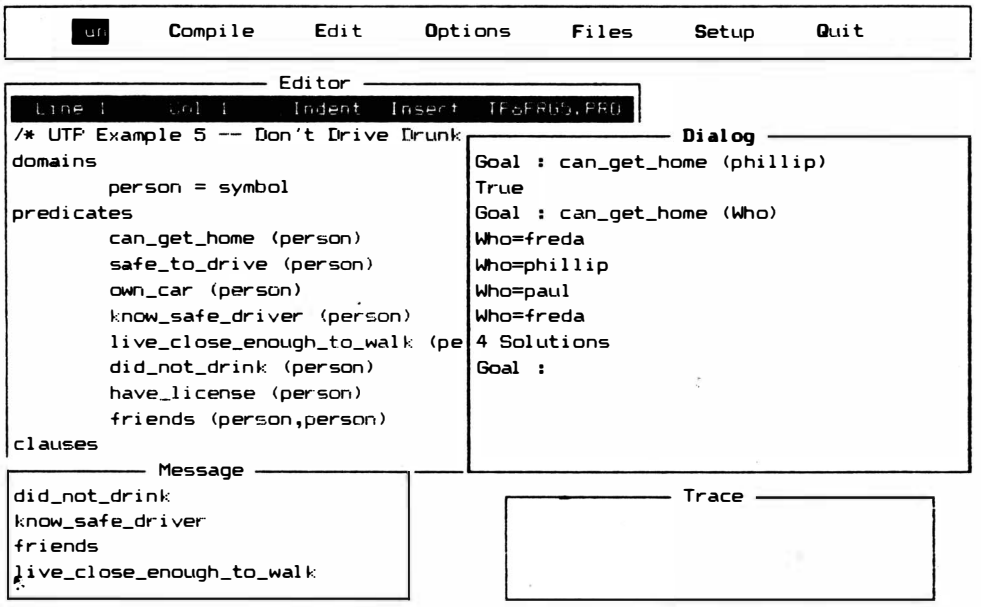


Figura 6-8 Utilizando uma variável na meta

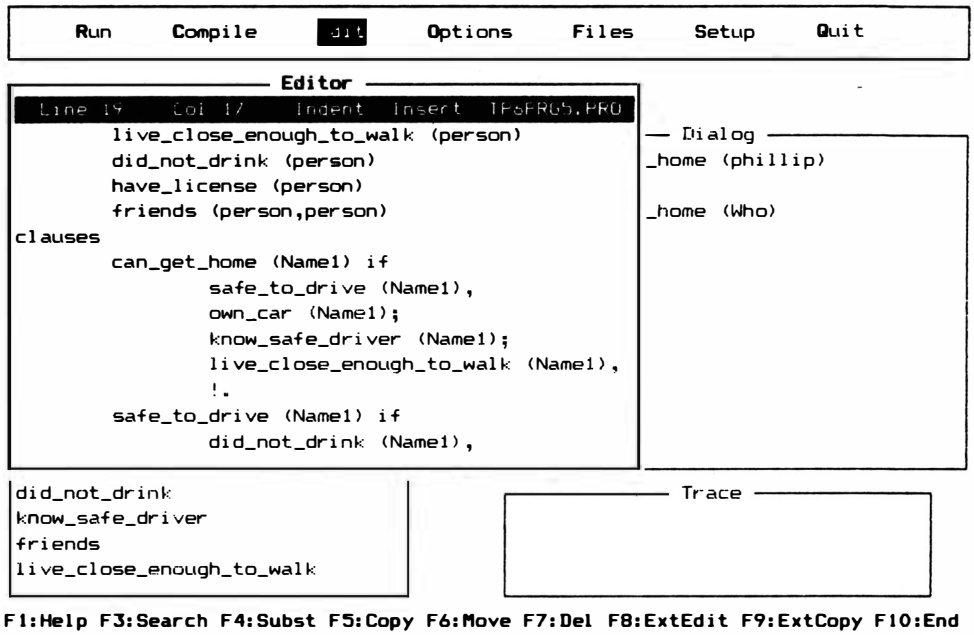


Figura 6-9 Inserindo um "cut" para variáveis de retrocesso (linha 19, coluna 17)

Conforme mencionado anteriormente "cut" impede que você seja afogado pelas soluções, tanto em termos de quantidade de saídas como o tempo gasto para a obtenção da meta. Novamente, observe que "cut" poderia ser um exagero quando é utilizado em programas tão pequenos como este, que é executado quase que imediatamente e tem percursos de pesquisa pequenos. Mas, mesmo programas pequenos podem fugir ao controle se você não utilizar "cut".

Por exemplo, se um programa utiliza recorrência (que você vai aprender no próximo capítulo), uma única cláusula pode mantê-lo girando por um longo tempo. Outro exemplo onde um "cut" seria apropriado é um conjunto de cláusulas profundamente interrelacionadas que imprimiria uma lista de resposta quando você quisesse apenas uma. Este é o caso do programa dos convidados da festa.

Como observação final, você deveria saber que Clocksin e Mellish advertem que um bom estilo de programação para programas legíveis substitui ! com "not", sempre que possível. Isto gera programas mais lentos que utilizam mais memória, mas os programas serão muito mais fáceis de serem lidos. A escolha é sua, evidentemente, em cada situação.

FAIL

Outro comando inerente para controlar o retrocesso, "fail" é um predicado sem argumento que automaticamente falha e força um retrocesso. Em um certo sentido é o oposto de "cut", que elimina o retrocesso.

UM TIPO NOVO DE PREDICADO

Suponha agora que Prolog tenha um predicado inerente que imprime informações na tela (ele o faz). Da mesma forma que com outros predicados, Prolog tenta fazer com que este seja bem-sucedido. Mas, neste caso, em vez de procurar em outro local pelo material de concordância, Prolog sabe que tudo que precisa fazer para satisfazer o predicado é realmente executar algumas funções. Tais predicados são realmente "procedimentos" e impedem que Turbo Prolog seja uma linguagem de programação declarativa estritamente lógica. Tornam Prolog útil também para tarefas de programação do dia-a-dia.

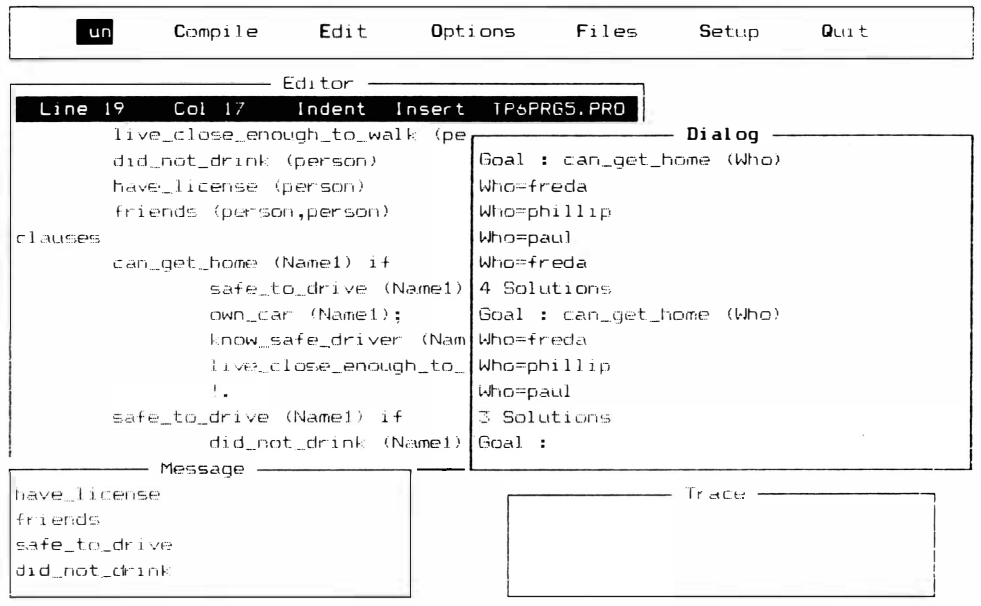


Figura 6-10 O resultado da colocação do "!"

Aqui está um exemplo de tal predicado: "write". Em uma regra, pode ter a seguinte forma:

A if write ("Hello Earth")

Quando Turbo Prolog encontra "A" nas cláusulas, sabe que tem de satisfazer o lado direito da regra para satisfazer "A". A única ação necessária para satisfazer o lado direito é escrever a frase

Hello Earth

na tela, na posição corrente do cursor. O predicado – write ("Hello Earth") – pode ser presumido como correto e, por sua vez, "greetings" pode ser presumido como verdadeiro. Veja a Figura 6-11 para um exemplo de um programa que utiliza esta regra e predicado.

Você verá que este programa é muito simples. A declaração do predicado nem

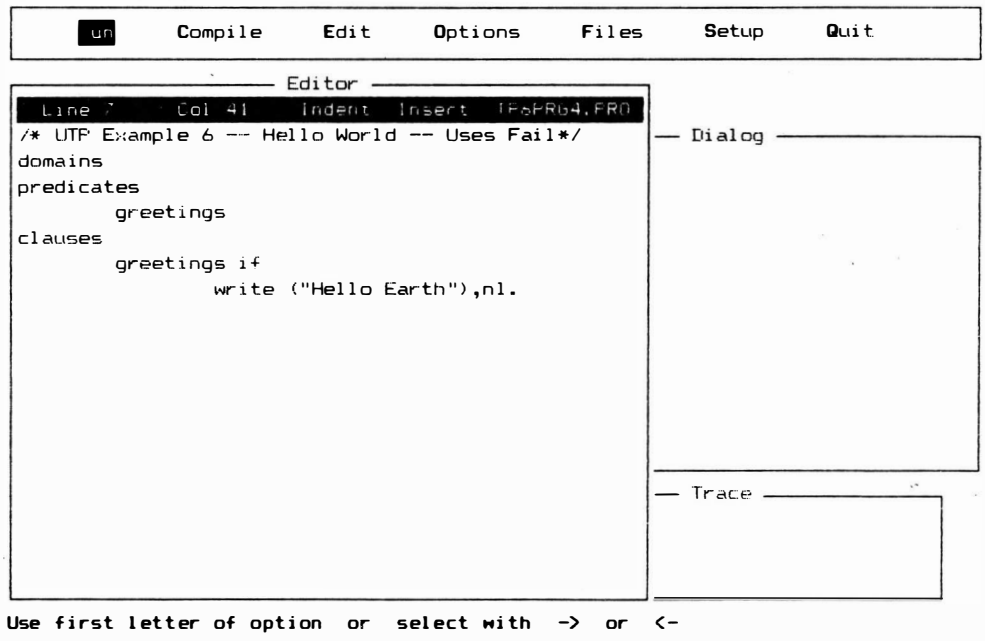


Figura 6-11 Predicado e regra amostra

precisa tratar com um argumento. A seção de cláusulas contém apenas uma única regra:

```
greetings if write ("Hello Earth"), nl.
```

Obs.: Em nl, o segundo caractere representa a letra l (ele) minúscula. Não confundir com o número 1.

Se você processar este programa e apresentar-lhe "greetings" como meta, Turbo Pascal vai imediatamente até esta regra para tentar um sucesso encontrando um conjunto de variáveis que satisfaçam a lógica. Assim que verifica o lado direito, Prolog percebe que pode satisfazer a primeira parte daquele lado pela impressão da frase

```
Hello Earth
```

no quadro Dialog. Depois, o restante da regra precisa ser satisfeito, porque a vírgula em

```
,nl.
```

indica ser isto uma regra composta com um "and" no meio. A segunda parte da regra, "nl.", precisa ser satisfeita. O "nl" significa "linha nova" (new line). Para satisfazer este predicado inerente, Prolog precisa apenas mover-se para a próxima linha do quadro Dialog antes de imprimir qualquer outra coisa. O ponto no final indica que é o fim da regra. Ambas as partes do lado direito foram bem-sucedidas: write e newline. Portanto, o lado esquerdo será bem-sucedido e a meta foi satisfeita. Prolog devolve a resposta "True".

Pode parecer que este programa é muito pequeno para termos todo este trabalho, mas tente as três coisas seguintes:

- Retire a vírgula e o "nl" da regra. Depois, compile e processe novamente o programa e utilize "greetings" como meta. Você vai ver que o comentário "True" no quadro Dialog é impresso imediatamente depois da frase "Hello Earth".

- Coloque a vírgula e "nl" de volta. Depois, coloque a palavra "trace" em cima da palavra "predicates". Depois, compile e processe o programa, utilizando novamente "greetings" como meta. Em seguida, utilize a tecla F10 para acompanhar o programa passo a passo. Você verá Prolog passar pelas partes do programa.

- Finalmente, acrescente uma seção de meta ao programa, conforme descrito na discussão seguinte.

METAS INTERNAS "VERSUS" EXTERNAS

Conforme mencionado anteriormente, existem dois tipos de metas em Turbo Prolog: externa e interna. Muitos Prolog aceitam apenas metas externas. Até aqui, neste livro, utilizamos metas externas. Uma meta *externa* é aquela que você propõe ao programa na janela Dialog depois que o programa foi editado e compilado e está em processamento. Já vimos as metas externas em ação. Uma vez que a meta digitada foi provada verdadeira ou falsa, o programa solicita outra meta.

Uma meta *interna* tem a mesma sintaxe que uma meta externa. A diferença é que você a coloca em uma seção de metas no código do programa. Em seguida, você completa a edição e compilação e processa o programa. Depois que o programa tiver sido processado, automaticamente continua para provar ou não a meta e dar a resposta. Depois, o programa termina, e você não lhe é solicitado colocar outra meta.

Acrescente as duas linhas

```
goal
    greetings.
```

ao programa que você tem utilizado, bem abaixo da declaração de predicado; depois, faça a compilação e processe-o. Você tem que operar a barra de espaço para encerrar o processamento do programa. Você verá o resultado na janela Dialog mas não saberá se a meta é verdadeira ou falsa.

DE VOLTA À FAIL

A possibilidade de digitar "greetings" no computador de comunicação não lhe satisfaz. Você quer enviar uma série de mensagens ao mundo, e não quer escrever uma série destas regras. Em seu lugar, poderá utilizar uma única regra que inclui o predicado "fail" e é depois seguida pelos dados necessários em fatos.

O programa da Figura 6-12 contém países em suas cláusulas. Se processar o programa sem "fail", você verá que apenas o primeiro país será saudado. A Figura 6-12 mostra o programa e os resultados com o predicado "fail" no fim da regra principal. O predicado obriga Prolog a retroceder até o começo da meta e a começar de novo as procuras de concordância. Cada vez que Prolog encontra uma, satisfaz novamente a primeira submeta, ligando a variável ao nome do país; satisfaz a segunda submeta, imprimindo "Hello" e o nome do país; e depois falha no predicado "fail", porque diz "False" no fim. Fail obriga ao retrocesso, assim como "cut" o elimina.

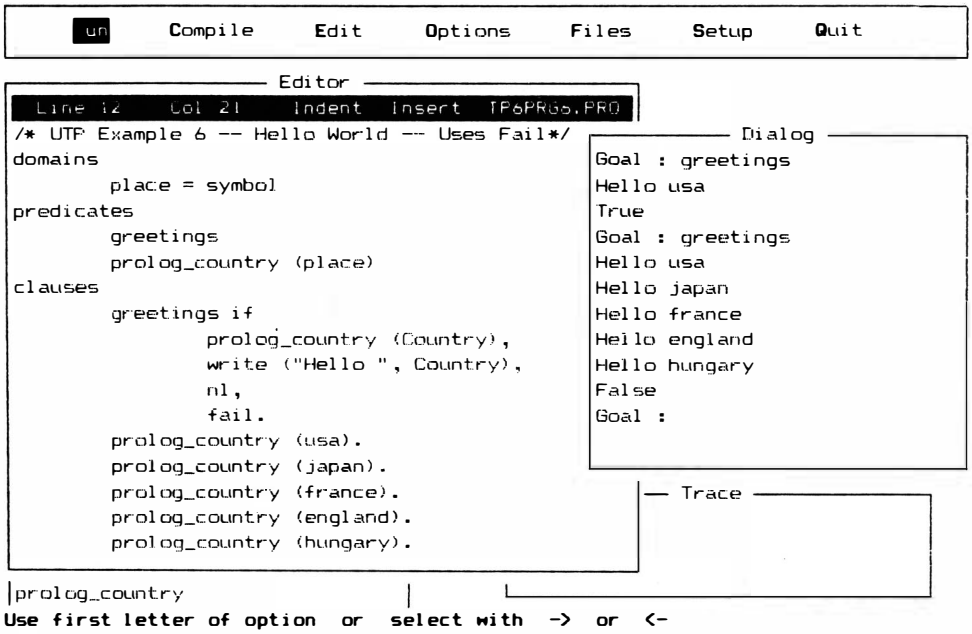


Figura 6-12 Programas que utilizam um predicado "fail"

CAPÍTULO 7

DOMÍNIOS, ARITMÉTICA E RECORRÊNCIA

Prolog não foi projetado originalmente para ser adequado aos problemas matemáticos, mas seu porte desajeitado não durou muito. As pessoas que se habilitam em uma linguagem de computação normalmente querem utilizá-la para um número cada vez maior de tarefas. Os recursos aritméticos foram logo acrescentados ao Prolog, e o compilador Turbo Prolog tem um conjunto completo.

A discussão da aritmética neste capítulo é preparada por uma discussão mais específica da seção de domínio de um programa Turbo Prolog. Esta parte do código normalmente o defende de erros de confusão entre tipos de argumentos. Ajuda, também, muitos programas a serem processados com maior rapidez, porque estes tipos de declaração, com freqüência, permitem que o compilador determine com mais precisão com que tipo de dados está tratando e o espaço de memória que precisa reservar para os dados.

Recorrência é um recurso da programação de computador que permite a definição de algumas situações em termos simples, que se referem a si próprios. Pode parecer confuso até tê-los em ação, mas não o ignore. É fundamental tanto para o processamento aritmético como o de lista.

Se você estiver acostumado à verificação de tipos em uma linguagem como Turbo Pascal, pode ignorar a seção "Domínios" deste capítulo. Provavelmente deverá ler a discussão aritmética, mesmo que conheça bem a matemática de computação. O significado dos operadores aritméticos lógicos do Prolog não é sempre tão evidente como possa parecer. Mas, assim que você pegar o estilo da aritmética em Prolog, estará apto a terminar rapidamente a seção. A seção entitulada "Recorrência" será novamente redundante para os bons conhecedores da ciência de computação, mas a sua utilização em Prolog é, definiti-

vamente, algo que você não vai querer perder. O Capítulo 8 fornecerá uma discussão mais aprofundada da recorrência.

VERIFICAÇÃO DE TIPOS: DECLARAÇÕES DE DOMÍNIO E PREDICADO

O compilador Turbo Prolog utiliza verificações de *tipos* para melhorar a depuração dos programas e diminuir o espaço de memória necessário para programas. Programas de computadores operam com vários tipos de dados: caracteres, inteiros, números reais e outros tipos de informações estruturadas. Muitas operações estão limitadas quanto ao tipo de dados com os quais podem operar. Se for utilizado o tipo errado de dados, o resultado poderá ser uma resposta sem sentido. Infelizmente, estes erros algumas vezes são sutis e podem, por isso, tornar difícil a depuração de um programa.

Se os tipos de dados a serem utilizados são especificados de modo que a linguagem saiba, toda vez que encontrar um dado, se o dado é real ou inteiro, símbolo ou caractere, o compilador pode verificar o código-fonte para ver se você está usando um tipo inadequado de dados em uma operação ou predicado. Se estiver, rapidamente receberá uma mensagem de erro.

Turbo Prolog consegue sua ajuda na verificação de tipos pedindo duas declarações a respeito da situação do programa: solicita os predicados e os domínios. Estas seções são encabeçadas pelas palavras "predicados" e "domínios" no programa Turbo Prolog.

PREDICADOS

Na seção de predicados de um programa Turbo Prolog, você vai informar ao compilador quais predicados utilizará (além dos predicados inerentes) e depois especificar o número e os nomes dos argumentos. Você não tem de fazer isto para predicados inerentes, uma vez que Turbo Prolog já os conhece. Outro tipo de declaração de predicado é o tipo global. Se você quer que uma declaração se aplique a mais de um módulo, deverá colocá-la em uma seção encabeçada pelas palavras "predicados globais". (Veja o Capítulo 12 para maiores detalhes a respeito de predicados globais.)

Uma típica declaração de predicado poderá ser:

```
predicatename (argument1, argument2) [nomedopredicado (argumento1, argumento2)]
```

Entretanto, os predicados não precisam ter nenhum argumento. Você também poderia utilizar apenas

```
predicatename                [nomedopredicado]
```

nas declarações. Finalmente, você poderá ter múltiplas declarações de predicados, onde o mesmo predicado é definido mais de uma vez, cada vez com domínios de argumento diferentes, como em

```
predicatename (argument1, argument2) [nomedopredicado (argumento1 .....]
predicatename (argument3, argument4, argument5)
```

ou

```
has_toes (person, number)        [temdedosdepé (pessoa, número)]
has_toes (animal, number)        [temdedosdepé (animal, número)]
```

Neste caso, você será capaz de utilizar o predicado nas duas formas. O argumento diferente também não precisa ser do mesmo tipo de domínio.

DOMÍNIOS

Assim que Turbo Prolog sabe quais predicados vão com cada argumento, precisa saber com quais domínios – que tipos de dados – os argumentos trabalham. Os mesmos argumentos podem ser utilizados com diversos predicados. Por exemplo, você poderá ver o seguinte:

```
has_toes (person, number) [temdedosdepé (pessoa, número)]
has_fingers (person, number) [temdedos (pessoa, número)]
```

O compilador não precisa de uma declaração de domínio diferente para cada ocorrência de "person" ou "number", mas vai querer saber o que estes dados representam. Isto é feito na seção de domínio, que é a primeira seção no programa Turbo Prolog, precedido apenas por diretrizes de compilador e comentários. (Se você quer que uma declaração de domínio se aplique a mais de um módulo de programa, você a coloca na seção de domínios global, encabeçada pelas palavras descritas no Capítulo 12.)

A Tabela 7-1 mostra os tipos de domínio-padrão. Você pode também definir seus próprios tipos (que seriam não-padronizados, evidentemente). Estes tipos são declarados nos quatro formatos diferentes seguintes. Lembre-se de que as palavras "argument1", "argument2", e assim por diante, são apenas termos genéricos. Um programa real teria palavras de argumento que você escolheria.

Tabela 7-1 Tipos de domínio padrão

Symbol	**(Símbolo)**
Existem dois tipos de símbolos: (1) um grupo de caracteres consecutivos – letras, números e travessão (isolados, e não colocados diretamente abaixo de uma letra ou número) – que começam por letra minúscula.	

Exemplos

fred	[fred]
freds_car	[carro_de_fred]
the_thing_in_the_cave	[coisas_no_porão]

2) Um grupo de caracteres consecutivos (letras ou números) que comaça e termina com aspas ("). Este tipo é útil se você quer começar o símbolo com uma letra maiúscula ou se quer incluir espaços entre os caracteres no símbolo.

Símbolos e cadeias de caracteres são muito parecidos, e muitos objetos podem ser utilizados em qualquer um dos dois domínios, mas Turbo Prolog interpreta-os de forma muito diferente. Os símbolos ocupam mais memória do que as cadeias de caracteres, mas as concordâncias com símbolos são realizadas mais rapidamente quando o programa é processado.

Exemplos

"fred"	["fred"]
"freds car that I drove into a tree"	["Carro de fred que joguei contra uma árvore"]

String (Cadeia de caracteres)

Qualquer grupo de caracteres consecutivos (letras ou números) que começa e termina com aspas ("). É o mesmo que o segundo tipo de símbolo descrito acima. Lembre-se, entretanto, de que símbolos e strings não são tratados da mesma forma pelo Turbo Prolog.

Exemplos

"fred"	["fred"]
"fred is a *&#%$#@ * + "	["fred é um..."]
"When in the course of human events it becomes necessary"	["Quando no decurso de eventos humanos se faz necessário"]

Integer (Inteiro)

Qualquer número inteiro de (e inclusive) –32.768 a 32.767. O limite existe porque os inteiros são armazenados como valores de 16 bits. Quinze destes bits representam o número, e o décimo sexto representa o sinal do número.

Exemplos

9
–200
21444

Real (Número real)

Qualquer número real na faixa +/– 1E –397 até +/–1E + 308. O formato inclui estas opções: sinal,

número, ponto decimal, fração, E (para indicar um expoente), sinal para o expoente, expoente. Os números reais podem ir de muito simples para muito complexos.

Exemplos

3
–3.1415296525E–218

(Ambos são reais, mas seria melhor em muitos casos definir o primeiro como inteiro em vez de real. O inteiro ocuparia menos memória.)

Char (Caractere)

Qualquer caractere isolado da tabela ASCII padrão, colocado entre apóstrofes (').

Exemplos

't'
'X'
'&'

File (Arquivo)

Para declarar um domínio de arquivo, você tem que declarar os nomes de arquivos simbólicos que vai utilizar. Você pode ter apenas uma única declaração de domínio de arquivo em um programa, mas aquela declaração poderá conter diversos nomes de arquivos simbólicos. Todos os nomes de arquivos simbólicos têm que começar com letra minúscula. Existem dois tipos de arquivos.
(1) Arquivos predefinidos. Estes são os arquivos que sempre estão disponíveis em Turbo Prolog. Você não precisa declará-los; de fato, não deve. Prolog sabe que estão ali.

Exemplos

printer
keyboard
screen
coml

Observe que "printer" refere-se à porta paralela e "coml" se refere à porta serial. Estes exemplos são sua única escolha para declarações de domínio de arquivo.
(2) Arquivos definidos pelo usuário. Estes são arquivos que você nomeia. Qualquer nome Turbo Prolog que não é reservado pode ser utilizado.

Exemplos

thisfile	[estearquivo]
thatfile	[essearquivo]
the_other_file	[o_outro_arquivo
employee_records_1986	[registros_funcionários_1986]

- **Simple** Este não é o nome oficial – estas declarações são as normais que você encontraria em quase todos os programas. Utilizam uma equivalência da forma

```
argument1 = domain
```

com o domínio escolhido da lista-padrão-inteiro, char, real string ou símbolo. Um exemplo que você viu diversas vezes neste livro é

```
person = symbol
```

Se você tiver várias destas declarações em seqüência, poderá encurtar seu trabalho de digitação, utilizando vírgulas entre os argumentos que têm o mesmo tipo de domínio. Em vez de

```
argument1 = domain
argument2 = domain
argument3 = domain
```

Você poderia escrever uma única linha

```
argument1, argument2, argument3 = domain.
```

Em termos práticos, em vez de

```
price = integer                [preço = integer]
height = integer               [altura = integer
position = integer             [posição = integer]
```

poderia utilizar

```
price, height, position = integer        [preço, altura, posição = integer]
```

- **List (lista)** É utilizado para estruturas de listas (que serão discutidas posteriormente). As listas são feitas de múltiplos objetos agrupados em um só tipo de dados. Uma declaração como

```
thislist = things*
```

significa que os objetos individuais na lista "thislist" (que é um dado único, mesmo que

seja uma estrutura formada por dados menores) estão no domínio de "things". O asterisco é lido como "list", de modo que "things*" é lido como "thingslist". O domínio com asterisco poderá ser um domínio-padrão (veja Tabela 7-1) ou um domínio definido pelo usuário.

- **Compound (composto)** Novamente, você ainda não foi apresentado aos objetos compostos, de modo que deve ignorar isto até o próximo capítulo. Objetos compostos se parecem com relacionamentos dentro de relacionamentos. Eis um exemplo: predicate

  (argument1, predicate2 (argument3, argument4))

  Um exemplo prático poderia ser

  ```
  person (first_name, last_name, dates (birth, start_employment))
  [pessoa (nome, sobrenome, datas (nascimento, entrada_em_serviço))]
  ```

  O "predicate2" (ou "dates") é denominado um *functor*. Para declarar um objeto composto, você declara o functor e depois declara os domínios do argumento daquele functor. O lado direito de uma declaração deste tipo poderá também ter diversas partes, cada um uma declaração de um único functor que pode operar no predicado original.

- **File (arquivo)** Esta declaração é utilizada quando um programa precisa se referir a arquivos por meio de nomes simbólicos. Apenas uma única declaração de arquivo é permitida. Esta declaração utiliza uma equivalência, com ponto-e-vírgula, para detalhar os arquivos a serem utilizados:

  ```
  file = thisfile ; thatfile ; otherfile
  [file = estearquivo ;essearquivo : outroarquivo]
  ```

  A melhor maneira de compreender a declaração de predicado e domínio depois desta introdução é olhar os programas-exemplos e escrever (e depurar) alguns exemplos simples. Também, declarações de listas e arquivos serão discutidas em maiores detalhes nos capítulos dedicados às listas e aos arquivos.

ARITMÉTICA

Turbo Prolog permite que você utilize numerosas operações matemáticas para operar com inteiros (entre –32.768 e 32.767) e números reais (de +/– 1E–307 até +/– 1E–308). Os números reais poderão ter sinal, mantissa, um ponto decimal, uma parte fracionária, outro sinal, o caractere "E" (para estabelecer o expoente) e o expoente. O si-

nal, a parte fracionária com um ponto decimal e o expoente são opcionais. Os inteiros serão convertidos automaticamente em números reais se a operação aritmética requiser. Turbo Prolog manuseia números com predicados lógicos. Além dos predicados aritméticos e matemáticos oferecerem as possibilidades normais para o sucesso ou falha de uma meta, os argumentos destes predicados podem ser avaliados, com o uso dos predicados para a obtenção de um resultado matemático. Para que você se sinta à vontade, Turbo Prolog lhe oferece também a capacidade de representar predicados aritméticos-padrões como operadores *infix* (operadores que aparecem entre constantes ou variáveis). Fornece, também, uma lista generosa de operações logarítmicas, trigonométricas e entre bits que podem ser utilizados com formatos-padrões de predicados.

POSIÇÃO DO OPERADOR

Na matemática, em geral, um símbolo de operação não precisa ser escrito entre as constantes ou variáveis, mesmo sendo esta a maneira como a maioria de nós aprende a usá-lo. A forma *infix*

```
y * 4
```

poderá parecer natural, mas não é a única maneira na qual esta operação pode ser escrita. Os sinais + e –, antes de um valor, apenas indicam se é positivo ou negativo. São operadores unários e, portanto, não têm que ficar entre dois termos. Poderia ainda tomar a forma

```
*y 4
```

que é denominada notação *prefixada*. Um operador prefixado comum é o sinal de menos, uma vez que é utilizado para significar um número negativo. Outra possibilidade é escrever uma operação com uma notação posfixada, como a seguir:

```
y 4*
```

O símbolo fatorial (!) é um operador matemático normalmente posfixado. Infelizmente, utiliza o mesmo símbolo (o ponto de exclamação), que o predicado "cut" em Prolog. Turbo Prolog não inclui a função fatorial como recurso aritmético inerente, de modo que você não precisa se preocupar com a confusão que possa ocorrer entre estes dois símbolos.

Prolog poderia representar esta multiplicação como um relacionamento (*) entre dois objetos (a variável y e a constante 4). Aqui está como ficaria:

```
*(Y,4)
```

Nesta forma ou na forma *infix*, a operação não significa que os valores são combinados na forma simbólica. A operação apenas mostra o relacionamento entre os objetos. Quando Turbo Prolog verificar o predicado contra a meta, o relacionamento será avaliado.

IGUALDADE

O predicado da *igualdade* é vital para a operação aritmética em Turbo Prolog. O sinal de igual (=) é utilizado para indicar que o predicado tem sucesso se os objetos dos dois lados da igualdade coincidem. Em vez de dizer que as duas coisas são iguais, o predicado diz que o Prolog pode tentar torná-las iguais olhando o valor da instância correto. O operador está fazendo uma verificação lógica; não está estabelecendo um valor.

Existem, também, desigualdades que podem ser escritas como operadores *infix* em Turbo Prolog. Estes operadores relacionais estão listados na Tabela 7-2. Todos os operadores relacionais funcionam da mesma forma, como será demonstrado em exemplos posteriores.

Tabela 7-2 Operadores relacionais padronizados

Símbolo	Significado
<	menor que
>	maior que
=	igual a
< =	menor ou igual a
> =	maior ou igual a
< > ou > <	diferente de

OPERADORES ARITMÉTICOS

A Tabela 7-3 relaciona as funções aritméticas básicas em Turbo Prolog. Mostra também o tipo do resultado. Desde que você se lembre de que estes são avaliados de forma diferente em Prolog do que em uma linguagem procedural ou na aritmética-padrão, você pode utilizar os operadores no mesmo tipo de função que utilizaria naquelas outras circunstâncias.

A Tabela 7-3 mostra também a ordem de precedência para os operadores aritméticos. Se uma cláusula utiliza diversos operadores, eles são avaliados na ordem indicada nesta tabela.

Tabela 7-3 Operadores aritméticos padrão *(infix)*

Símbolo	Operação	Prioridade[1]	Op1	Op2 Domínio	Op3
+	Adição	1	I	I	I
			I	R	R
			R	I	R
			R	R	R
–	Subtração	1	I	I	I
			I	R	R
			R	I	R
			R	R	R
*	Multiplicação	2	I	I	I
			I	R	R
			R	I	R
			R	R	R
/	Divisão	2	I	I	R
			I	R	R
			R	I	R
			R	R	R
mod	Módulo (o resultado é o resto da divisão dos operandos 1 e 2)	3	I	I	I
div	Divisão (o resultado é o quociente da divisão dos operandos 1 e 2)	3	I	I	I
+	Positivo (unário)[2]	4	I		I
		4	R		R
–	Negativo (unário)[2]		I		I
			R		R

[1] "Prioridade" refere-se à ordem na qual as operações são executadas. Operações de prioridade mais alta são executadas antes das operações de baixa prioridade. Operações entre parênteses são executadas antes das que estão fora do parênteses, e as operações são executadas da esquerda para a direita em uma expressão. Tanto os operadores "mod" como "div" têm prioridade 3.

[2] "Unário" refere-se à utilização do símbolo como uma especificação do sinal de um valor simples.

ADIÇÃO E SUBTRAÇÃO

Dê uma olhada no programa mostrado na Figura 7-1. Isto demonstra a utilização do operador de adição. A única cláusula é a "do_it" (faça). É também o único predicado que não é inerente e é, portanto, o único predicado que precisa de uma declaração. Tam-

bém, o predicado "do_it" não tem nenhum argumento; assim, não precisa de nenhuma declaração de domínio.

A regra "do_it" tem sucesso apenas se a variável "Result" é instanciada ao valor 5 e se este valor é depois mostrado na janela Dialog. Pelo fato de "do_it" ser também a meta, esta seqüência é exatamente a seguida pelo Prolog em sua vontade de satisfazer a meta. A frase "Press the SPACEbar" que aparece no resultado (veja Figura 7-2) está lá porque uma meta interna foi utilizada.

Agora, dê uma olhada na Figura 7-3. O operador de subtração foi acrescentado à regra. Se você processar este programa, obterá o resultado indicado na janela Dialog da Figura 7-3. Por que o resultado deste programa é exatamente o mesmo do último programa? Por que o resultado da subtração não foi mostrado? Porque isto não é BASIC ou Pascal, e o sinal de igual não funciona como uma atribuição que pode dar valores novos às variáveis.

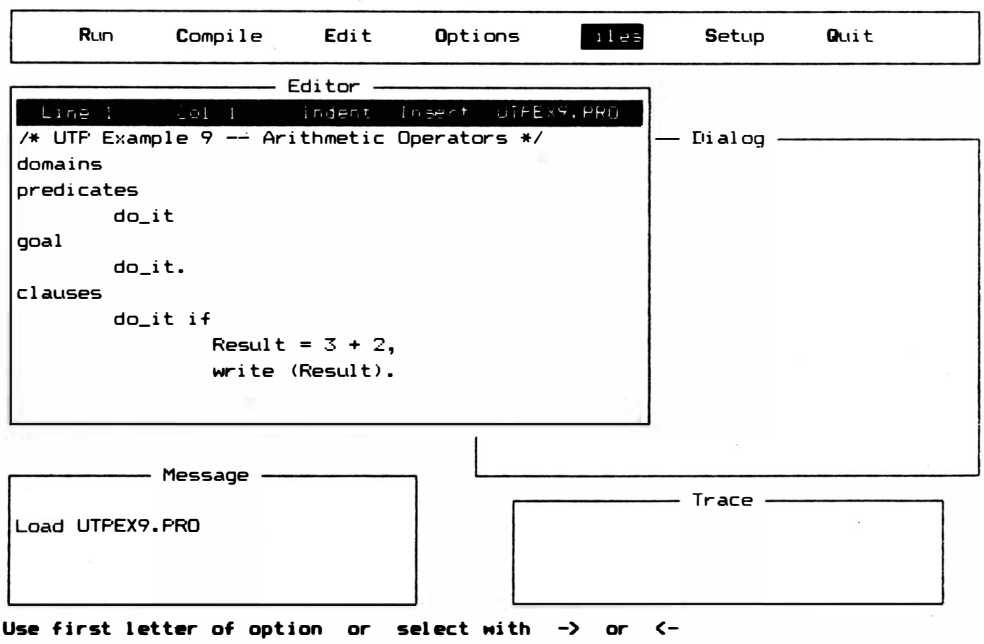


Figura 7-1 Utilização do operador de adição

Acrescente a palavra "trace" na parte superior do programa (insira-a em uma linha nova acima de "domains") e depois recompile e processe o programa. Utilize a tecla F10 para passar pela lógica do programa. Observe os passos na janela Trace. Os sete passos principais da pesquisa são mostrados na Figura 7-4. Aqui estão os passos e a explicação do que aconteceu em cada um:

CALL: goal ( )

Prolog começou chamando a meta.

CALL: do_it ( )

Determina então que a meta interna é o predicado "do_it". Prolog encontra uma cláusula que concorda com o predicado e descobre que a cláusula é o começo de uma regra. Depois

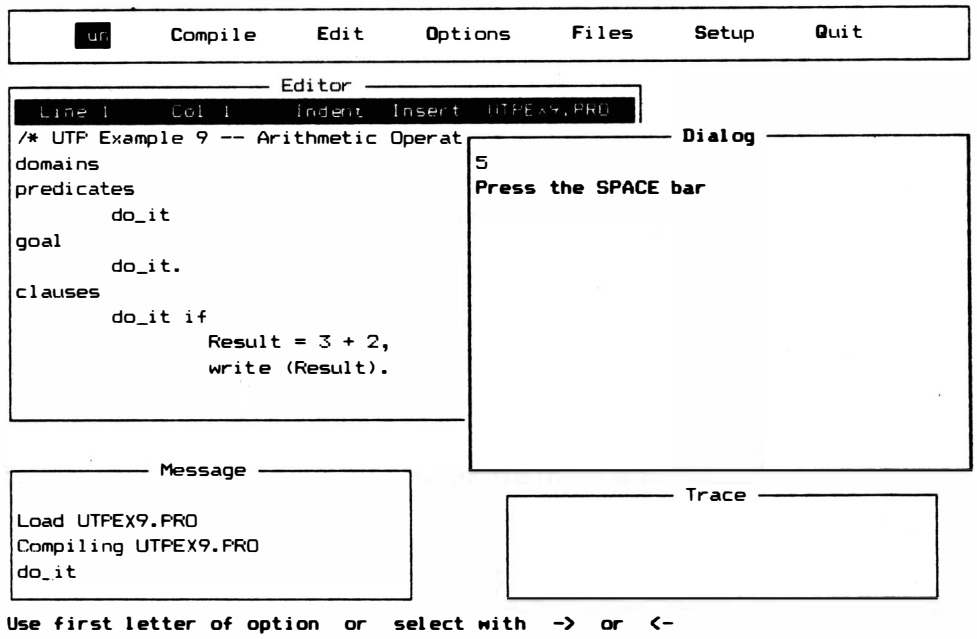


Figura 7-2 A frase "Press the SPACEbar" como resultado de uma meta interna

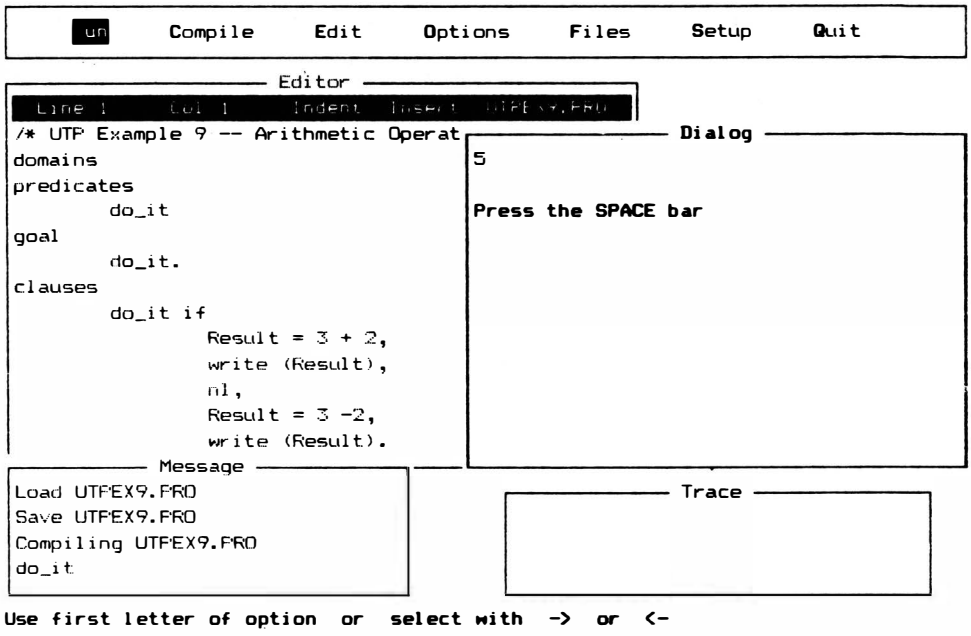


Figura 7-3 Utilização do operador de subtração

vai para o lado direito da regra e tenta atender às cláusulas daquele lado.

5=5

Para a cláusula do lado direito, Prolog descobre um operador de igualdade entre a variável (aquele da esquerda) e duas constantes (à direita). Prolog precisa encontrar um valor instanciado para a variável ("Result") antes que possa decidir se esta cláusula pode ou não ser bem-sucedida. O operador de igualdade juntamente com o operador de adição pede a Prolog que utilize a soma aritmética das constantes. Ele instancia "Result" àquele valor, e depois vai a próxima cláusula. As vírgulas entre as cláusulas significam "and". Significam que todas as cláusulas do lado direito têm de ser satisfeitas para que a regra como um todo seja satisfeita.

write (5)

Esta cláusula pede ao Prolog para instanciar o valor de "Result". Ele assim o faz. É tudo o que é necessário para satisfazer à cláusula, de modo que Prolog continua em frente. (Ob-

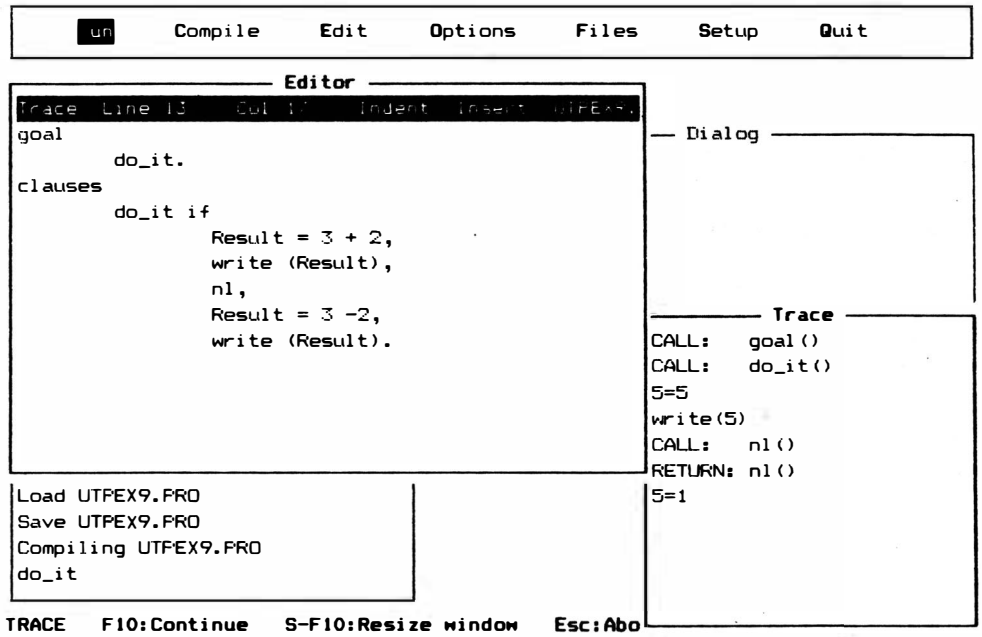


Figura 7-4 Principais passos do acompanhamento

serve que aspas são utilizadas no texto deste livro para mostrar as palavras como aparecem na tela. Aspas não devem ser utilizadas a não ser quando está indicado. Por exemplo, se a palavra "Result" estivesse entre aspas dentro do parênteses, o comando write a teria escrito como uma palavra em vez de escrever seu valor.)

CALL: nl ( ) RETURN: nl( )

Certifique-se de ter digitado "nl", que significa "new line", com uma letra l minúscula e não o número 1. O predicado "nl" é intrínseco, e precisa apenas do cursor da janela Dialog passando para uma linha nova para ser satisfeito.

5=1

Finalmente, Prolog encontra um bloqueio. Com sucesso tem conseguido satisfazer cada uma das cláusulas do lado direito da regra "do_it". Precisa satisfazer esta regra para satisfazer a meta. Mas quando encontra a expressão

Result = 3 – 2

a situação não é a mesma de quando somou 3 e 2 anteriormente. Em lugar de uma igualdade com uma variável não ligada e diversas constantes, esta é uma igualdade com uma variável ligada e diversas constantes. Portanto, em vez de avaliar o lado direito da igualdade, ligando a variável do lado esquerdo à do valor, Turbo Prolog apenas verifica para ver se os dois lados são realmente iguais. Tem valores que permitem este tipo de verificação. Pelo fato dos dois lados não serem iguais, a cláusula falha, a regra falha e a meta falha. Turbo Prolog nunca chega ao predicado "write" final.

Como pode ver, uma variável não-instanciada pode ser utilizada como se você estivesse na linguagem de procedimento com declarações de atribuição. Isto é verdade tanto para igualdades como para desigualdades. Os valores instanciados e constantes serão ligados à variável livre. A isto denomina-se *compartilhar* o valor. Em uma cláusula que não tem uma variável livre, porém, Prolog vai avaliar matematicamente a expressão para depois tratar com sua verdade ou falsidade.

A Figura 7-5 mostra como este programa poderia ser reescrito para funcionar.

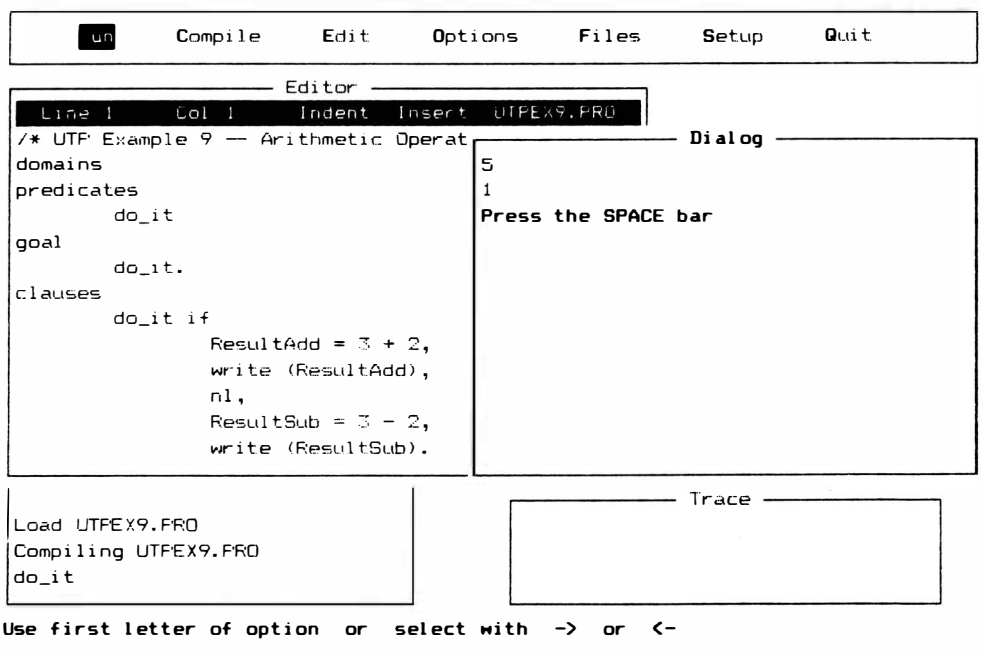


Figura 7-5 Programa reescrito mostrando como Turbo Prolog vai funcionar

MULTIPLICAÇÃO E DIVISÃO

A multiplicação e a divisão são executadas com os símbolos *infix*-padrão * e /. Dois outros operadores de divisão, "mod" e "div" podem ser escritos na forma *infix*. O operador "mod" devolve o resto de uma divisão; "div" devolve o quociente. Não os coloque em maiúsculas, pois serão transformados em variáveis em lugar de operadores intrínsecos. Caso "AnswerA" e "AnswerB" forem variáveis não ligadas

```
AnswerA = 9 div 4
```

instancia "AnswerA" em 2, e

```
AnswerB = 9 mod 4
```

instancia "AnsswerB" em 1.

PREDICADOS MATEMÁTICOS

A Tabela 7-4 relaciona predicados matemáticos que você pode utilizar em Turbo Prolog. Cada um deles utiliza o formato Prolog padrão

```
relationship (object)          [relação (objeto)]
```

e será executado como as funções aritméticas anteriormente descritas. Expressões com variáveis livres tentarão instanciar aquelas variáveis, e expressões sem variáveis livres serão testadas quanto ao seu sucesso ou falha.

As operações logarítmicas e trigonométricas são explicadas na Tabela 7-4. Operam com um único argumento e devolvem o valor calculado adequado daquele argumento.

As operações com bits operam com três argumentos. (Operações com bits são originalmente utilizadas para programar portas e outros aspectos de hardware de seu PC. Para maiores informações sobre operações com bits, veja um livro como: *Introdução aos microcomputadores*, Adam Osborne, 2ª ed., volume 1, Berkeley, CA: Osborne/McGraw-Hill, 1980). As operações bitand, bitor, bitnot e bitxor vão tomar os valores inteiros ligados dos dois primeiros argumentos, traduzi-los em números binários de 16 bits, executar a operação binária apropriada e ligar o terceiro argumento (caso seja uma variável livre) àquele valor. As outras operações com bits, bitleft e bitright traduzem o valor inteiro liga-

do do primeiro argumento em um valor binário de 16 bits, deslocam aquele valor binário da quantidade de posições dada pelo valor do segundo argumento, e depois ligam o valor final, deslocado, ao terceiro argumento.

Tabela 7-4 Predicados matemáticos padrões

Miscelânea

Abs (*x*)	Liga *x* ao valor absoluto do *x* original.
sqrt (*x*)	Liga *x* à raiz quadrada do *x* original.
random (*x*)	Liga *x* a um número real "aleatório", de modo que $0 <= x < 1$.

Logarítmico

log (*x*)	Liga *x* ao logaritmo de base 10 do *x* original.
ln(*x*)	Liga *x* ao logaritmo de base *l* do *x* original.
exp (*x*)	Liga *x* ao valor de *e* elevado à potência do *x* original.

Trigonométricas (é necessário que x seja um valor de ângulos em radianos)

cos (*x*)	Liga *x* ao cosseno do *x* original.
sen(*x*)	Liga *x* ao seno do *x* original.
tan (*x*)	Liga *x* à tangente do *x* original.
arctan (*x*)	Liga *x* ao arcotangente do *x* original.

Operações com bits

Se *x* e *y* são ligados a valores inteiros quando um destes predicados é chamado, o predicado vai traduzir aqueles valores inteiros em valores binários de 16 bits com sinal, executar a operação dada com estes valores, traduzir o resultado em um inteiro e ligar *z* àquele inteiro. O predicado bitnot executa da mesma forma mas precisa trabalhar com apenas um único valor de entrada: *x*.

bitand (*x, y, z*)	Liga *z* ao resultado lógico da operação AND entre *x* e *y*.
bitor (*x, y, z*)	Liga *z* ao resultado lógico da operação OR entre *x* e *y*.
bitxor (*x, y, z*)	Liga *z* ao resultado lógico da operação XOR entre *x* e *y*.
bitnot (*x, z*)	Liga *z* ao resultado lógico da operação NOT sobre *x*.

Deslocamentos de bits

Se *x* e *n* são ligados a valores inteiros, a operação deslocamento de bits vai traduzir o valor de *x* em um valor binário de 16 bits, com sinal, deslocar o valor *n* espaços na direção dada, traduzir o resultado em um valor inteiro, e ligar *y* àquele inteiro.

bitleft (*x, n, y*)	Liga *y* ao resultado do deslocamento de *x*, à esquerda, *n* posições.
bitright (*x, n, y*)	Liga *y* ao resultado do deslocamento de *x*, à direita, *n* posições.

Mesmo em operações com bits, *a matemática não é a mesma que em linguagens procedurais.* As expressões não são apenas avaliadas. Caso haja uma variável livre, a expressão será utilizada para instanciá-la. Caso não haja uma variável livre a expressão será avaliada para ver se os argumentos coincidem – se têm sentido matemático juntos. Se tiverem, o predicado será bem-sucedido. Se não, o predicado falhará.

Enquanto você estiver experimentando, lembre-se de que pode optar por deixar qualquer um dos três argumentos com uma variável livre. Você não precisa seguir em frente. Poderá operar para trás (retrocesso) para ver qual teria sido o valor deslocado ou qual teria sido o valor do primeiro argumento antes da operação AND.

RESUMO DE ARITMÉTICA

Faça alguns cálculos. Caso ocorram problemas, acompanhe (trace) o que está fazendo. Provavelmente você deve ter começado a pensar de forma procedural em lugar da forma "Prológica". Entretanto, existe outra fonte comum de erro: a matemática de computador algumas vezes é menos precisa do que você pensa. Turbo Prolog pode calcular um valor difícil até a precisão disponível do computador, e mesmo assim ainda obter 0.999999999 quando o esperado era 1.000000000. Algumas vezes você tem que tomar cuidado com resultados quase certos mas não exatamente corretos.

RECORRÊNCIA

Recorrência é um conceito importante em computação. Em Prolog, poderia facilmente ser discutido no Capítulo 5 (porque a recorrência remete o controle de procura da meta de volta a ele mesmo) ou neste capítulo (porque as operações fatoriais são o primeiro tipo de recorrência que as pessoas normalmente aprendem na escola). Entretanto, é muito útil com listas, de modo que você deve examiná-lo em detalhes naquele capítulo. Neste capítulo dará uma olhada rápida no funcionamento da recorrência em Prolog e como você pode determinar uma fatorial com ele. Você vai examinar o exemplo de programa EXAMPL10.PRO no disco Biblioteca/Exemplo. A *recorrência* é utilizada para descrever operações – não necessariamente matemáticas – que se chamam a si próprias como parte do processo. A operação fatorial é recursiva e é simbolizada na matemática geral por um ponto de exclamação. O fatorial se parece com o seguinte:

Y!

Esta expressão é avaliada como

$$Y! = Y*Y-1*Y-2*Y-3(...)*1$$

A indicação (...) significa que pode haver diversos números que você não quer escrever. A fatorial de um número é igual ao número vezes ele mesmo e todos números inteiros menores do que ele até o número 1. No entanto, não repita nenhum inteiro na série. A série Y–1, Y–2, e assim por diante, está ali apenas para mostrar como funcionaria com números grandes.

Para descrever a operação fatorial de forma recursiva, você escreveria:

$$Y! = Y*(Y-1)!$$

Como pode você ver, para qualquer número maior do que 5 ou 6 a definição recursiva de uma fatorial é muito mais curta e fácil de ser escrita. Esta definição diz que "a fatorial de Y é igual ao Y vezes a fatorial de Y menos 1". Mas como é que você sabe o valor de (Y–1)!? Você reutiliza – chamada recursiva – a definição.

$$(Y-1)!=(Y-1) * ((Y-1)-1)!$$

Uma expressão recursiva pode fugir ao controle se não tiver um fim bem determinado. A operação fatorial pára quando atinge o 1. Outras expressões recursivas têm de ter imposições de limite semelhantes. O computador utiliza uma pilha – uma criação lógica de sua própria memória – para guardar o valor e as variáveis mencionadas na procura do ponto final da recorrência. Se a definição pede níveis de recorrência em excesso, a pilha pode passar do limite: os dados podem se perder e o valor de recorrência pode voltar errado. Você pode aumentar o tamanho da pilha indo até o submenu Miscellaneous Settings do menu Setup. (Isto você precisa conhecer apenas quando se tornar um programador avançado.)

Eventualmente, a expressão recursiva atinge seu limite. Poderá então encontrar um valor que vai retornar por todos os níveis de parênteses até o resultado final. Lembre-se: a operação fatorial não é mostrada na sintaxe Prolog. O parênteses não contém relacionamentos e objetos: isto é matemática normal. Em Prolog, uma definição do predicado de recorrência marca os pontos CALL e RETURN temporários ao longo do caminho e, depois de pegar um valor final como instância, volta apela *string* até o começo. Isto pode ser muito poderoso quando você opera com listas muito longas que precisam ser inspecionadas, divididas e juntadas novamente. Você pode também definir objetos recursivos.

Turbo Prolog não oferece a fatorial como um operador intrínseco. A Figura 7-6

mostra o Programa-Exemplo 10 (EXAMPL10.PRO) no disco Biblioteca/Exemplo. O programa mostra como construir uma definição recursiva da fatorial.

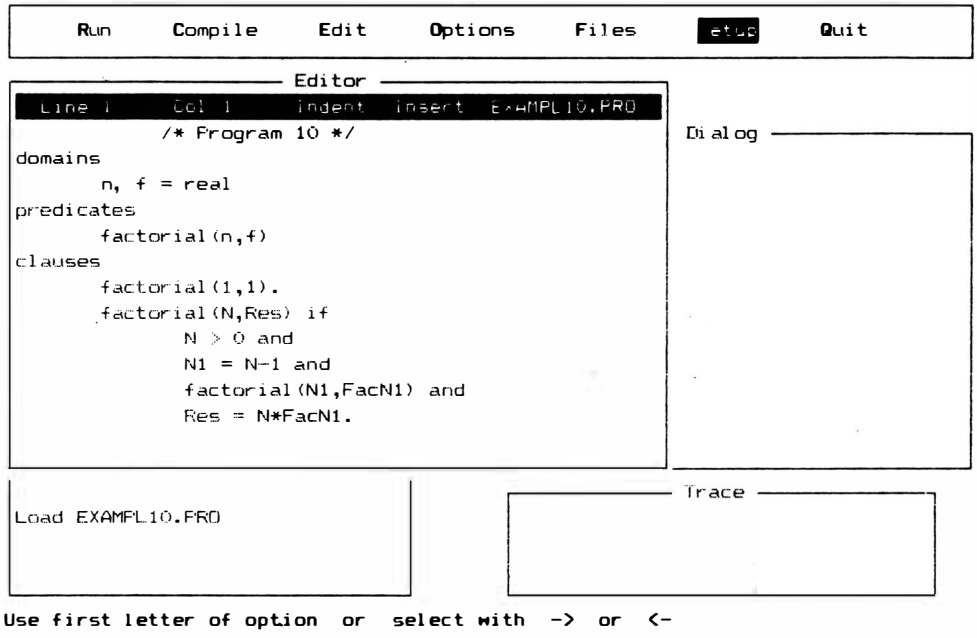


Figura 7-6 O programa EXAMPL10.PRO no disco Biblioteca/Exemplo

CAPÍTULO 8

ESTRUTURAS: FUNCTORES, LISTAS E CADEIAS DE CARACTERES

Na maioria das aplicações, você vai querer utilizar tipos de dados mais complexos que aqueles utilizados até aqui. A habilidade de construir estruturas de dados é fundamental para qualquer tipo de computação prática. Turbo Prolog não permite apenas argumentos mais complexos, mas tem também um número de recursos intrínsecos para a manipulação de listas de dados. Tem até predicados intrínsecos que operam em strings como se fossem listas.

Até aqui, neste livro, você cuidou apenas de objetos simples. Mesmo que um predicado tivesse uma aridade maior do que 1, cada um de seus argumentos foi simples, sem uma estrutura mais profunda. Mas na discussão do último capítulo da declaração de domínios, você aprendeu que outras estruturas, mais complexas, eram possíveis. Caso você seja um programador Prolog experiente, sabe a respeito de objetos e listas complexas. Entretanto, preste atenção à parte deste capítulo denominada "declarações de domínio". Caso não seja um programador Prolog experiente, gaste mais tempo neste capítulo do que gastou nos demais. Você não deve ter uma compreensão muito clara da manipulação de listas. Utilize o recurso de acompanhamento para seguir os programas.

FUNCTORES

Agora você já sabe que um relacionamento e seus objetos são freqüentemente escritos em Turbo Prolog como

```
relationship (object, object)          [relação (objeto, objeto)]
```

com o relacionamento precedendo os objetos, e os objetos entre parênteses. Sabe também que isto pode ser expresso como

```
predicate (argument, argument)          [predicado (argumento, argumento)]
```

Está e a única estrutura utilizada até aqui. Entretanto, nem sempre será suficiente. Mesmo que você não utilize listas (assunto discutido mais adiante neste capítulo), poderá simplificar a representação do conhecimento pela utilização de objetos mais complexos em suas cláusulas.

Por exemplo, uma lista de contas a serem pagaas poderia ser escrita como uma séries de fatos:

```
paid (phillip, food)                    pago (filipe, comida)
paid (phillip, phone)                   pago (filipe, telefone)
paid (phillip, rent)                    pago (filipe, aluguel)
paid (phillip, store)                   pago (filipe, loja)
```

Objetos compostos permitem que você acrescente maiores detalhes às cláusulas. O que segue é um exemplo utilizando a informação recém-apresentada:

```
paid (phillip, food (good_earth, 23.50))        pago (filipe, comida)
paid (phillip, phone (pacbell, 9999999, 55.55)) pago (filipe, telefone)
paid (phillip, rent (landlords, 1000))          pago (felipe, aluguel)
paid (phillip, store (macys, 250))              pago (filipe, loja)
```

Quando um argumento em si é um predicado, é denominado *functor*. Os argumentos de um functor são denominados *componentes*. Portanto, a estrutura da cláusula recém-mostrada com objetos compostos é:

```
predicate (argument, functor (component, component))
```

Não se deve fazer isto indefinidamente, porque um nível muito grande de parênteses poderia dificultar a leitura. Utilizados moderadamente, porém, objetos compostos podem tornar um programa mais fácil de ser organizado. Também, mesmo os componentes do objeto composto podem também ser objetos compostos.

DECLARAÇÕES DE PREDICADO E DOMÍNIO PARA OBJETOS COMPOSTOS

Um problema que você poderá ter com objetos compostos em Turbo Prolog é declarar corretamente os predicados e os domínios. A Figura 8-1 mostra um programa para os fatos de pagamentos de contas mostrados na última seção, juntamente com as declarações corretas.

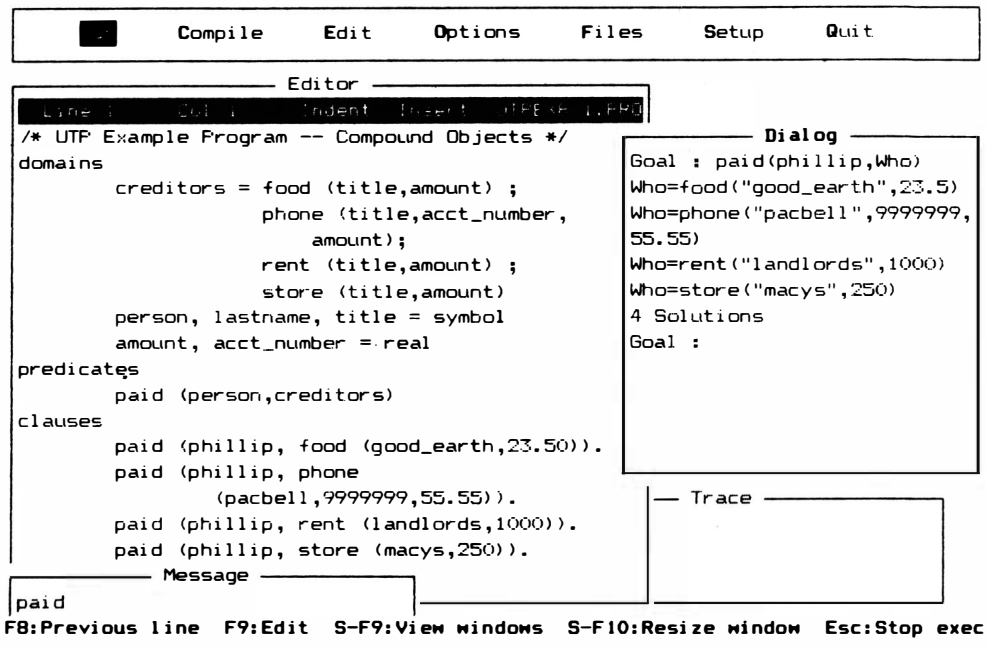


Figura 8-1 Programa para pagamento de contas com as declarações corretas

A Figura 8-2 mostra o que acontece se você estabelece algumas metas externas ao mesmo programa: devolve os functores inteiros com seus componentes. Na Figura 8-1, a utilização da variável na segunda posição originou uma lista de todas as contas pagas (juntamente com seus detalhes). Na Figura 8-2, a utilização de uma variável, na primeira posição, acompanhada por uma constante ("phone") na segunda posição, originou como resposta quem pagou o telefone. A segunda meta precisou das três variáveis vazias. Se não as tivesse, a meta functor "phone" não teria concordado com o functor cláusulas "phone" (que tinham três argumentos).

Tenha cuidado na escolha da linguagem dos functores e dos componentes, e poderá obter respostas úteis deste tipo de estrutura. Terá que declarar cada functor e os domínios para todos os componentes functores. Cada functor tem de ser único, mas o lado direito de uma declaração de domínio poderá ter diversas alternativas separadas por ponto-e-vírgula.

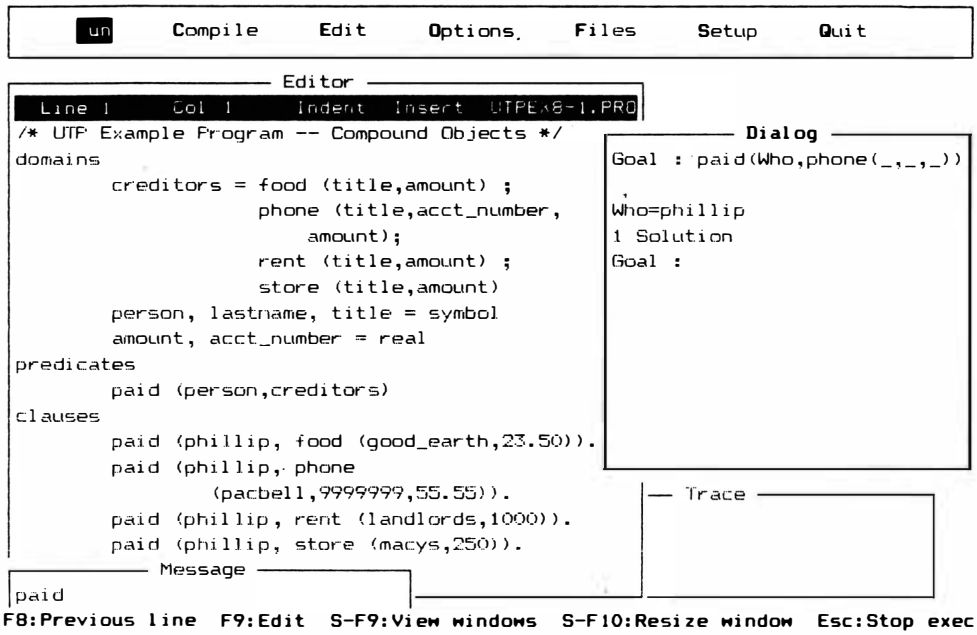


Figura 8-2 Metas externas estabelecidas para um programa de pagamento de contas

LISTAS

Uma lista é apenas outra forma de um objeto composto, mas é uma importante estrutura de dados. Parece uma coleção de termos – neste caso, *elementos* – separados por vírgulas e colocados no interior de colchetes. Aqui está uma lista de inteiros:

[1,2,3,5,8,13]

Aqui está outra lista de números:

[2,8,3.9E4,18,-200]

Esta é uma lista de símbolos:

[igneous,metamorphic,sedimentary] [ígneo, metamórfico, sedimentor]

O que segue é uma lista com os mesmos símbolos representados como string, cadeias de caracteres, (que será explicado na próxima seção):

```
["igneous","metamorphic","sedimentary"]
```

Finalmente, esta lista não contendo nada é denominada lista vazia:

```
[ ]
```

DECLARAÇÕES DE DOMÍNIO

Conforme mencionado no capítulo anterior, a declaração de domínio para listas não é muito diferente da declaração de outros tipos de objetos. A declaração, de maneira geral, tem a seguinte forma:

```
domains
        thingslist = things*
        things = objects
```

O asterisco indica que existem zero ou mais elementos na lista. A primeira linha da declaração significa que todo objeto da lista "thingslist" pertence ao tipo "things". Todos os objetos em uma lista têm de ser de um mesmo tipo, mas pelo fato de poder definir o tipo na complexidade desejada, esta restrição não será muito problemática. A segunda linha define "things" em termos das declarações de domínio-padrão. A sua declaração poderia ser mais simples se você tivesse utilizado um dos tipos padrões na primeira linha. Por exemplo,

```
domains
        numberslist = integer*
```

declara que todos os objetos em "numberslist" são inteiros. Você não precisa de uma segunda linha.

MANIPULAÇÃO DE LISTAS

Prolog opera com uma lista, dividindo-a em uma cabeça e uma cauda. A cabeça de uma lista é o primeiro membro à esquerda. A cauda é o resto da lista. A lista

```
[list]
```

tem uma divisão estrutural escrita com uma barra vertical entre a cabeça e a cauda, conforme mostrado aqui:

```
[head / tail]                    [cabeça / cauda]
```

Pelo fato da lista ser apenas outro tipo de objeto, tem de estar entre parênteses quando utilizada em uma cláusula, conforme mostrado aqui:

```
predicate ([list])
```

Aqui está outro exemplo:

```
predicate ([head / trail])
```

Novamente, pelo fato de uma lista ser um objeto, ela pode ser utilizada da forma que você utilizou outros objetos. Por exemplo, você poderia ter muitos objetos para um único predicado:

```
predicate ([list1], [list2], [list3])
```

CONCORDÂNCIA CABEÇA E CAUDA

O único processo que você precisa conhecer agora é como Prolog encontra a cabeça e a cauda. Depois disto, você pode utilizar as listas na mesma situação em que utilizaria outros objetos. Os processos de retrocesso, instanciamento, e assim por diante, funcionam da mesma forma com listas como com outros objetos.

Dê uma olhada no programa da Figura 8-3. É um programa curto que utiliza uma lista. (Observe que os membros da lista podem ser continuados em linhas consecutivas se houver vírgulas. Isto torna o programa mais fácil de ser lido em um determinado tamanho de janela.) Caso utilize a meta indicada na Figura 8-4

```
greek_letters ([Head : Tail])
```

onde as palavras "Head" e "Tail" são utilizadas como variáveis, Prolog lhe dirá quais são a cabeça e cauda da lista. Você poderia ter utilizado qualquer outro nome de variável, como X e Y.

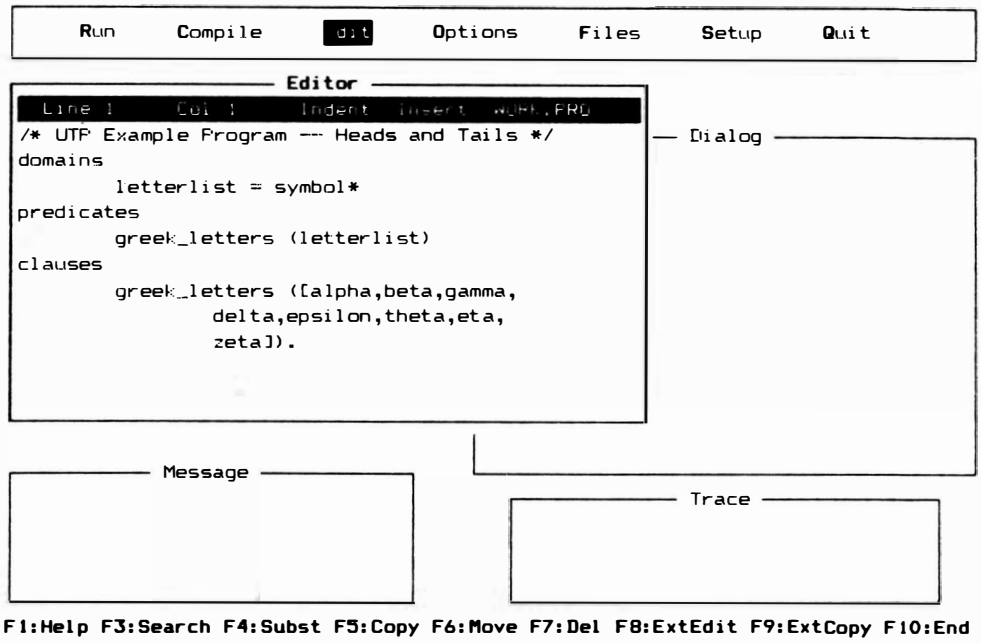


Figura 8-3 Um programa curto, utilizando uma lista

Caso você utilize a janela Trace, pode ver o que aconteceu na medida em que esta meta foi procurada. Acrescente a palavra "trace" acima da palavra "domains" no programa. Depois, faça uma nova compilação, processe o programa, estabeleça a meta novamente e utilize a tecla F10 para percorrer a ação. A Figura 8-5 mostra uma grande janela Trace contendo informações a respeito dos passos que verá.

Existem apenas três passos. O sistema vai procurar uma cláusula que concorde com a meta. Neste caso, o predicado tem de concordar, o número de argumentos tem de ser o mesmo, e os próprios argumentos têm de concordar diretamente, ou ser colocados em condição de concordar por intermédio de instâncias legítimas. A única claúsula combina com o predicado "greek_letters". Este predicado também tem apenas um argumento, e portanto combina com a segunda exigência. Finalmente, a lista em si da cláusula pode ser instanciada às variáveis não-ligadas da meta, para termos uma concordância completa. O sistema chama a cláusula, devolve a cláusula com os valores instanciados, e depois divide a lista em uma cabeça e uma cauda.

Retire a palavra "trace" do programa, recompile e processe, e utilize a meta:

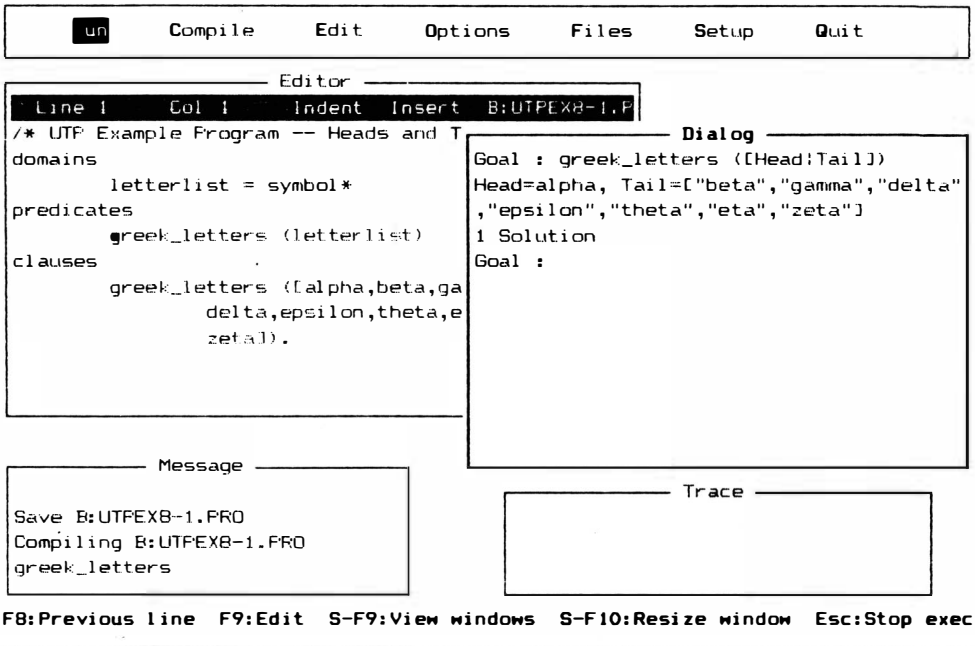


Figura 8-4 Utilizando uma meta com cabeça e cauda

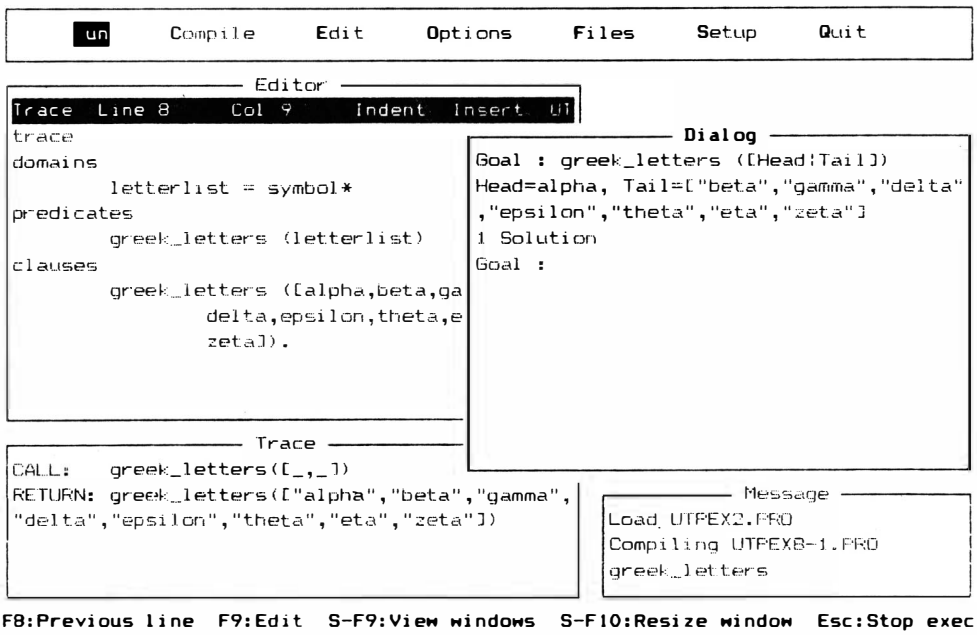


Figura 8-5 Acompanhando o programa

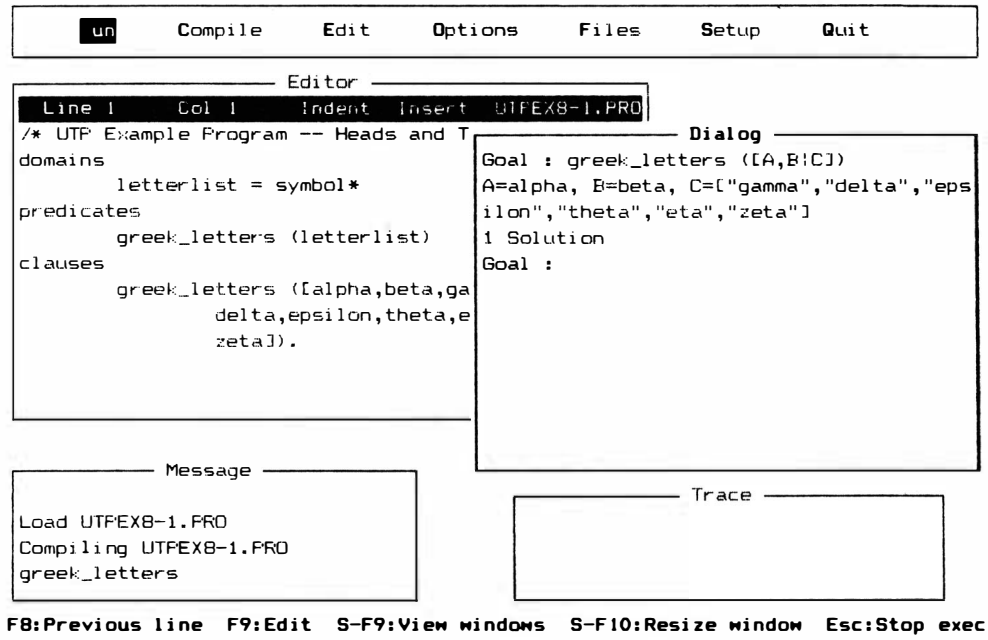


Figura 8-6 Meta dividida em três partes

```
greek_letters ([A,B/C])
```

A Figura 8-6 mostra o resultado. Esta meta divide a lista em três partes: A e B são os dois primeiros elementos da lista, e C é a cauda (o restante da lista). Novamente, a cauda é em si uma lista, mesmo quando não há nenhum membro sobrando e é uma lista vazia. Tente a meta

```
greek_letters ([A,B,C,D,E,F,G,H:I])
```

para demonstrar este ponto (veja a Figura 8-7). Esta divisão leva mais tempo que qualquer programa que você utilizou até aqui. (Caso queira saber por que, coloque o comando Trace de volta e veja todos os passos que Prolog precisa dar para obter o resultado.) "I" é comparado com a cauda e é mostrado como uma lista vazia:

```
I = [ ]
```

Caso a lista original esteja vazia, tanto a cabeça como a cauda serão indefinidos. A Figura

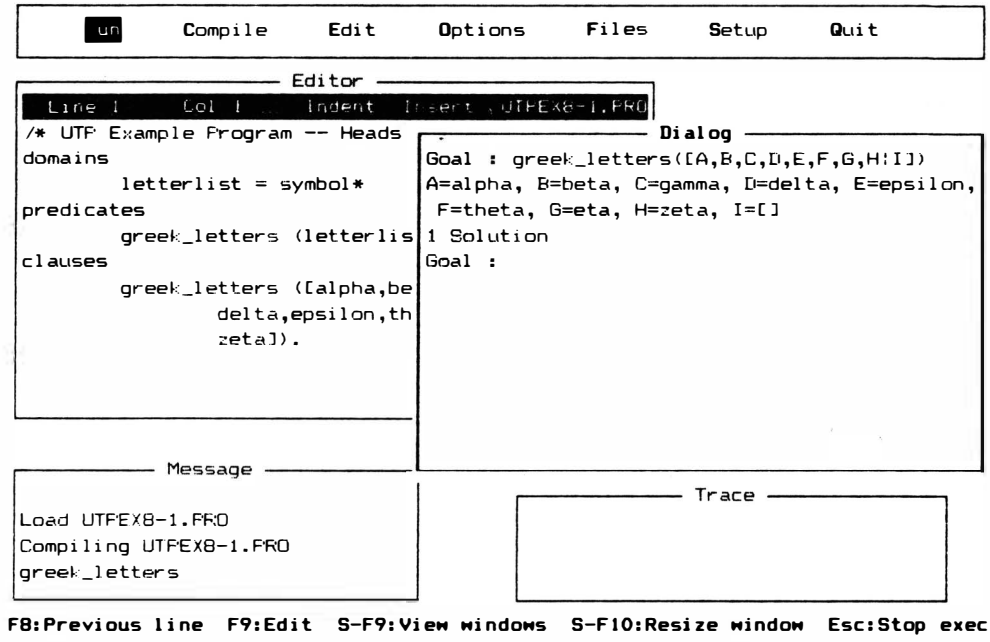


Figura 8-7 Comparando a letra "I" com a cauda

8-8 mostra como isto fica na tela. A Figura 8-9 mostra o que acontece se a cabeça da lista meta não coincide com a cabeça da lista cláusula. Não há solução para a meta; como ocorre com qualquer outra meta, os argumentos têm de se combinar para satisfazer a meta.

LISTA DE LISTAS

Você pode dar às estruturas a complexidade que desejar em Prolog. Conforme anteriormente demonstrado neste capítulo, você pode colocar objetos compostos em objetos compostos. Pode também colocar listas em listas. A lista

animals ([mammals, reptiles, birds, fish]) [animais([mamíferos, répteis, aves, peixes])]

poderia ter uma profundidade maior. Cada elemento poderia ser uma lista:

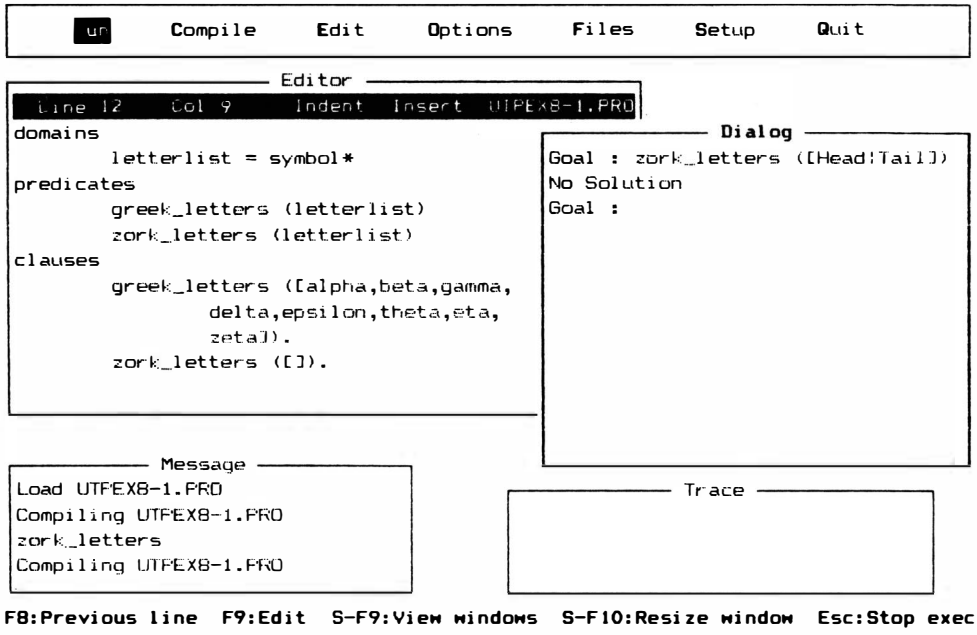


Figura 8-8 Lista vazia com cabeça e cauda indefinidas

animals([[monkey, buffalo, rat], [snake, iguana, crocodile], [seagull, penguin], [bass, salmon]])

[animais([[macaco, bufalo, rato], [cobra, iguana, jacaré], [garça, pingüim], [pescada, salmão]])]

Também, cada elemento poderia ter um functor e uma lista como o objeto do functor, como no seguinte:

animals (mammals ([monkey, buffalo, rat]), reptiles ([snake, iguana, crocodile]), birds ([seagull, penguin]), fish ([bass, salmon]))

[animais ([[macaco, bufalo, rato], [cobra, iguana, jacaré], [garça, pingüim],[pescada, salmão]])

RECORRÊNCIA

Muitas das manipulações que você gostaria de executar em listas são

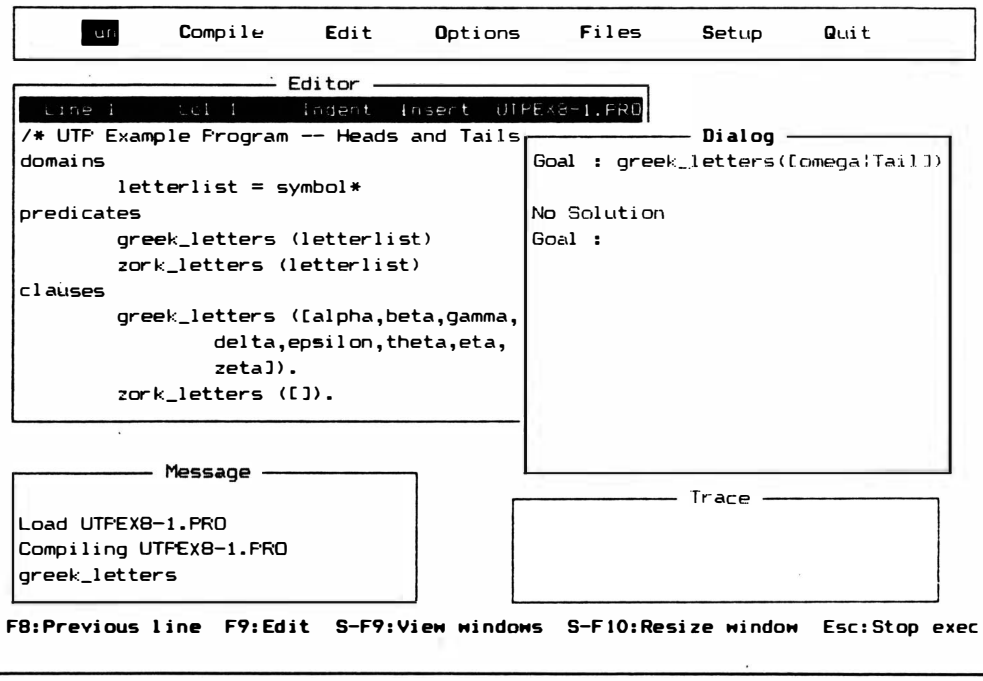


Figura 8-9 Cabeça da lista meta não coincidindo com cabeça da lista cláusula

facilmente escritas como operações recursivas – operações que se chamam a si próprios. Um exemplo simples é verificar se um determinado objeto é um elemento de uma lista. Vamos supor que você tenha um banco de dados geográfico que inclua uma lista destas cidades de Ohio e Califórnia:

cleveland
cincinnati
columbus
dayton
akron
zanesville
cupertino
santa_cruz
yreka
gualala
fresno

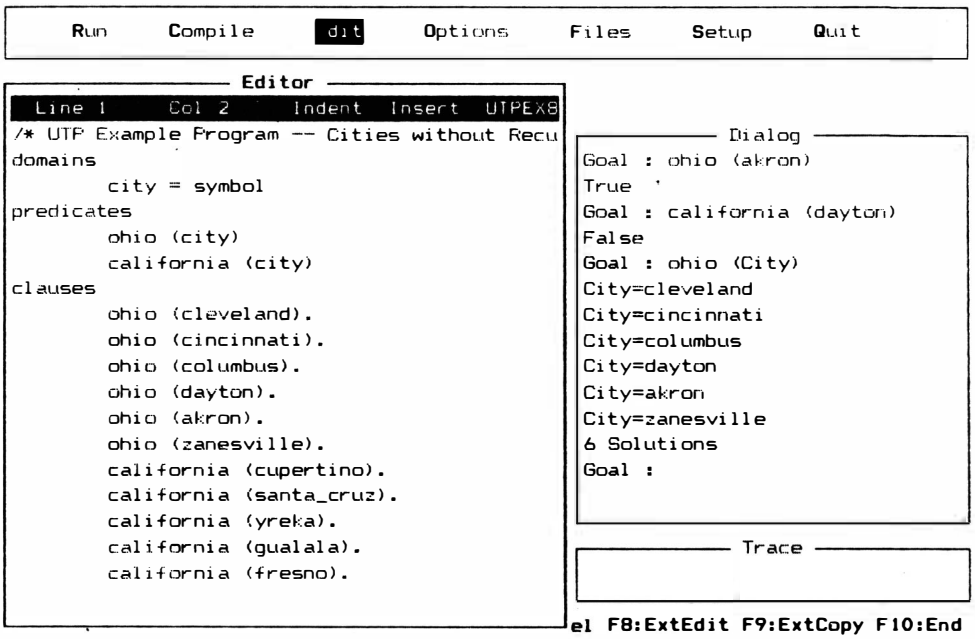


Figura 8-10 Um programa mostrando fatos a respeito de cidades

Para ver se uma determinada cidade está em Ohio ou na Califórnia, você poderia começar relacionando fatos a respeito destas cidades. Veja na Figura 8-10 o programa que faz exatamente isto. O programa poderia ser solicitado com uma meta como

```
ohio (akron)
```

ou a meta

```
california (dayton)
```

Definições de membros recursivos e listas farão a mesma coisa em menos espaço. A utilização de listas permite também que você acrescente mais cidades e estados aos fatos, sem ter de fazer uma longa e cansativa lista de todos os predicados a cada vez.

O programa da Figura 8-11 mostra como as listas podem substituir o predicado no programa anterior. Este programa tem o mesmo tamanho que o programa anterior, mas serve para um caso mais geral e seria, portanto, mais curto se mais fatos a respeito de cidades fossem acrescentados.

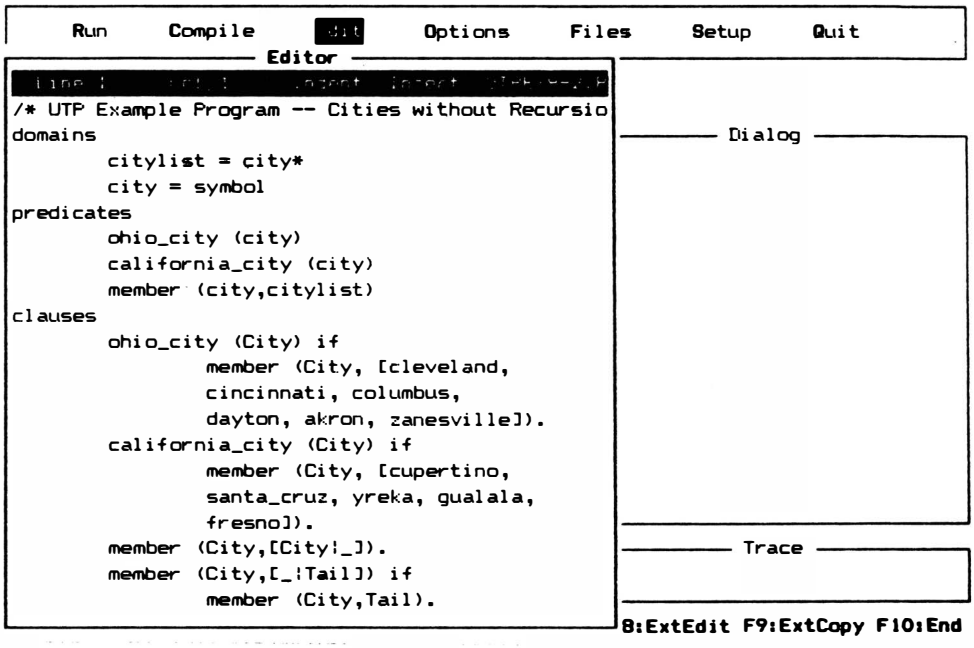


Figura 8-11 Listas substituindo predicados

As duas primeiras cláusulas são realmente bem diretas. Definem uma cidade como pertencente a um estado se for parte da lista de cidades daquele estado. As duas cláusulas seguintes são o núcleo recursivo da operação. Estas duas cláusulas informam ao Turbo Prolog como determinar se uma cidade é um elemento de uma lista de cidades pela divisão daquela lista, passo a passo. Tente esta meta:

```
ohio_city (zanesville)
```

Você vai obter a resposta "True". Utilize o comando Trace no programa para acompanhá-lo e ver como o resultado foi obtido. A Figura 8-12 mostra o efeito na janela Trace. O primeiro passo é chamar pela meta, como no seguinte:

```
CALL: ohio_city ("zanesville")
```

Isto leva imediatamente à regra "ohio" na seção de cláusulas. Para atender a esta regra, Prolog se movimenta para o lado direito e passa a se ocupar da submeta:

```
CALL.: member ("zanesville", ["cleveland", "columbus", "cincinnati", "dayton", "a-
kron", "zanesville"])
```

Para satisfazer esta meta – para saber se "zanesville" é um elemento da lista – Prolog precisa descer até a cláusula "member".

A primeira cláusula "member" declara que uma cidade é membro de uma lista se aquela cidade é a cabeça da lista. Em nosso caso, não é. A cabeça da lista é "cleveland", e não pode ser combinada com "zanesville".

É preciso tentar a próxima cláusula "member". Ela declara que uma cidade é um membro da lista (ou é um elemento da lista) se esta cidade é um elemento da cauda da lista. Juntamente com a cláusula anterior, isto faz sentido. Uma lista é totalmente formada pela sua cabeça e cauda; portanto, qualquer elemento da lista estará ou na cabeça ou na cauda.

As variáveis vazias em cada cláusula significam que você não se importa com a outra parte da lista. Se um elemento é a cabeça da lista, você não se importa com o que está na cauda. Se um elemento estiver na cauda, você não se importará com o que seja a cabeça, só que neste caso já procurou uma identificação com a cabeça.

Turbo Prolog passou por diversos níveis de metas para chegar até este ponto, mas precisa passar por mais uma submeta. Para atender à submeta de "member", Turbo Prolog tem de satisfazer a cláusula final: a regra que diz que uma cidade é membro de uma lista de cidades se ela é um elemento da cauda da lista. Prolog determina se a cidade é um elemento da cauda tratando a cauda como uma nova lista (que, evidentemente, ela é) e dissecando-a, utilizando as duas cláusulas "member".

Este processo é a recorrência. Assim que você tecla F10 para pesquisar este programa, a janela Trace mostra listas cada vez menores. Cada vez que Turbo Prolog chega à cláusula final e descobre que precisa verificar se "zanesville" é membro da nova cauda, retrocede à cláusula do primeiro membro, retira uma cabeça daquela cláusula, compara "zanesville" com a cabeça e depois volta à cláusula do "member" final se a cabeça não combinou. Este procedimento aparece como uma série de mensagens CALL e REDO na janela Trace. Neste instante "zanesville" será a cabeça da lista e a meta "zanesville" concorda com ela. Pelo fato do Turbo Prolog ter passado por uma longa série de níveis de recorrência, tem de retroceder novamente por estas mesmas definições. Isto é, assim que descobre que "zanesville" combina com a cabeça da lista "[zanesville]", pode retroceder um nível e saber que "zanesville" é um membro da lista "[akron, zanesville]". Depois pode

saber que "zanesville", como membro da cauda em cada estágio, também é membro da lista formada pelas cauda e cabeça. Turbo Prolog retrocede por estas listas com declarações RETURN, como na seguinte:

```
[zanesville]
[akron, zanesville]
[dayton, akron, zanesville]
[columbus, dayton, akron, zanesville]
[cincinnati, columbus, dayton, akron, zanesville]
[cleveland, cincinnati, columbus, dayton, akron, zanesville]
```

Depois volta à cláusula "ohio", com a afirmativa de que "zanesville" está em "ohio". A resposta "True" aparece na janela Dialog, e o programa está pronto para uma nova meta. Utilize a janela Trace para seguir este tipo de recorrência até compreender como funciona. Em cada estágio, a pilha da memória mantém controle de quantos níveis de submetas e retornos foram feitos. Esta maneira de desvendar listas é vital para toda manipulação de listas. Por exemplo, escrever todos os elementos de uma lista poderia ser conseguido com as duas cláusulas de recorrência

```
write_list_elements ([ ]).
write_list_elements (Head!Tail]) if
    write (Head) and
    write (Tail).
```

A segunda cláusula escreveria a cabeça de toda lista e depois faria uma nova lista da cauda. A primeira cláusula seria então examinada para verificar se a nova lista está vazia. Em caso positivo, a escrita estaria terminada. (Este ponto final faz com que a rotina pare e proteja a memória contra uma pilha perdida.) Caso a nova lista não estivesse vazia, a segunda cláusula seria chamada novamente de forma recursiva para cortar uma nova cabeça da lista, escrever a cabeça, e fazer uma lista ainda mais nova da cauda-mais-curta.

ACRESCENTANDO

Você pode construir muitos predicados para finalidades diversas. Um que você vai encontrar em qualquer livro Prolog é um predicado de "concatenação". Você pode definir

```
append (firstlist, secondlist, newlist)
```

para significar que a primeira e a segunda lista estão combinadas nesta ordem para formar uma lista maior, a nova lista. A segunda lista é anexada ao final da primeira. Uma ma-

neira recursiva de fazer isto é utilizar as duas cláusulas seguintes:

```
append ([ ], B_List, B_List).
append ([Head:Tail_old], B_List, [Head:Tail_new]) if
    append (Tail_old, B_List, Tail_new).
```

A Figura 8-13 mostra esta duas cláusulas em ação. O mesmo predicado pode ser utilizado para juntar listas, como indicado pela primeira meta. Poderá ser utilizado também para dividir uma lista, como indicado pelas metas dois e três. Será possível descobrir uma solução particular ou mostrar todas as soluções para dividir a lista (como mostrado na Figura 8-14).

STRINGS COMO LISTAS

Turbo Prolog tem muitos predicados intrínsecos para tratar *strings*. Strings são seqüências de letras ou caracteres. Uma variável string em Turbo Prolog poderá estar li-

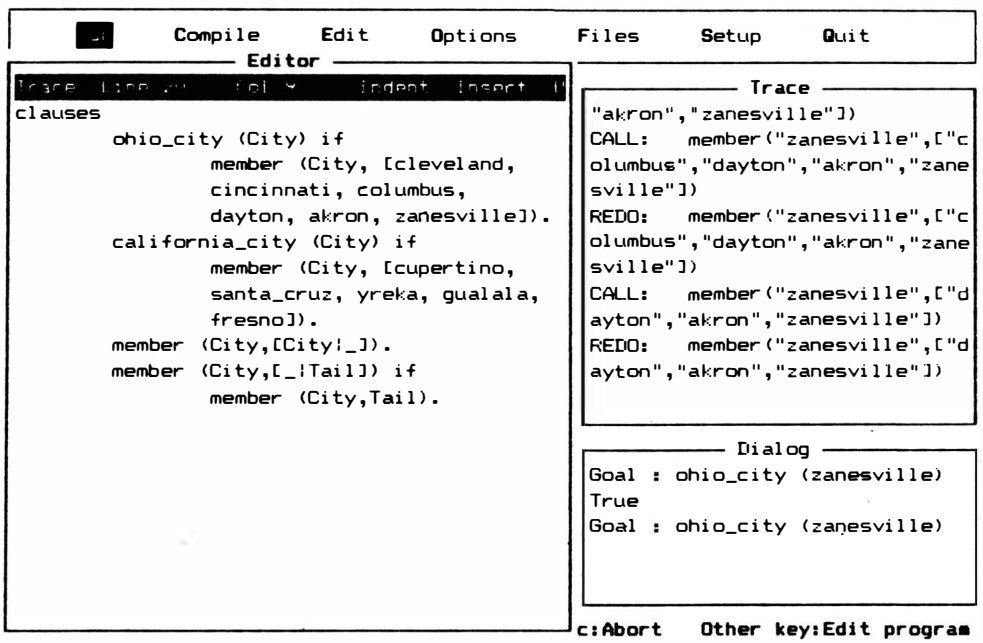


Figura 8-12 Efeitos mostrados em uma janela Trace

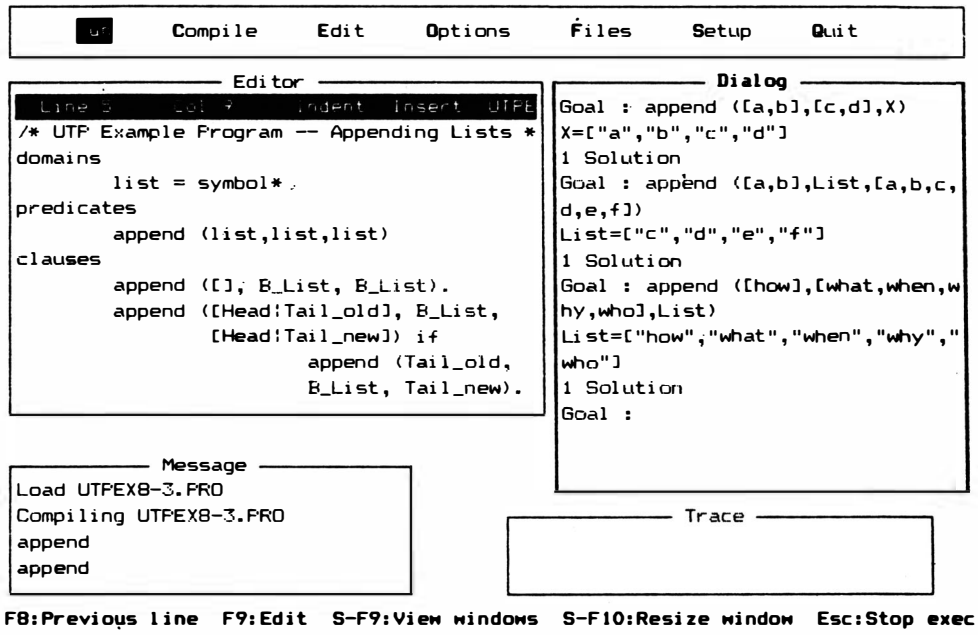


Figura 8-13 Cláusulas de recorrência

gada a uma string com um máximo de 64k caracteres. Uma constante string pode ser formada por um máximo de 250 caracteres. O tipo string é aquele utilizado com maior freqüência para representar palavras e sentenças. É, portanto, muito importante para o processamento de linguagem-natural, uma das forças do Turbo Prolog. Os dois predicados string fundamentais, "frontchar" e "frontstr", funcionam com os mesmos princípios anteriormente explicados para listas.

Predicados intrínsecos são relacionados alfabeticamente no "Guia de Referência" do *Manual de Usuário* do Turbo Prolog. Naquela seção, cada uma das listas inclui uma seleção de comentários entre parênteses como "(o,o,o)", "(i,o)" ou "(i)". O motivo disto é que a maioria dos predicados pode operar de diversas formas. O "i" significa "instanciado". O "o" representa "open" (aberto), como em não-ligado. Ambos se referem aos argumentos dos predicados.

Por exemplo, o predicado

isname (StringParam)

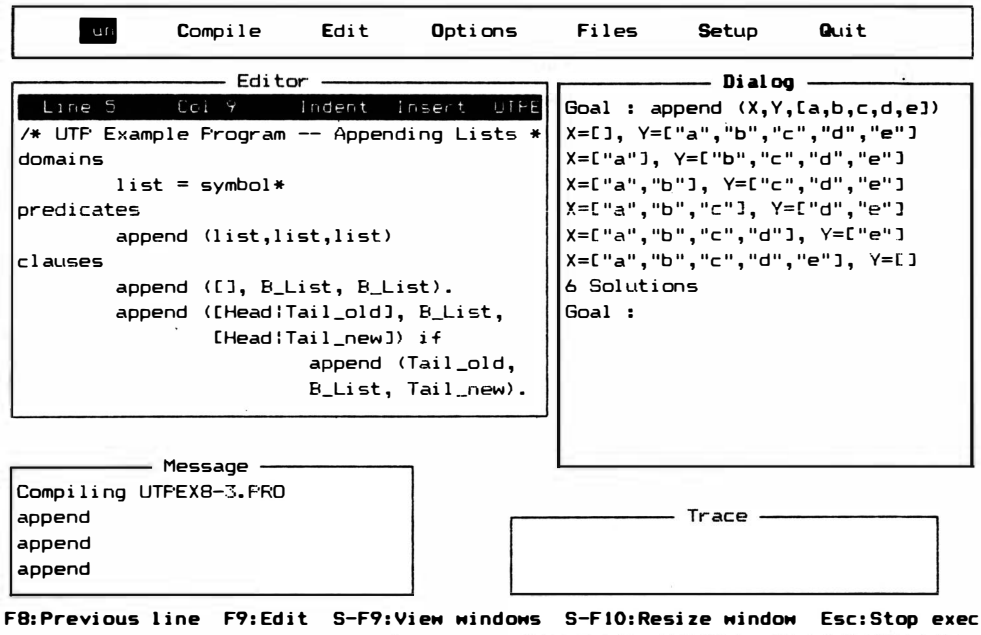


Figura 8-14 Soluções para o problema de dividir a lista

tem apenas a possibilidade "i". Significa que os argumentos têm de estar instanciados quando o predicado é atingido, para que o predicado seja bem-sucedido. Caso não esteja instanciado, Prolog não vai procurar alguma coisa a instanciar – simplesmente falhará o predicado.

Por outro lado, o predicado

right (Angle)

refere-se ao ângulo da tartaruga gráfica (discutido mais adiante no livro). Este predicado tem duas alternativas: "(i)" ou "(o)". "Angle" pode ser instanciado ou aberto. Caso esteja instanciado quando o predicado for atingido, a tartaruga será colocada neste ângulo. Caso "Angle" não esteja instanciado quando o predicado for atingido, o oposto ocorre – "Angle" será instanciado com o ângulo atual, corrente da tartaruga. Todos os predicados funcionam desta forma. Quando "(o)" estiver disponível, o predicado poderá atribuir valores da situação à variável para obter sucesso. Quando "(i)" estiver disponível, o argumento já deverá

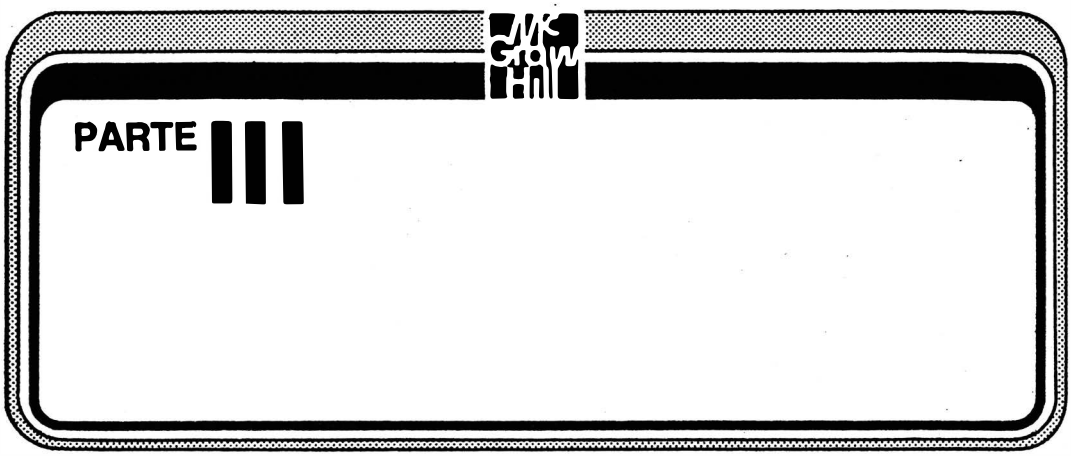
ter um valor que pode ser testado para se obter sucesso ou guardado como situação.

FRONTCHAR (STRING, FRONTCHAR, RESTSTRING)

Este predicado encontra o caractere inicial de uma string. Os três argumentos têm tipos de domínio string, char e string respectivamente.

FRONTSTR (NUMBEROFCHARS, STRING1, STARTSTR, STRING2)

Este é outro predicado do tipo "concatenação", exceto que este é especializado. Os quatro argumentos têm os tipos de domínio integer, string, string e string respectivamente. Caso você coloque um inteiro como "NumberOfChars" e coloque um "StartStr" (string inicial), o predicado devolve um String1 com esta quantidade de caracteres do começo de "StartStr" e um "String2" com o restante dos caracteres de "StartStr".



Até agora este livro discutiu os fundamentos de utilização do Turbo Prolog. Esta seção vai apresentar algumas das características únicas ao programa que lhe permitirão escrever programas mais sofisticados e poderosos. Estes tópicos avançados incluem janelas, gráficos, sons e características de compilação.

CAPÍTULO 9

ESCREVENDO, LENDO E JANELAS

Este capítulo é importante porque você provavelmente não quer fazer apenas pequenos programas e vê-los funcionando na janela Dialog. Apesar do coração do Prolog ser a máquina lógica (que não especifica as belezas de entrada e saída), você quer mais do que as respostas "True" e "False" para as metas propostas. Acrescente o que vai aprender neste capítulo às informações de manuseio de arquivos, sons e gráficos que descobrirá no capítulo seguinte, e poderá fazer uma interface sob medida.

Este e os capítulos seguintes, vai fazê-lo passar por um curso intensivo de predicados padrões. Depois dos predicados simples de leitura e escrita, você vai entrar rapidamente em um território específico do Turbo Prolog e não em outros Prolog. Até este ponto do livro, tudo que você aprendeu é comum a quase todos os outros Prolog. Apenas a sintaxe se mostrava diferente.

Entretanto, os predicados de E/S do Turbo Prolog e funções especiais (como janelas) simplesmente não são disponíveis ou são manuseados por predicados completamente diferentes em outras implementações do Prolog. Arquivos, gráficos, som e coisas semelhantes são diferentes em Turbo Prolog comparados com outros Prolog (como verá em capítulos posteriores). Isto é porque tais predicados são os elementos de procedimento do Prolog e não são os descendentes diretos da lógica pura. São os elementos menos padronizados no campo da programação lógica.

Os predicados são bem diretos, e você deverá ser capaz de fazer muito de seu trabalho com alguns predicados de leitura, escrita e janelas. Você poderá, mais tarde, retornar para aprender predicados mais especializados.

DUAS MANEIRAS DE OPERAR

Muitos dos predicados padronizados do Prolog funcionam de duas maneiras: tomam valores de argumento ligados e os colocam no sistema; ou tomam argumentos livres e os instanciam com os valores relevantes atuais no sistema. O "Guia de Referência" do *Manual do Proprietário* da Borland utiliza os especificadores "(o)" e "(i)" para indicar quando um predicado pode operar com variáveis abertas (ou previamente livres) ou instanciadas (ou previamente ligadas). O estado da variável para o argumento no momento em que o predicado é chamado é o que conta. As palavras "livre" e "ligado" são utilizadas nas tabelas em todo este capítulo. Os dois predicados padrões seguintes permitem que você determine se um predicado é livre ou ligado.

FREE (VARIÁVEL)

Este predicado será bem-sucedido se a variável à qual se refere não estiver instanciada com um valor.

BOUND (VARIÁVEL)

Este predicado será bem-sucedido se a variável à qual se refere estiver instanciada com algum valor.

FINDALL

Depois de especificar o nome do predicado e da variável na qual você está interessado, pode utilizar "findall" à medida que o programa for processado, para fazer uma lista dos valores aos quais a variável se liga com sucesso. Isto pode ser útil para pesquisa ou para coletar respostas corretas a uma exploração lógica determinada. O Programa-Exemplo 49 no disco Biblioteca/Exemplo utiliza "findall" para obter uma série de números para extrair a média.

E/S SIMPLES

Pelo fato do Turbo Prolog ter um sistema de janelas intrínseco (e seu Editor), não foi preciso que você fizesse programas capazes de executar sérias operações de E/S, como aquelas necessárias em outras linguagens de programação. As quatro janelas – Editor, Trace, Message e Dialog – estiveram ocupadas apresentando mensagens e resultados. Em alguns Prolog, o único gesto em direção à gentileza é mostrar uma indicação de prontidão: ponto-de-interrogação na tela. Com estes computadores pouco amigáveis, se você

estabelecer uma meta que ocorra ser verdadeira, tudo o que conseguir será novamente o ponto-de-interrogação. Com padrões deste tipo, Turbo Prolog é praticamente loquaz.

Entretanto, as janelas intrínsecas são destinadass principalmente a um sistema de desenvolvimento. Têm a intenção de levá-lo a um programa que funcione, não sendo necessariamente parte do programa. Não devem ser vistas como a única residência permanente de uma aplicação prática que você possa desenvolver. Por este motivo, você pode construir as suas próprias janelas para ler e escrever através delas.

ESCREVENDO

Prologs antiquados (pré-Turbo) ofereciam um comando de impressão denominado "pp", que significa "Pretty Print" (Bela Impressão). Este predicado ajudou o programador na colocação dos caracteres na posição correta da tela. Turbo Prolog não tem Bela Impressão, mas tem uma série de predicados de impressão, conforme indicado na Tabela 9-1.

WRITE

O primeiro predicado de escrita é o predicado de utilização geral "write" que você já conhece. O predicado "write" é bem-sucedido quando escreve os argumentos que o seguem na tela na posição atual do cursor. Pelo fato de "write" ser definido como

```
write (el, e2, e3,...eN)
```

você pode ligar a ele tantos argumentos quantos desejar. Com uma definição de fluxo de "(i)", você pode definir cada um e qualquer um dos argumentos como uma constante (de um tipo de domínio padrão) ou como uma variável instanciada. (Os padrões de fluxo estão contidos na seção "Guia de Referência" do *Manual do Proprietário*. Caso você tente utilizar uma variável não-instanciada, conseguirá apenas uma mensagem de erro. O predicado vai escrever as constantes e os valores das variáveis à janela atualmente ativa do "writedevice" corrente. Infelizmente, "write" não funciona como um predicado isolado. A Figura 9-1 mostra um exemplo do predicado "write" em seu estado mais primitivo. Este programa nem mesmo tem uma seção de cláusulas. (Lembre-se de que você não precisa declarar um predicado intrínseco.) O predicado "write" é apenas escrito como uma meta interna. Depois executa na janela Dialog. A Figura 9-2, mostra a mensagem de erro que se obtém se é utilizada uma variável não-instanciada. A Figura 9-3 mostra o que acontece se você utilizar uma variável instanciada.

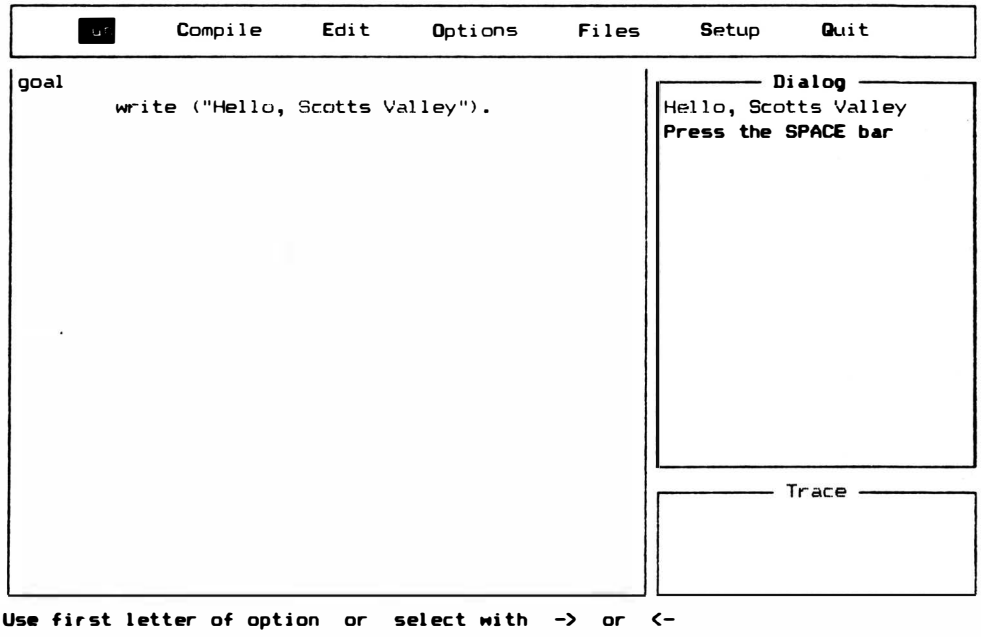


Figura 9-1 Exemplo de "predicado write"

Tabela 9-1 Predicados de escrita padrão

write (e1,e2,e3,...eN (I*)

Escreve qualquer número de constantes ou variáveis instanciadas na janela ativa do "writedevice" (dispositivo de escrita) ativo. O asterisco significa que pode trabalhar com um número opcional de argumentos.

nl

O predicado "nl" representa *newline* (novalinha). Faz o cursor se mover para linha seguinte da janela ou tela ativas. Qualquer escrita ou leitura feita depois de "nl" ocorrerá a partir de nova posição do cursor. Tecnicamente, escreve um retorno de carro e uma alimentação de linha ao "writedevice" ativo. Este predicado não tem nenhum argumento.

writedevice (SymbolicFileName) símbolo –(I) (O)

Permite especificar o "writedevice" ativo (atual). Pode trabalhar com qualquer um dos três tipos de "SymbolicFileName" – um dos arquivos simbólicos predefinidos (tela ou impressora), um nome de arquivo definido pelo usuário para um arquivo aberto ou uma variável não-ligada. Caso seja atualizada uma

variável não-ligada, o predicado vai ligar "SymbolicFileName" ao nome do "writedevice" atual. Os valores têm de estar dentro do domínio "symbol" padrão.

writef(FormatString, Arg1, Arg2, Arg3,...) (I, I*)

Escreve formatado com qualquer número de argumentos. Os argumentos podem ser constantes ou variáveis ligadas. "FormatString" permite que você informe ao Turbo Prolog o formato que deve ser utilizado para imprimir o argumento. O asterisco significa que ele pode operar com um número opcional de argumentos, da mesma forma que "write". Estes são os especificadores que você pode utilizar dentro do formato "%–m.p" para construir "FormatString":

– Alinha à esquerda. O padrão (a condição sem "–") é alinhamento pela direita.

m Especifica a largura mínima do campo.

p Especifica a precisão de um número em ponto flutuante ou o número máximo de caracteres que devem ser impressos de uma *string*. A este campo você pode acrescentar qualquer uma das seguintes três letras para condições especiais:

f Escreve números reais em notação decimal fixa (é também o padrão)

e Escreve números reais em notação exponencial

g Escreve utilizando o formato mais curto

display (String) **string–(I)**

Deve ter uma variável instanciada dentro do domínio de *string* padrão. Escreve o conteúdo de "String" à janela ativa na posição do cursor. Pelo fato de você depois poder inspecionar a *string* utilizando os comandos das teclas de controle da janela do Editor, este predicado também é considerado um dos predicados especiais que lhe permitem acessar o Editor a partir de um programa.

nl

Este predicado inerente – denominado *novalinha* – é semelhante a um predicado "write" sem argumentos. Tudo o que faz é mover o cursor para a próxima linha. Em termos técnicos, escreve um retorno de carro e uma alimentação de linha ao "writedevice" atual (o dispositivo que você definiu como sendo ativo para aceitar predicados "write"). O "writedevice" default é a tela. Pode ajudá-lo a melhorar a aparência de sua saída. A Figura 9-4 mostra o resultado quando "nl" é acrescentado ao programa mostrado na figura anterior.

Outra maneira de conseguir uma linha nova é escrever a *string* ("\n"). A barra invertida transforma isto em um caractere de controle. Outros caracteres de controle são apresenta-

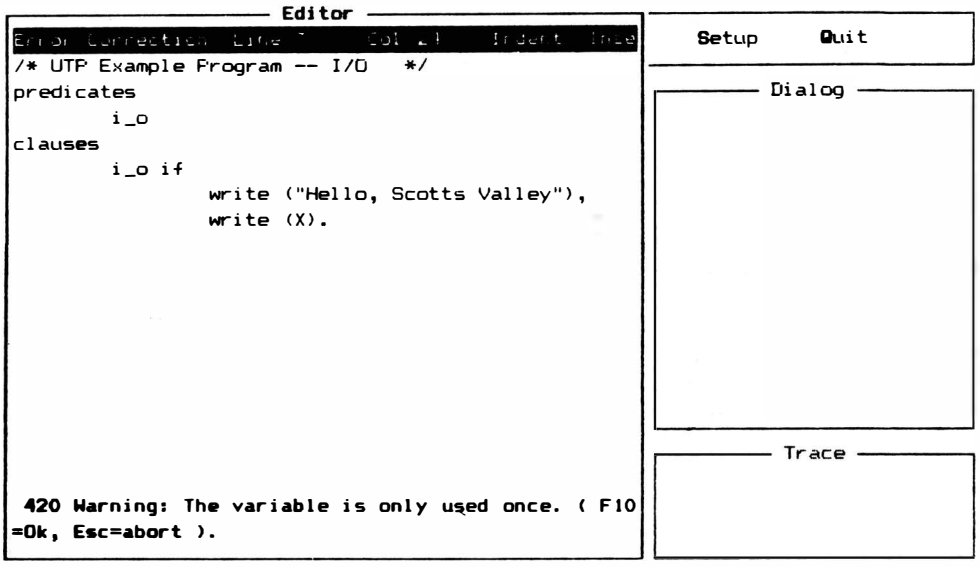


Figura 9-2 Mensagem de erro de uma variável não-instanciada

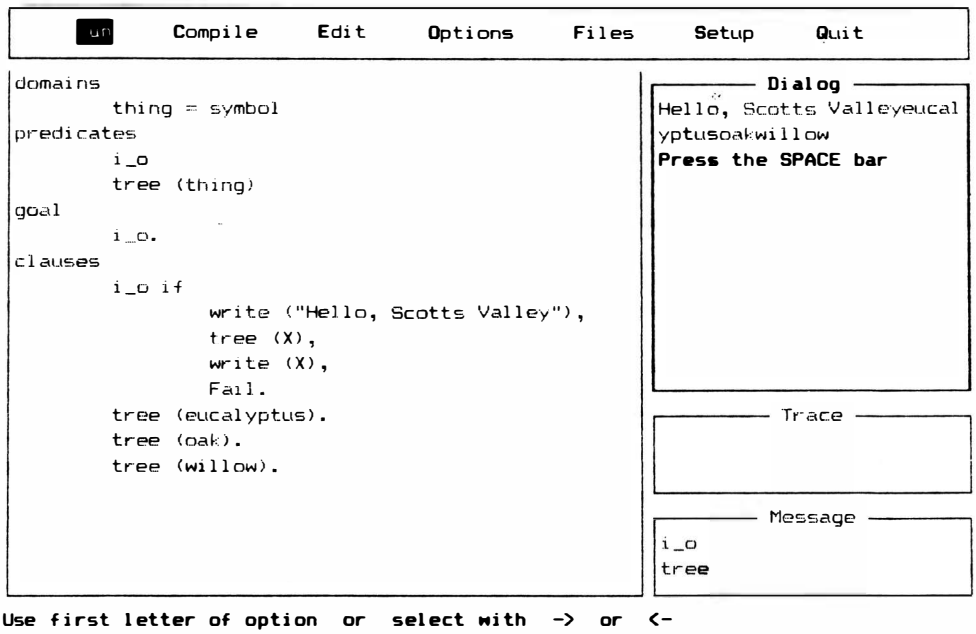


Figura 9-3 Exemplo da utilização de variável instanciada

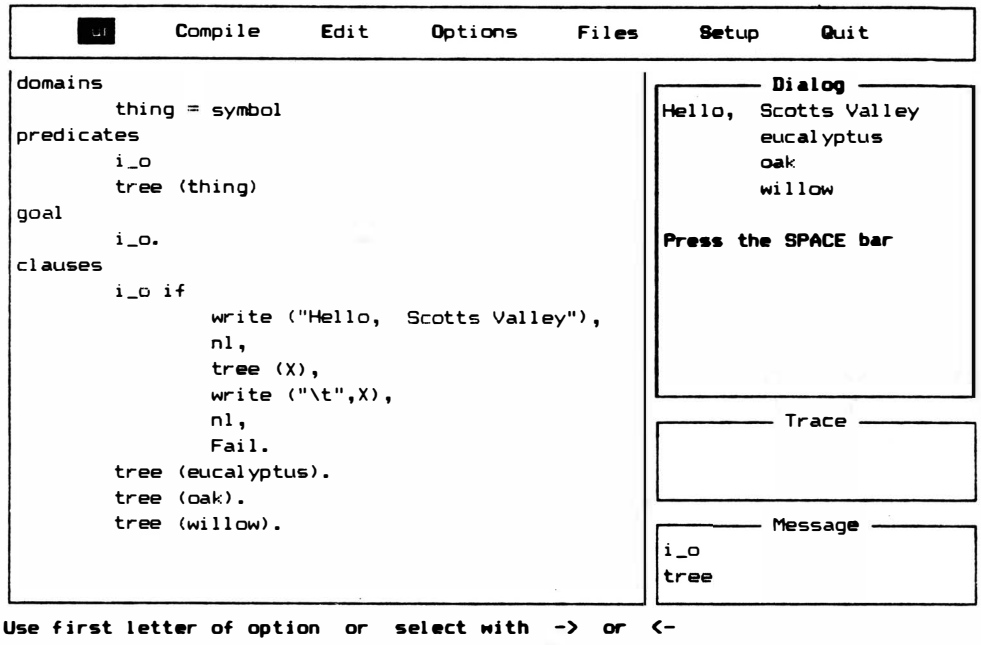


Figura 9-4 Acréscimo do "nl" ao programa

dos na Tabela 9-2. Caso você queira imprimir uma única barra invertida (por exemplo, na especificação de um percurso DOS), utilize duas barras invertidas consecutivas entre aspas. A Figura 9-4 mostra também os exemplos de escrita com utilização de caracteres de controle.

Tabela 9-2 Caracteres de controle para o Predicado* "write"

\	Isolada, indica que o próximo caractere é de controle.
\n	O caractere de controle para uma linha nova.
\t	O caractere de controle para uma tabulação.
\b	O caractere de controle para um retrocesso.
\\	A maneira de se escrever uma barra invertida na página.

* Outros caracteres de controle dependem de seu sistema. Por exemplo, se quer enviar um caractere de controle que faz sua impressora emitir um som, precisa saber que caracteres a sua impressora interpreta com um som. Depois, coloque aquele caractere após uma barra invertida em uma operação de escrita do Turbo Prolog.

WRITEDEVICE (SYMBOLICFILENAME)

O predicado "writedevice" permite que você defina para que dispositivo (tela, impressora ou outro) o predicado "write" vai enviar informação. Este predicado pode trabalhar com um arquivo simbólico predefinido (tela ou impressora) ou um arquivo que você designa e que está aberto. A Tabela 9-3 relaciona os dispositivos predefinidos para este predicado e para "readdevice". Arquivos abertos serão discutidos posteriormente neste livro. A Figura 9-5 mostra o que acontece tanto quando você altera o predicado "writedevice" como quando utiliza mais argumentos no predicado "write". Os dois últimos valores instanciados para "tree" foram enviados à impressora em vez de à tela. Você pode acompanhar este processo para ver os detalhes. A Figura 9-6 mostra outra utilização do predicado "writedevice". Caso o argumento, "SymbolicFileName", seja não-instanciado, o predicado vai instanciá-lo ao valor atual. Neste caso, o valor é "screen". Outra declaração "write" poderá informar-lhe qual é este valor.

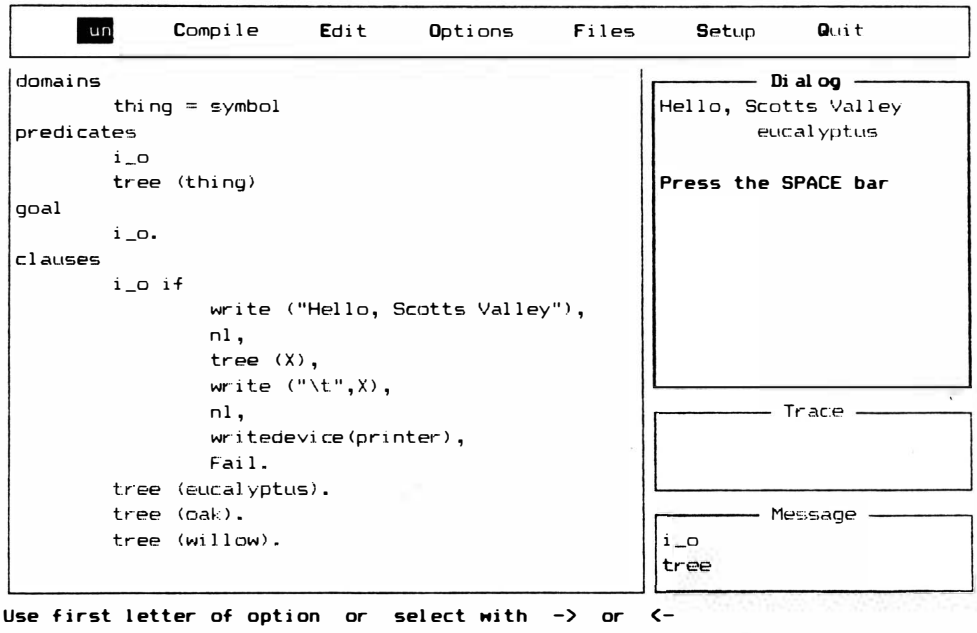


Figura 9-5 Alterando "writedevice" e utilizando mais argumentos em "write"

Tabela 9-3 Dispositivos padrões (predefinidos) para "writedevice" e "readdevice" como predicados

Dispositivo predefinido	Dispositivo de saída
printer	porta paralela
screen	monitor
keyboard	teclado
com1	porta serial

WRITEF

Caso você queira uma formatação especial para a escrita, poderá utilizar o predicado "writef". (Veja a Tabela 9-1 para ver como funciona.) "FormatString" especifica como imprimir os argumentos (dos quais poderá haver qualquer número). Os especificadores de formato

%–m.p

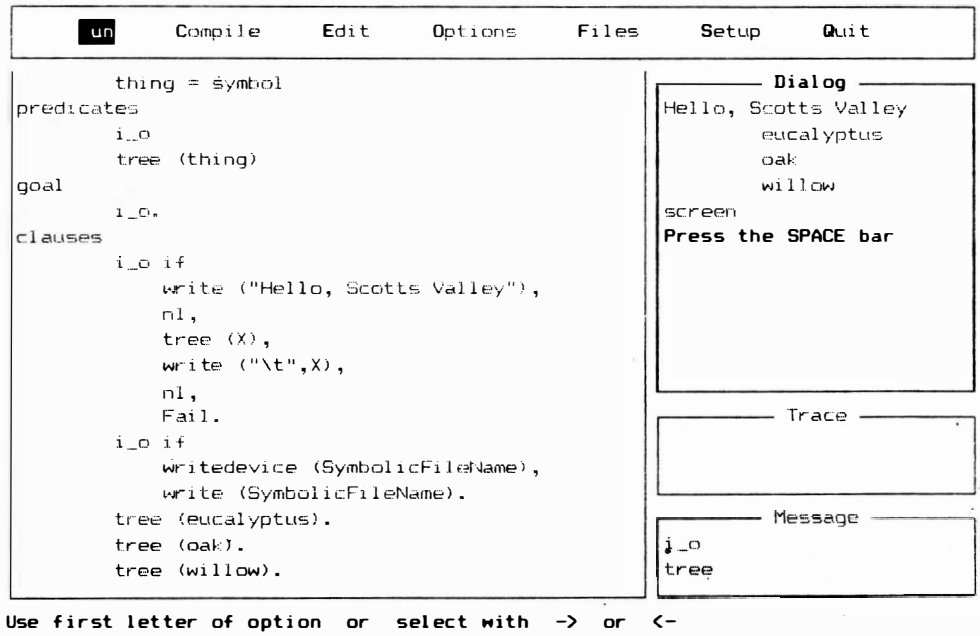


Figura 9-6 Outra utilização do predicado do "writedevice"

são utilizados para delinear quais as instruções especiais que se aplicam a cada argumento. A Figura 9-7 mostra o predicado "writef" em operação.

DISPLAY

Este predicado apresenta a *string* na janela ativa. Você pode utilizar os comandos de cursor que operam na janela Editor para fazer inspeção, mas não para modificar, a *string*. Veja a seção "Revisão da Janela Editor" no final deste capítulo para maiores informações sobre este predicado.

PREDICADOS DEFINIDOS PELO USUÁRIO

É possível também construir os seus próprios predicados para escrever coisas exatamente como você quer. Isto normalmente é verdade em Prolog, porque você pode utilizar uma regra para definir o que deveria logicamente ser verdadeiro em determinadas circunstâncias.

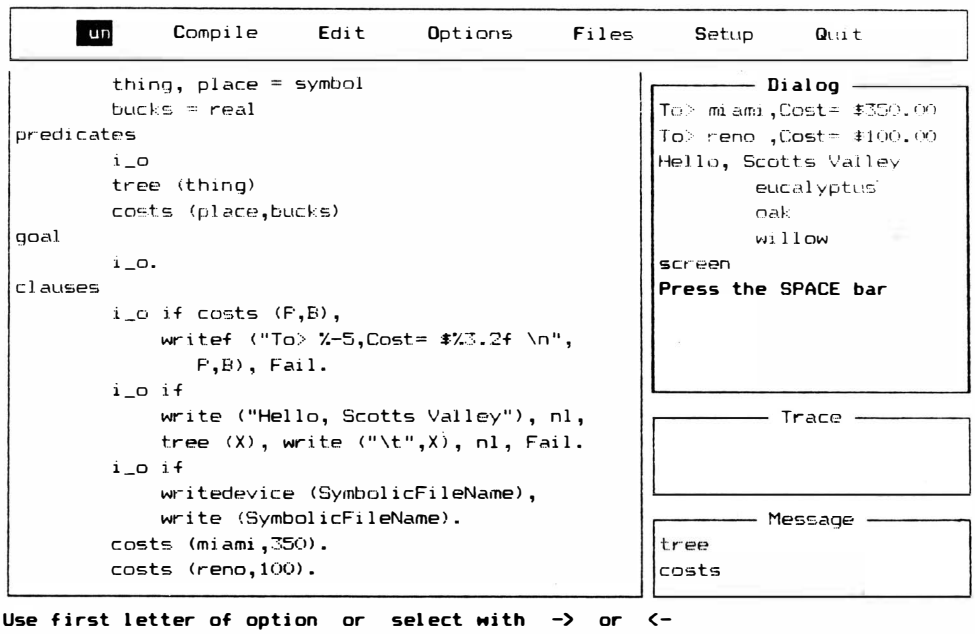


Figura 9-7 O predicado "writef"

Por exemplo, o Exemplo de Programa 20 no disco Biblioteca/Exemplo mostra os elementos de uma lista um por vez. A Figura 9-8 mostra este programa. Os predicados "writelist" e "write5" cuidam de tarefas especiais de escrita e são "construídos" a partir do predicado "write" e de outros predicados.

LENDO

Turbo Prolog também pode ler informação do teclado ou de um arquivo em disco. Pode ler qualquer coisa desde um único caractere até uma linha inteira de caracteres. Não pode ler diretamente objetos compostos ou listas. A Tabela 9-4 mostra os predicados padrões de leitura.

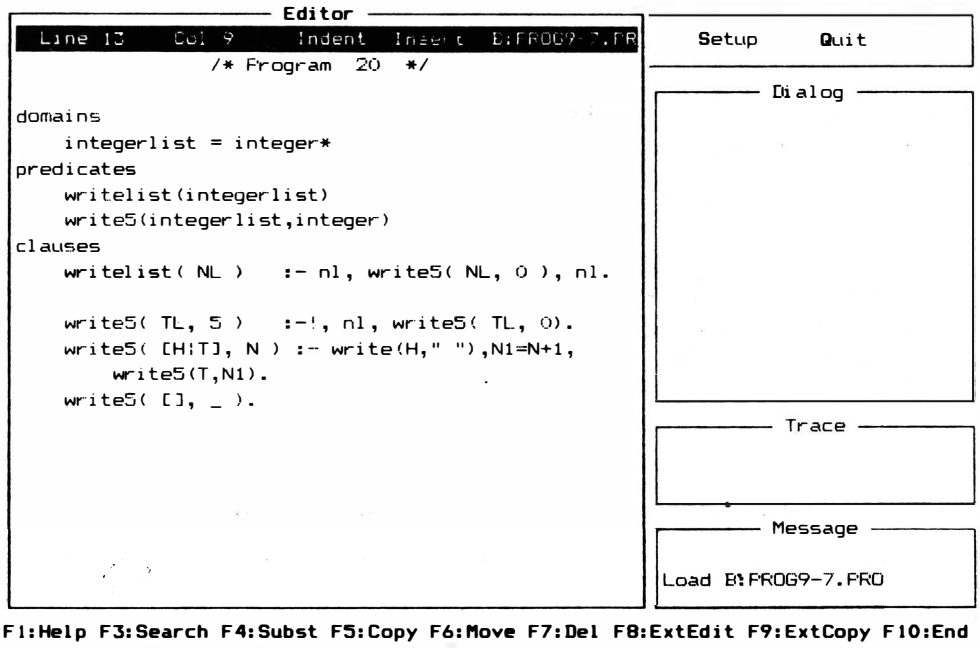


Figura 9-8 Programa-Exemplo 20 do disco Biblioteca/Exemplo

Tabela 9-4 Predicados de Leitura Padrão

readint (IntegerVariable) **integer–(0)**

Lê um inteiro. A variável tem que estar livre e o valor ao qual se liga tem que estar no domínio "int" padrão. Lê do dispositivo de entrada atual até a operação de RETURN.

readreal (RealVariable) **real–(0)**

Lê um número real. A variável tem que estar livre e o valor ao qual se liga tem que estar no domínio "real" padrão. Lê do dispositivo de entrada atual até que RETURN seja acionado.

radchar (CharVariable) **char–(0)**

Lê um caractere. A variável tem que estar livre e o valor ao qual se liga tem que estar no domínio "char" padrão. Lê um único caractere da posição atual do cursor. Ao contrário de "readint" e "readreal", "readchar" não espera que RETURN seja operado.

readln (String) **string–(0)**

Lê uma *string*. A variável tem que estar livre e o valor ao qual se liga tem que estar no domínio "string" padrão. Lê do dispositivo de entrada atual até encontrar um retorno de carro ASCII.

readterm (Domain, Term) **(O,I)**

Lê um objeto escrito pelo predicado "write". A variável "Term" será ligada ao objeto que é lido se este objeto está na seção de domínio. Isto é útil na manipulação de arquivos, porque permite que você traga informação-objeto de fora do programa.

readdevice (SymbolicFilename) **symbol–(I)(O)**

Permite que você especifique o predicado "readdevice" ativo (atual). Pode funcionar com qualquer um dos três tipos de "SymbolicFilename": os arquivos simbólicos predefinidos (teclado) o nome de um arquivo simbólico definido pelo usuário para um arquivo aberto; ou uma variável não-ligada. Caso uma variável não-ligada seja utilizada, o predicado liga "SymbolicFilename" ao nome do "readdevice" atual. Os valores devem estar no domínio "symbol" padrão.

inkey(char-O)

Lê um caractere do dispositivo de entrada padrão. Caso não consiga ler tal caractere, "inkey" falha.

READINT (INTVARIABLE)

A Figura 9-9 mostra o programa apresentado na Figura 9-7, mas com acréscimo

de linhas. Estas linhas mostram como "readint" lê um único valor inteiro do dispositivo "read" atual, que no caso é o teclado. O predicado "readint" não sabe que você ofereceu um inteiro antes de RETURN ser teclado (Figura 9-10).

READREAL (REALVARIABLE)

Este predicado funciona quase que da mesma forma que "readint"; só que trabalha com números reais em lugar de inteiros. Caso você ofereça algo diferente de um número real, vai falhar.

READCHAR (CHARVARIABLE)

Este predicado também funciona de forma semelhante ao "readint". Entretanto, pelo fato de querer apenas um único caractere, não espera por um retorno de carro (uma operação de RETURN) para a obtenção da variável. Assim que tem um retorno de carro, volta ao restante do programa. O valor tem que estar no domínio de "char". A Figura 9-11 mostra "readreal" e "readchar" acrescentados ao mesmo programa.

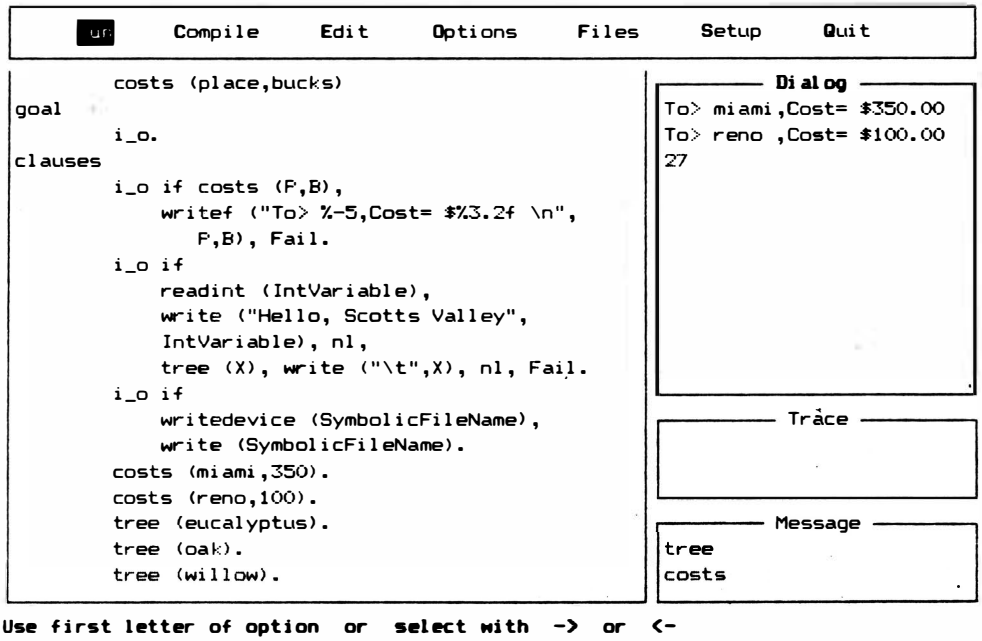


Figura 9-9 Utilização de "readint" para ler um único valor inteiro do dispositivo "read" atual

READLN (STRINGVARIABLE)

Caso você queira ler caracteres até que seja encontrado um retorno de carro, o predicado "readln" lhe permite isto. Uma *string* não precisa ter apenas tipo de domínio "char". A Figura 9-12 mostra como "readln" é utilizado para ler uma linha de texto.

READDEVICE (SYMBOLICFILENAME)

O predicado "readdevice" é muito semelhante ao predicado "writedevice". Permite determinar a partir de qual dispositivo (teclado ou outro) outro predicado "read" receberá informação. Isto poderá ser o nome de arquivo predefinido "keyboard" ou um arquivo que você nomear que esteja aberto. (A Tabela 9-3 relaciona os dispositivos predefinidos.)

Se a variável "SymbolicFileName" é não-instanciada, o predicado "readdevice" vai ligá-lo ao nome do dispositivo "read" ativo atual. Pelo fato de não ter aprendido ainda

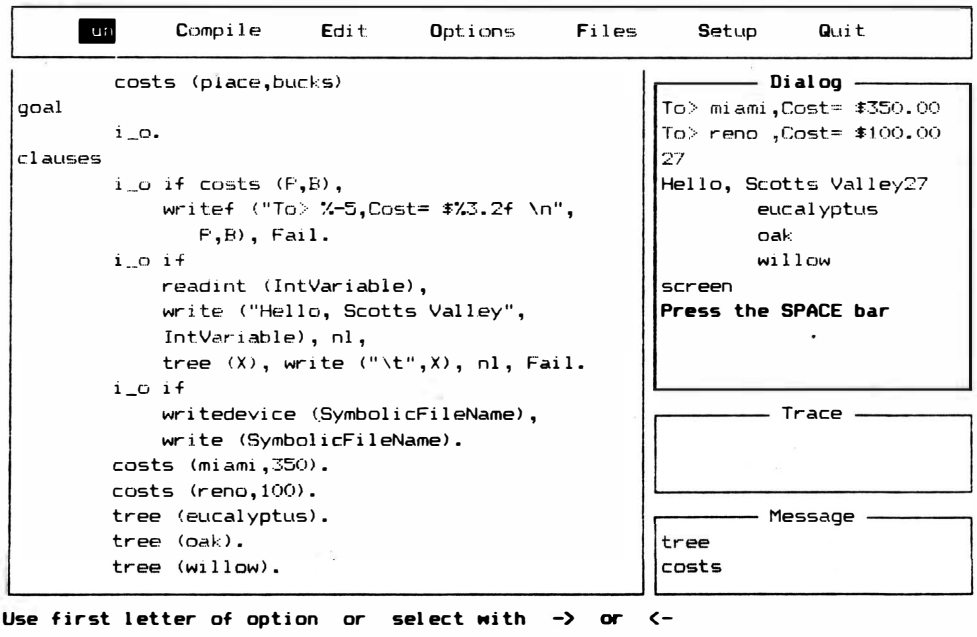


Figura 9-10 Apresentação depois de digitar um valor inteiro e antes de teclar RETURN

a respeito de predicados sobre arquivos, a Figura 9-13 mostra um exemplo de ligação do "SymbolicFileName" do predicado "reddevice" ao nome padrão, e depois a escrita deste nome.

PREDICADOS DEFINIDOS PELO USUÁRIO

Assim como ocorre com os predicados de escrita, você poderá criar seus próprios predicados de leitura: poderá construir regras pela utilização dos predicados padrões. Experimente os predicados de escrita e leitura para ver que tipo de funções especializadas você pode executar.

JANELAS E POSIÇÃO DO CURSOR

Janelas – partes da tela que agem como telas próprias – tornaram-se uma parte padronizada dos programas de microcomputadores. Turbo Prolog depende muito de suas quatro janelas intrínsecas para tornar mais fácil e mais estruturado o desenvolvimento de

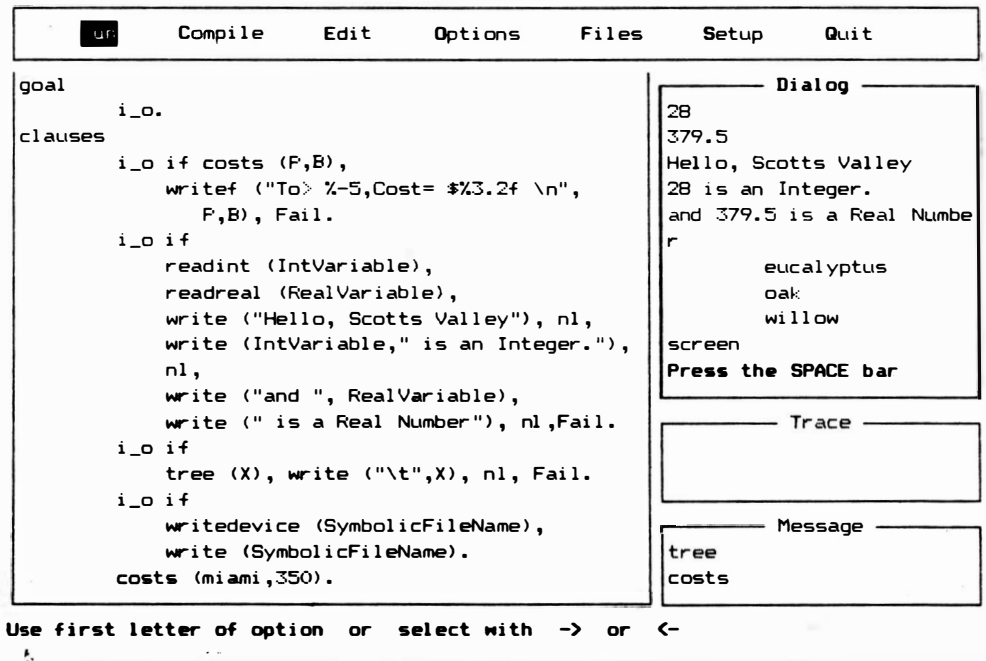


Figura 9-11 Acréscimo de "readreal" e "readchar"

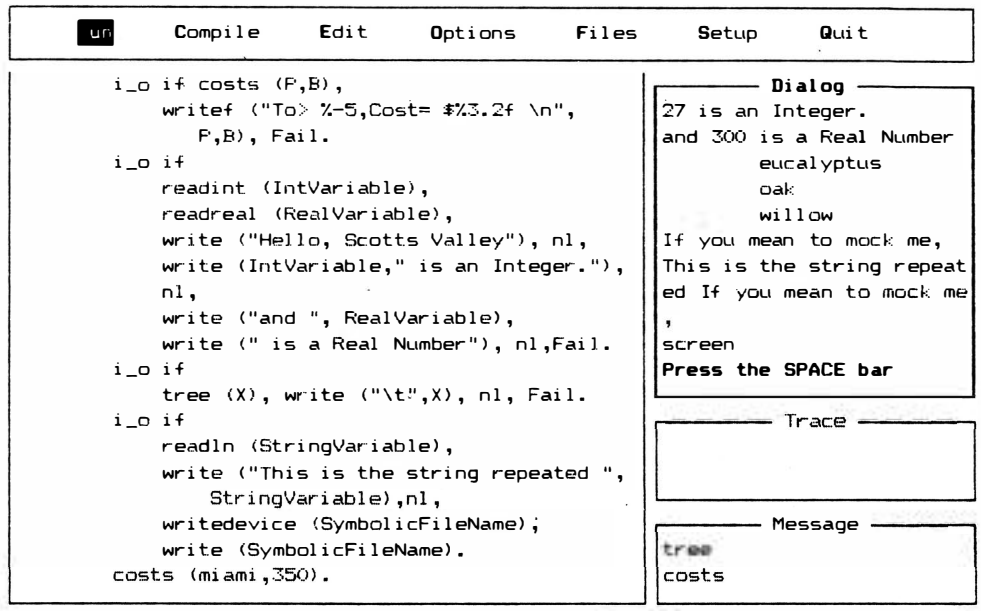


Figura 9-12 Utilizando "readln" para ler uma linha de texto

programas. Mas Turbo Prolog contém também predicados que lhe permitem formar suas próprias janelas para entrada e saída de programas. Mesmo em uma janela, você pode utilizar o predicado do cursor para começar a ler ou escrever em uma determinada posição.

A Tabela 9-5 relaciona os predicados de janela padrão. Inclui instruções para controlar a cor da tela.

Tabela 9-5 Predicados de janela padrão

makewindow(WindowNo,ScrAtt,FrameAttr,Header,Row,Col,Height,Width)
(int,int,int,string,int,int,int,int)(I,I,I,I,I,I,I,I)

Gera uma janela limpa. Todas as variáveis têm de estar ligadas.

WindowNo	Número de identificação da janela. Cada janela tem um número inteiro diferente.
ScrAtt	Determina o tipo de tela para a janela. As Tabelas 9-6 e 9-7 mostram cálculos de atributo.
FrameAtt	Determina a existência e estilo para a moldura em torno da janela. As Tabelas 9-6 e 9–7 mostram cálculos de atributo. O valor 0 do atributo significa nenhuma moldura.

Header	*String* escrita no topo da janela.
Row	Número (inteiro) de linha. As linhas são contadas a partir do topo da tela. Com valores além das dimensões da tela (25 linhas), teremos uma mensagem de erro.
Col	Número (inteiro) da coluna. As colunas são contadas a partir da esquerda da tela. Caso o valor esteja além das dimensões da tela (80 colunas), resultará uma mensagem de erro.
Height	Número de linhas (inteiro) na janela. Caso a soma de "Height" e "Row" coloque a parte inferior da janela fora da tela, resultará uma mensagem de erro.
Width	Número (inteiro) de colunas na janela. CAso a soma de "Width" e "Col" coloque o lado direito da janela fora da tela, ocorrerá uma mensagem de erro.

Nota: O predicado "graphics" pode ser utilizado para alterar o número de colunas e linhas na tela. Os modos CGA e EGA podem chegar até 640 x 200 e 640 x 350, respectivamente.

window_attr(Attr) inteiro–(I)

Altera o atributo da janela ativa para o valor inteiro do valor ligado. Permite que você altere o estilo da tela ou a cor da janela aberta, sem retroceder ao predicado "makewindow".

removewindow

remove a janela ativa da tela. Não tem argumentos.

shiftwindow(WindowNo) inteiro–(I)(O)

Se a variável estiver livre, vai ligá-la ao número da janela ativa. Se a variável estiver ligada, vai tornar "WindowNo" a nova janela ativa. Permite que você comute entre múltiplas janelas que podem estar na tela. Os predicados "write" e "read" trabalham apenas na janela ativa.

clearwindow

retira todos os caracteres da janela ativa e os substitui com espaços em branco com atributo de fundo.

cursor (Row, Column) inteiro,inteiro–(I,I)(O,O)

requer que ambas as variáveis estejam livres ou ligadas. Caso ambas estejam livres, vai ligá-las aos inteiros, representando as posições atuais de linha e coluna do cursor na janela ativa ou na tela. Caso as duas estejam ligadas, moverá o cursor ao ponto que representam na grade.

cursorform (Starline,Endline) (inteiro,inteiro)–(I,I)

Na área de um caractere, se estabelece a posição do cursor X–Y. Cada caractere ocupa 14 linhas de varredura da tela. "Starline" e "Endline" devem estar ligados a valores entre 1 e 14 (inclusive).

window_str(ScreenString) string–(I)(O)

Caso a variável esteja ligada, escreverá o valor de "ScreenString" à janela ativa. Começa na posição atual do cursor, trunca linhas longas demais (não faz divisão de palavra), e trunca linhas que ficariam fora da parte inferior da janela. Caso a variável esteja livre, esta variável será instanciada à *string* apresentada na janela ativa. A variável terá tantas linhas quantas houver na janela, e cada linha será lida até o último caractere não-vazio.

MAKEWINDOW

Para criar uma janela, basta utilizar o predicado "makewindow" e preencher seus argumentos. Todos os argumentos têm de ser instanciados no momento em que este predicado é chamado, e todos, exceto "FrameAttr" (que é uma *string*), têm de ser do tipo de domínio inteiro. Entretanto, os valores são mais restritos ainda. Lembre-se de que a tela mostra 25 linhas e 80 colunas. Caso seus valores de posição e tamanho coloquem uma janela além deste território, você receberá uma mensagem de erro.

A Figura 9-14 mostra este predicado de janela básico adicionado ao programa

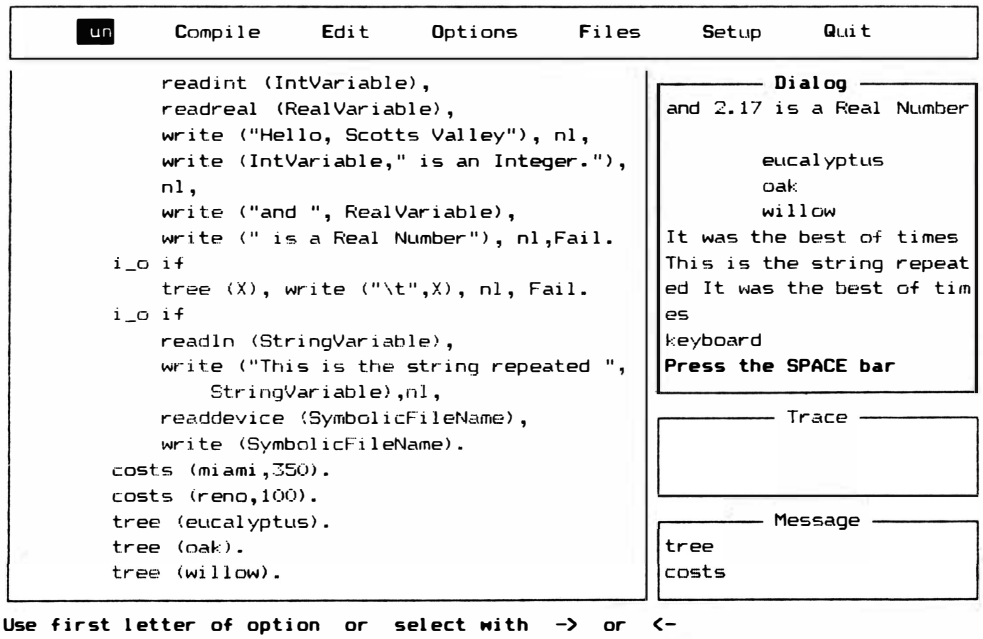


Figura 9-13 Exemplo da ligação de "SymbolicFileName" do predicado "readdevice" ao nome default

feito até agora. A inspeção dos seguintes argumentos mostra-nos o tipo de janela que vai ser.

```
makewindow (1, 7, 112, "A Window", 5, 20, 20, 28)
```

WINDOWNO

O "1" para "WindowNo" especifica apenas que esta será a janela número 1. Utilize o número da janela para identificar a janela ativa.

SCRATT

O valor "7" de "ScrAtt" (Atributo de Tela) estabelece que a área principal da janela deve ter caracteres brancos em um fundo preto, caso um monitor monocromático esteja sendo utilizado. Veja na Tabela 9-6 uma lista de valores que você pode escolher.

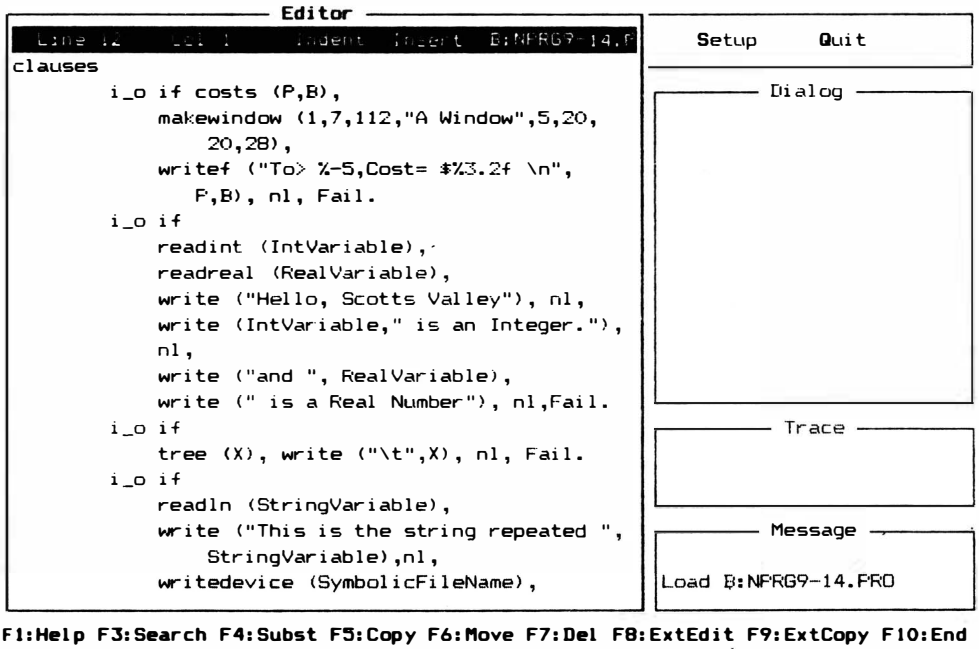


Figura 9-14 Predicado de janela básico, acrescentado ao programa

Para escolher o atributo de um monitor monocromático, encontre o valor para a decisão maior e ajuste-o pelos valores possíveis de decisão menor. A Tabela 9-7 mostra os valores semelhantes para um monitor colorido. (Os valores no manual original da Borland estavam errados.) Para calcular o valor de atributo desejado para um monitor colorido, escolha uma cor de primeiro plano e uma cor de fundo. Os caracteres serão apresentados na cor do primeiro plano. A soma dos valores para as duas cores é o atributo final.

Tabela 9-6 Atributos para monitores monocromáticos

Decisões maiores:

Estilo	**Valor**	**Primeiro Plano (caracteres)**	**Fundo**
Vazio	0	Preto	Preto
Normal	7	Branco	Preto
Vídeo reverso	112	Preto	Branco

Decisões menores:

1. Sublinhar caracteres na cor de primeiro plano: acrescente 1.
2. Mostrar a parte branca do display em alta intensidade: some 8.
3. Piscar caracteres: acrescente 128%.

Tabela 9-7 Atributos para monitores coloridos*

Cor **Em ordem de cor:**	**Primeiro plano**	**Fundo**
Preto	0	0
Cinza	8	na
Azul	1	16
Azul claro	9	na
Verde	2	32
Verde claro	10	na
Verde azulado	3	48
Verde azulado claro	11	na
Vermelho	4	64
Rosa	12	na
Magenta	5	80
Magenta claro	13	na
Marron	6	96
Amarelo	14	na
Branco	7	112
Branco intenso	15	na

Em ordem de valores:

Preto	0	0
Azul	1	16
Verde	2	32
Verde azulado	3	48
Vermelho	4	64
Magenta	5	80
Marron	6	96
Branco	7	112
Cinza	8	na
Azul claro	9	na
Verde claro	10	na
Verde azulado claro	11	na
Rosa	12	na
Magenta claro	13	na
Amarelo	14	na
Branco intenso	15	na

* "na" significa que esta cor em particular não está disponível para o fundo ("branco intenso" significa branco de alta intensidade).

FRAMEATTR

O valor "112" para "FrameAttr" (Atributo de Quadro) diz que um *vídeo reverso* (caracteres pretos em fundo branco) deve emoldurar a janela. Caso você escolha "O" para este atributo, terá mais espaço para manobras na janela, porque o quadro utiliza duas linhas e duas colunas – tem a largura de um caractere em torno da janela.

HEADER

O cabeçalho "A Window" é apenas uma *string* mostrada no topo da janela como um título.

ROW, COL

Os próximos dois argumentos devem ser considerados juntos. Os valores "5,20" para estes dois especifica que a janela deve começar na linha 5, coluna 20.

HEIGHT, WIDTH

Estes dois argumentos também podem ser considerados como um par. Os valores de "20,28" especifica que a janela deve ter a altura de 20 linhas e largura de 28 colunas. A Figura 9-15 mostra a janela que esta linha gera. Os predicados "write" que vêm depois de "makewindow" nesta regra realizam suas tarefas na janela. Sempre se aplicam à janela ativa.

WINDOW_ATTR

Este predicado tem de ter um valor instanciado. O valor daquele argumento torna-se o atributo de tela da janela atualmente ativa.

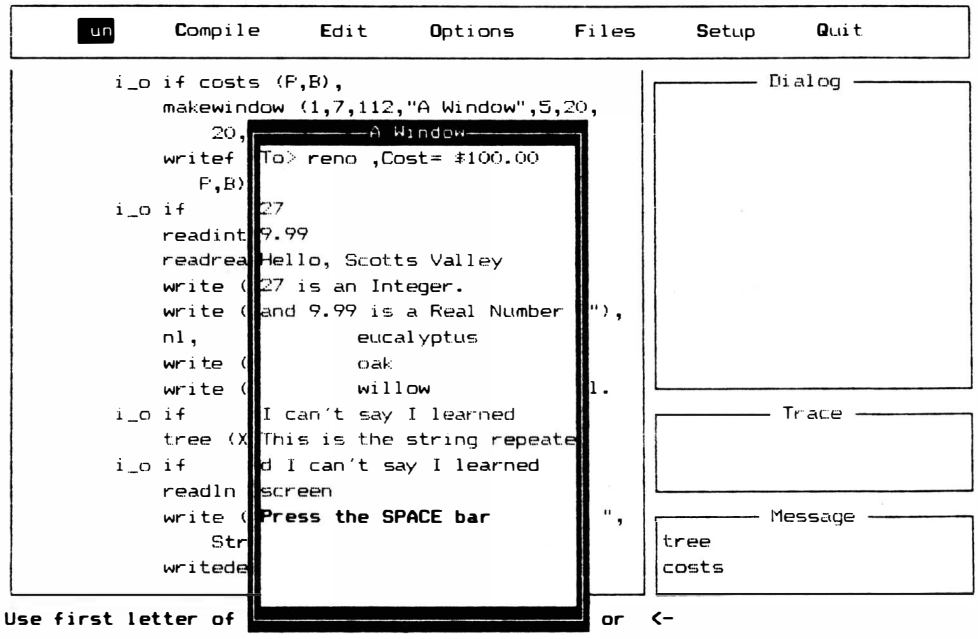


Figura 9-15 Janela criada com argumentos de altura e largura

REMOVEWINDOW

Caso você queira cancelar uma janela inteira da tela, basta chamar o predicado "removewindow". A janela ativa desaparece imediatamente

SHIFTWINDOW

Este predicado permite que você determine a janela ativa. Sem ele, a janela criada mais recentemente será a ativa. O cursor vai voltar à sua posição mais recente na janela. O único argumento que "shiftwindow" pede é o número da janela para onde você quer mudar. Caso o argumento esteja desligado de um valor, o predicado "shiftwindow" vai ligá-lo ao número da janela ativa. Se você criou janelas sobrepostas, a janela "mais recentemente ativa" parecerá se sobrepor às janelas "menos recentemente ativas".

CLEARWINDOW

Caso você queira apagar os caracteres de uma janela inteira, utilize este predicado. Tudo o que restar será a cor do segundo plano.

CURSOR

Todos os predicados de leitura e escrita funcionam a partir da posição atual do cursor na janela corrente. Assim que uma posição de cursor estiver determinada, Turbo Prolog se lembrará dela, mesmo que você mude de janela para janela. Entretanto, você pode afetar também a posição com este predicado. Veja como foi acrescentado ao programa da Figura 9-16. Para aclarar as coisas, a janela foi ampliada. Figura 9-17 mostra o efeito. Compare isto com a Figura 9-15.

Observe que apesar do cursor ter iniciado na linha 4, coluna 6 da janela (e não de toda tela), o retorno de carro provocado pela operação de RETURN (que deve ser feito para informar ao "readint" que o inteiro está completo) movimentou o cursor de volta à coluna 0 da janela. Daí em diante, o cursor voltou àquela coluna. Você pode também utilizar este predicado para descobrir a linha e a coluna do cursor. Caso "Row" e "Col" tenham valores não-ligados quando este predicado é chamado, serão instanciadas aos valores correntes de linha e coluna.

O predicado "cursorform" permite que você tenha um controle ainda mais fino sobre a posição do cursor.

WINDOW_STR

Este é um predicado difícil de classificar. Escreve ou lê, trabalha com *strings* e é orientado para janelas. Caso seu argumento não esteja ligado, instancia aquele argumento com a *string* apresentada na janela ativa. Ele lê todas as linhas na janela e termina cada linha ao chegar ao último caractere não-vazio. Caso o seu argumento já esteja ligado a um valor de *string*, "window_str" escreverá aquela *string* na janela ativa. Ele vai truncar cada linha para combinar com a janela (sem mudança de linha) e truncará a *string* toda, caso atinja o fim da janela.

REVISÃO DO EDITOR DE JANELA

Você sabe como utilizar a janela Editor como ferramenta de desenvolvimento de programas. Você pode, também, entrar na janela diretamente a partir de um programa Turbo Prolog, utilizando os dois predicados descritos nesta seção e relacionados na Tabela 9-8. O predicado "display" que fora previamente descrito na seção do predicado "write"

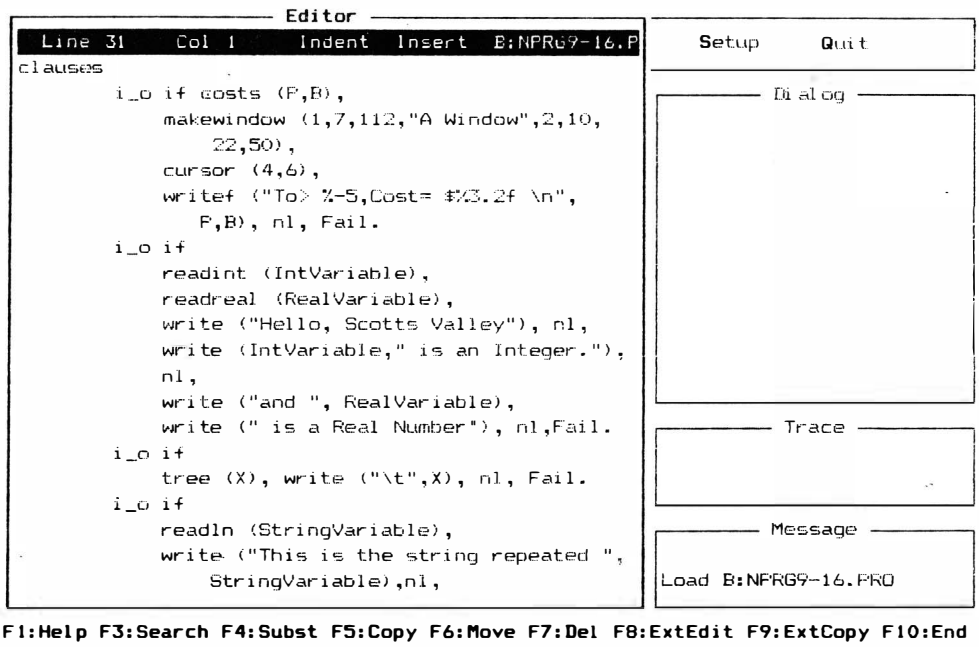


Figura 9-16 Acréscimo de predicados de "cursor" ao programa

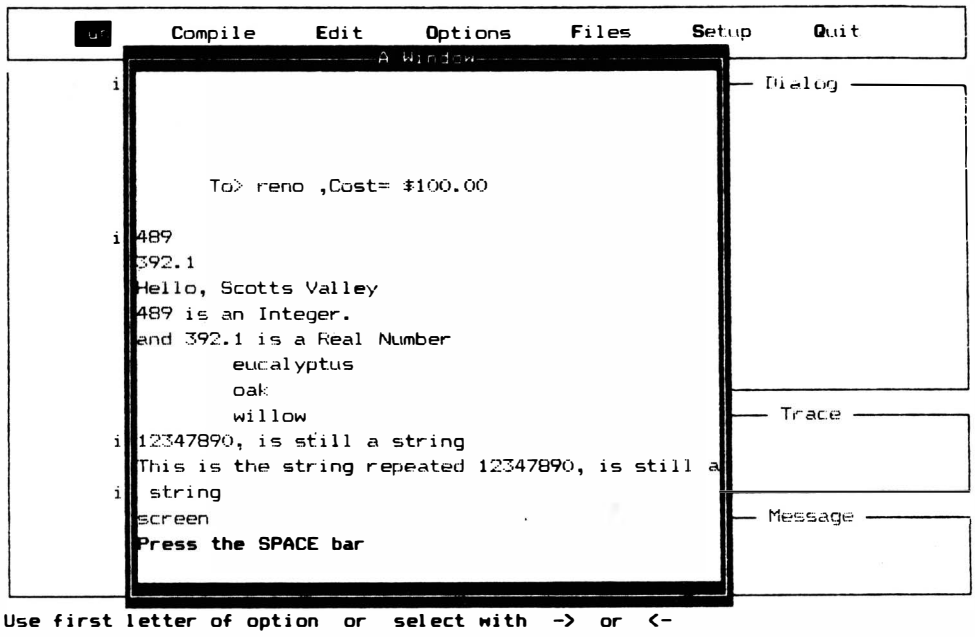


Figura 9-17 Efeito de uma janela ampliada

é algumas vezes considerado como este predicado porque permite que você utilize as teclas de cursor para inspecionar a *string* que apresenta.

Tabela 9-8 Predicados da Janela Editor

display(String) **string–(I)**

Precisa ter uma variávwel instanciada que esteja no domínio "string". Escreve o conteúdo de "string" na janela ativa na posição do cursor. Você pode, depois, inspecionar a *string*, utilizando os comandos das teclas de controle da janela Editor.

edit(InputString, OutputString) **string,string–(I,O)**

A variável "InputString" precisa ser instanciada e a variável "OutputString" deve estar livre. Ela permite que você utilize os comandos da janela Editor na janela ativa para editar "InputString". O resultado torna-se "OutputString".

editmsg(InStr,OutStr,LeftHeader,RightHeader,Message,Position,Help,FileName,Code)
string,string,string,string,string,string,integer,string,integer (I,0,I,I,I,I,I,0)

Todas as variáveis com exceção de "OutStr" e "Code" devem ser instanciadas. Ativa comandos do Editor na janela corrente, insere os dois cabeçalhos no cabeçalho principal, e escreve "Message" na parte inferior da janela. O cursor está em "Position" em "InStr" que você depois pode editar. O resultado torna-se "OutStr" e "Code" informa como a edição terminou. Um código "O" significa que a tecla F10 foi utilizada; um código "1" significa que a tecla ESC foi utilizada. "HelpFileName" especifica o arquivo que vai ser chamado se a tecla F1 for operada para obtenção de ajuda durante a edição.

EDIT

Este predicado escreve uma *string* na janela corrente e depois permite que você edite a *string* com os comandos da janela Editor. A *string* editada depois será instanciada com a variável "OutputString".

EDITMSG

Os oito argumentos deste predicado permitem que você chame o Editor, escreva uma *string* nele, edite esta *string*, insira dois textos no cabeçalho, desloque o cursor, e apresente uma mensagem na parte inferior da janela. O predicado "editmsg" especifica também que o arquivo Help deve ser carregado quando a tecla F1 é operada. O valor da variável "Code" informa-lhe como a edição terminou. Um valor 0 significa que a tecla F10 foi utilizada. Um valor 1 significa que a tecla ESC foi utilizada.

MANIPULAÇÃO DE TELAS: CARACTERES E CAMPOS

Diversos predicados permitem que você trabalhe em cada posição de caractere ou posição de campo na tela completa ou em uma janela. (Um campo é um trecho de posições de caracteres lado-a-lado em uma linha. Você define campos antes de utilizá-los – não são predefinidos pelo sistema.) Estes predicados são listados na Tabela 9-9. Você consegue que o trabalho seja feito mais rapidamente utilizando os predicados de campo em vez dos predicados de caractere.

Tabela 9-9 Predicados de manipulação de tela

scr_char (Row, Col, Char) **inteiro,inteiro,char–(I,I,I) (I,I,O)**

Se todas as variáveis estão ligadas, escreve "Char" com atributo atual na posição especificada pelos valores inteiros de "Row" e "Col". As linhas são contadas a partir de cima e as colunas a partir da esquerda.

Caso "Row" e "Col" estejam ligadas e "Char" livre, "scr_char" lerá o caractere que está na posição especificada por "Row" e "Col".

scr_attr (Row, Col, Attr) inteiro,inteiro,inteiro–(I,I,I) (I,I,O)

Se todas as variáveis estão ligadas, dá o atributo da posição especificada pelos valores inteiros de "Row" e "Col" a "Attr". As linhas são contadas a partir de cima e colunas a partir da esquerda. Caso "Row" e "Col" estejam ligados e "Attr" livre, "scr_attr" lerá o atributo que está na posição especificada por "Row" e "Col".

field_str(Row, Col, Lenght, String)
inteiro,inteiro,inteiro,inteiro–(I,I,I,I) (I,I,I,O)

Se todas as variáveis estão ligadas, se "Row" e "Col" especificam um valor na janela ativa e se o campo com "Length" de comprimento que cabe na janela começa na posição "Row, Col", então "String" será escrito naquele campo. Caso "String" seja mais longo que "Length" todos os caracteres além de "Length" serão truncados. Caso todas as variáveis com exceção de "String" estejam ligadas, se os valores "Row" e "Col" especificam um valor na janela ativa, e se um campo de comprimento "Length" que cabe na janela inicia na posição "Row" e "Col", então "String" no campo será ligada a "String".

field_attr, (Row, Col, Lenght, Attr)
inteiro,inteiro,inteiro,inteiro – (I,I,I,I) (I,I,I,O)

Se todas as variáveis estão ligadas, se os valores "Row" e "Col" especificam um valor na janela ativa, e se um campo de comprimento "Length" cabe na janela e começa na posição "Row, Col", então todas as posições do campo recebem o atributo "Attr". Se todas as variáveis com exceção de "Attr" estão ligadas, se os valores "Row" e "Col" especificam um valor na janela ativa, e se um campo de comprimento "Length" que cabe na janela inicia na posição "Row, Col", então o atributo da primeira posição (aquele em "Row, Col") daquele campo estará ligado a "Attr".

SCR_CHAR

Este predicado utiliza um número de linha e um número de coluna para especificar a posição de um único caractere. Pode escrever um caractere naquela posição ou ler um caractere daquela posição, dependendo da ligação de sua variável "Character".

SCR_ATTR

O predicado "scr_attr" (atributo de tela) trata o atributo da posição de caractere especificado pelos números de linha e coluna. Pode especificar ou ler o atributo que está ali (veja Tabelas 9-6 e 9-7 para o significado dos atributos).

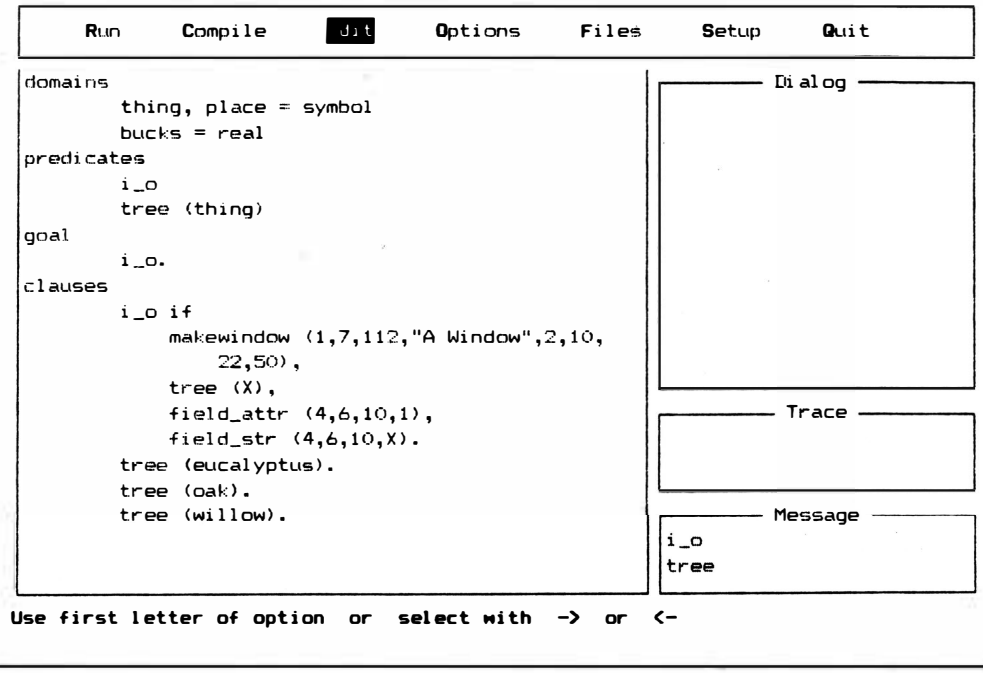


Figura 9-18 Acrescentando "field_str" e "field_attr" ao programa

FIELD_STR

Análogo ao predicado "scr_char", "field_str" especifica um campo (escolhendo a linha, a coluna e o comprimento nos caracteres) e depois lê ou escreve uma *string* para aquele campo.

FIELD_ATTR

Este predicado permite que você estabeleça ou leia o atributo para um campo todo de uma vez. Ele também especifica o campo pelos argumentos linha, coluna e comprimento. Por ocasião da leitura do atributo de um campo, "field_attr" depende do atributo da posição do primeiro caractere do campo. Esta especificação é necessária porque um campo poderá ter sua *string* escrita por um predicado de campo, e seus atributos especificados um caractere por vez pelo predicado "scr_attr". As Figuras 9-18 e 9-19 mostram a ação de "field_str" e "field_attr".

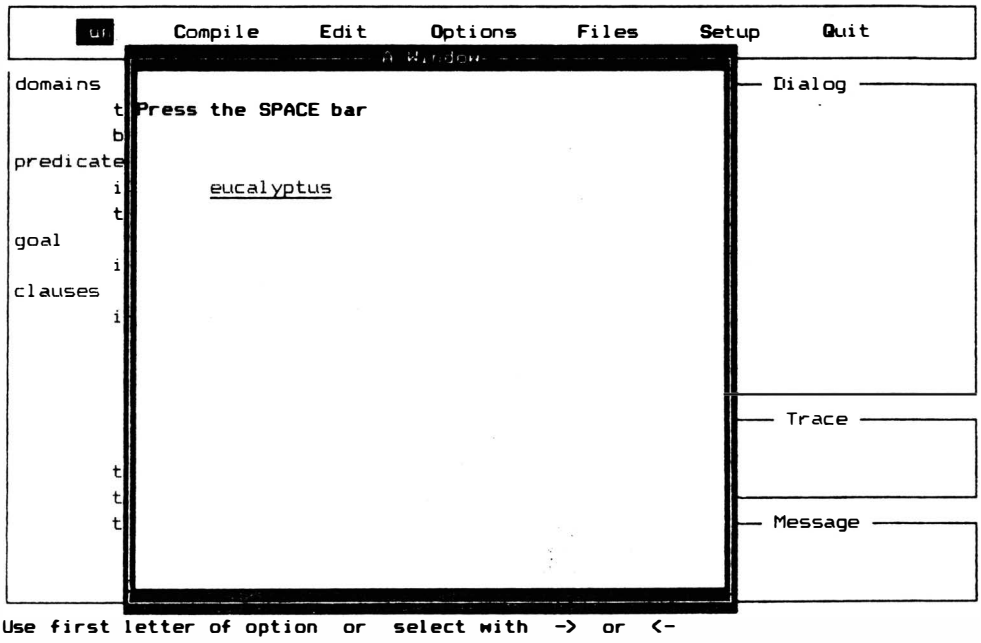


Figura 9-19 Efeito de "field_str" e "field_attr"

ATTRIBUTE

Predicado de utilização mais geral que o anterior, "attribute" permite que você decida ou leia qual é o atributo geral da tela.

CAPÍTULO **10**

ARQUIVOS

Para escrever programas práticos, grandes, que utilizem fatos armazenados e regras, você tem de ser capaz de ler e escrever arquivos em disco a partir de um programa em execução. Este tipo de E/S não é mais complexo do que as leituras e escritas da tela, explicadas no Capítulo 9. Mas para fazê-lo, é preciso aprender mais alguns predicados padrões. Não vamos gastar muito tempo em arquivos, porque assim que você tiver declarado e aberto um arquivo, a leitura e escrita dele funcionarão exatamente da mesma forma como funciona quando você lê e escreve informação da tela. Tudo aquilo que você iria enviar à tela ou impressora poderá enviar ao disco. Este capítulo descreve três pontos: predicados de arquivo, utilização direta dos comandos DOS a partir do Prolog e conversão do tipo de dados. Os predicados de arquivo permitem que você leia, escreva e modifique arquivos em disco, e os predicados DOS permitem que você vá diretamente à memória ou às portas de E/S. Caso você não esteja pronto para ler e escrever arquivos de seus programas Prolog deixe este capítulo de lado. Tudo o que você realmente precisa saber a respeito de arquivos é que deve declará-los antes de utilizá-los, abri-los antes de chamá-los e fechá-los para gravar alterações feitas neles.

PREDICADOS DE ARQUIVOS

Arquivos são entidades lógicas na programação de computadores utilizados para coletar dados relacionados. Os dados podem ser de qualquer outro domínio (caractere, real, inteiro, string, símbolo ou definido pelo usuário), e o relacionamento entre eles pode ser tão simples como "estão todos no mesmo local" (o termo "relação" é utilizado aqui de forma geral, e não no sentido de "objetos-relacionamentos", apresentados anteriormente neste livro).

Todo arquivo tem um nome, como você sabe por ter visto diretórios de discos. Sistemas operacionais em disco utilizam arquivos (com nomes de arquivo específicos) para organizar os dados no disco. A documentação Turbo Prolog refere-se a eles como "Dos-FileNames". Um fato interessante que deve ser lembrado é que Turbo Prolog trata os arquivos de disco, a tela, a impressora, a porta paralela, a porta serial e o teclado como arquivos. Você pode ler ou escrever em qualquer um destes "arquivos" ou "dispositivos". De fato, você pode ler ou escrever em diversos deles, no mesmo programa. Você utiliza os mesmos predicados de leitura e escrita (aqueles descritos no Capítulo 9) para operar com qualquer dispositivo ou arquivo. Portanto, não é preciso aprender mais nada a respeito de leitura e escrita. Tudo o que você tem a aprender é como especificar, abrir e fechar arquivos.

ESPECIFICANDO ARQUIVOS OU DISPOSITIVOS

Os dispositivos padrões estão relacionados na Tabela 10.1. Estes não precisam ser declarados e estão sempre disponíveis para o uso. Os dispositivos implícitos, ou aqueles que são especificados automaticamente até que o programa informe o contrário, são *keyboard* (teclado) para entrada e *screen* (tela) para saída. Um detalhe que deve ser observado é que apesar dos dispositivos *printer* e *coml* serem automaticamente declarados, não estão automaticamente prontos. Isto é, caso você decida escrever à impressora, mas a impressora (ou qualquer outra coisa que esteja ligada à porta paralela) não esteja conectada ou ligada, Turbo Prolog entra em um estado indeterminado. Utilize a tecla CTRL-BREAK para tentar sair desta condição. O mesmo problema pode ocorrer caso tente utilizar (ler ou escrever) a porta serial, coml. Caso não haja nenhuma placa serial em seu computador, obterá um erro de processamento.

Os predicados que permitem o redirecionamento da entrada e saída estão indicados na Tabela 10.2. São facilmente utilizados e estão disponíveis para especificar ou inspecionar o dispositivo corrente. Caso a variável "SymbolicFileName" esteja ligada, o dispositivo atual vai mudar para o valor da variável. Caso "SymbolicFileName" esteja livre, será ligado ao valor atual. A operação básica destes predicados foi mostrada na Figura 9-13 do Capítulo 9.

Tabela 10-1 Dispositivos padrões

Dispositivo	**Comentário**
keyboard (teclado)	(este é o dispositivo implícito de entrada – ou dispositivo de leitura)
printer (impressora)	porta paralela
coml	porta serial
screen (tela)	(este é o dispositivo implícito de saída – ou dispositivo de escrita)

Tabela 10-2 Predicados de especificação dos dispositivos

readdevice(SymbolicFileName)	**símbolo–(I) (O)**

Quando Turbo Prolog começa, o dispositivo de leitura implícito é o teclado. Você pode utilizar este predicado de duas formas: para alterar o dispositivo de leitura corrente ou descobrir o nome do dispositivo de leitura corrente. Quaisquer predicados "read" agem sobre o dispositivo de leitura atual.

Caso a variável "SymbolicFileName" esteja instanciada (ou ligada) a um dispositivo padrão ou a um arquivo definido pelo usuário, o predicado "readdevice" altera o dispositivo de leitura atual para o valor de "SymbolicFileName". Você pode fazer isto tantas vezes quantas desejar no programa, e pode retornar ao dispositivo de leitura original se desejar.

Caso a variável "SymbolicFileName" não esteja ligada a um dispositivo padrão ou definida pelo usuário, o predicado "readdevice" liga-a ao valor do dispositivo de leitura corrente.

writedevice(SymbolicFileName)	**símbolo–(I) (O)**

Quando Turbo Prolog começa o dispositivo de escrita implícito é a tela. Você pode utilizar este predicado de duas formas: para alterar o dispositivo de escrita corrente, ou descobrir o nome do dispositivo de escrita corrente. Qualquer predicado "write" vai atuar sobre o dispositivo de escrita atual.

Caso a variável "SymbolicFileName" esteja instanciada (ou ligada) a um dispositivo padrão ou a um arquivo definido pelo usuário, o predicado "writedevice" altera o dispositivo de escrita corrente para o valor de "SymbolicFileName". Você pode fazer isto tantas vezes quantas desejar no programa, e pode voltar ao dispositivo de escrita corrente se desejar.

Caso a variável "SymbolicFileName" não esteja ligada a um dispositivo padrão ou arquivo definido pelo usuário, o predicado "writedevice" liga-a ao valor do dispositivo de escrita corrente.

Tabela 10-3 Declarações de domínio de arquivo

SymbolicFileName

Para declarar um domínio de arquivo, você precisa declarar os nomes de arquivos simbólicos que vai utilizar. Você pode ter apenas uma única declaração de domínio de arquivo em um programa, mas a declaração poderá conter diversos nomes de arquivos simbólicos. Aqui está um exemplo:

```
        domínios

file=cloudnames;remains;employees_to_watch   [file=nomedenuvens;falta;funcionários_a_obser-
                                              var]
```

Todos os nomes de arquivos simbólicos devem iniciar com letra minúscula.

Tipos de arquivos

Existem dois tipos de arquivo:

- **Arquivos predefinidos** São os arquivos sempre disponíveis em Turbo Prolog. Você não precisa (de fato, não deve) fazer a sua declaração – Prolog sabe que estão ali. Arquivos predefinidos (padrão) estão na Tabela 10.1.

- **Arquivos definidos pelo usuário** Estes são arquivos que você designa. Qualquer nome Turbo Prolog que não é reservado (que não seja nem predicado nem operador) pode ser utilizado.

Exemplos são

thisfile	[estearquivo]
thatfile	[essearquivo]
the_other_file	[o_outro_arquivo]
employee_records_1986	[registro_funcionário_1986]

DECLARANDO ARQUIVOS

Você tem que declarar os nomes dos arquivos simbólicos que vai utilizar. Estas declarações pertencem à seção de domínios do programa. A Tabela 10-3 fornece os detalhes da declaração de arquivos. Cada programa pode ter apenas uma declaração de domínio de arquivo, mas a declarção poderá conter diversos nomes de arquivos simbólicos. Todos os nomes de arquivos simbólicos iniciam com letra minúscula. Existem dois tipos de arquivos: predefinidos e definidos pelo usuário.

ARQUIVOS PREDEFINIDOS

São os dispositivos padrões – os arquivos que estão sempre disponíveis no Turbo Prolog e não devem ser declarados. Estão incluídos aqui a impressora, o teclado a tela e o coml.

ARQUIVOS DEFINIDOS PELO USUÁRIO

São arquivos de disco para os quais pode-se utilizar qualquer nome simbólico desejado. Entretanto, você pode utilizar apenas um nome que não seja o mesmo que uma das palavras reservadas do Turbo Prolog (os nomes de predicados padrões e delimitadores de seção).

ABRINDO ARQUIVOS

Um arquivo tem de ser aberto antes que se possa operar com ele. Antes que você tente abrir um arquivo, precisa pensar em um nome simbólico para ele, o qual possa ser declarado na declaração de domínio de arquivo. Pode existir apenas uma declaração deste tipo em cada programa, mas a ela pode relacionar diversos arquivos. Diversos arquivos podem ser abertos ao mesmo tempo. Os predicados que abrem arquivos também relacionam seu nome simbólico a um nome de arquivo DOS real. Nomes DOS iniciam por uma letra e têm até oito letras antes de um ponto e uma extensão de três letras opcionais. Turbo Prolog normalmente utiliza o percurso de diretório padrão para encontrar o acionador e o diretório para um arquivo definido pelo usuário. Você pode alterar este percurso utilizando o predicado "disk" descrito posteriormente neste capítulo, na seção entitulada "Predicados para manipulação de arquivos".

FECHANDO ARQUIVOS

Ao terminar de trabalhar com um arquivo, deverá fechá-lo para garantir que qualquer modificação seja gravada. Caso contrário, alguma ou todas as modificações simplesmente desaparecerão.

PREDICADOS PARA MANIPULAÇÃO DE ARQUIVOS

A Tabela 10-4 relaciona os predicados de arquivo. Esta seção descreve alguns deles e fornece exemplos de sua ação.

EXISTFILE, RENAMEFILE, DELETEFILE

Estes provavelmente são os predicados de arquivo melhor compreendidos. "Existfile" simplesmente verifica se existe um arquivo com "DosFileName". O predicado "renamefile" vai alterar o nome do arquivo, e "deletefile" vai cancelar o arquivo. Você poderá precisar utilizar o próximo predicado para garantir que estes e outros predicados de arquivo estão procurando no local correto.

DISK (DOSPATH)

Caso "DosPath" esteja instanciado a um possível valor de string, aquele valor se tornará o novo acionador de disco e novo percurso padrão. Caso "DosPath" não esteja instanciado, será ligado ao acionador de disco e percurso corrente.

COMO FUNCIONAM OS PREDICADOS PARA MANIPULAÇÃO DE ARQUIVO

A Figura 10-1 mostra estes quatro predicados em funcionamento em um programa simples. Antes de se processar este programa, é necessário gravá-lo no acionador A como o arquivo "filetest.pro". (Você poderia gravá-lo em outro acionador se alterasse o argumento do predicado "disk" adequadamente.)

Processe o programa. Estabeleça a meta

```
file_play
```

na janela Dialog. O resultado será "True". Depois, processe o programa novamente. Verá a apresentação mostrada na Figura 10-2. O que aconteceu? Por que o programa funcionou uma vez e não novamente? Porque mudou o status dos arquivos no disco – os próprios arquivos dos quais depende.

Tabela 10.4 Predicados de arquivo

MANIPULAÇÃO DE ARQUIVOS

existfile(DosFileName) string–(I)

Este predicado requer uma variável instanciada (ou ligada). Verificará se um arquivo com o nome igual ao valor da variável está no disco. (O predicado "disk" especifica o arquivo e percurso que este predicado vai usar.) Caso encontre o arquivo, será bem-sucedido. Caso contrário, falhará.

deletefile(DosFileName) string–(I)

Este predicado requer uma variável instanciada (ou ligada). Vai cancelar o arquivo que tem o nome igual ao valor da variável. (O predicado "disk" especifica que acionador e percurso o predicado vai usar para encontrar o arquivo.) Caso cancele o arquivo ou não o encontre, será bem-sucedido.

renamefile(OldDosFileName,NewDosFileName) string,string–(I,I)

O predicado exige duas variáveis instanciadas. Vai encontrar o arquivo com o nome igual ao valor da primeira variável e vai alterar seu nome para o valor da segunda variável. (O predicado "disk" especifica o acionador e o percurso que o predicado vai usar.) É bem-sucedido com a renomeação do arquivo. Caso não encontre o arquivo para renomear, falhará.

disk(DosPath) string–(I)(O)

Este predicado pode funcionar de duas maneiras:

- Com uma variável instanciada (ou ligada) vai estabelecer o acionador e percurso ativo com o valor daquela variável.

- Com uma variável não-ligada, vai ligar a variável ao valor do acionador e percurso ativos.

Nota: Ambas as variáveis têm de ser ligadas para que qualquer um destes predicados seja bem-sucedido. A primeira variável é o nome simbólico (o nome que você quer utilizar no programa Prolog) do arquivo que quer abrir. A segunda variável é o nome do arquivo de disco (um nome DOS oficial) que responderá aos seus acessos de arquivos simbólicos. Em outras palavras, estes predicados estabelecem um relacionamento direto entre os dois arquivos e se referem apenas a um único arquivo. Assim que o arquivo estiver aberto, você poderá executar a ação relevante nele.

openwrite(SymbolicFileName,DosFileName) **arquivo,string–(I,I)**

Utilize este predicado para abrir um arquivo para escrita (enviando informação ao arquivo). Caso já exista um arquivo de nome "DosFileName", será cancelado para ser substituído pelo novo arquivo criado por este predicado.

openread(SymbolicFileName,DosFileName) **arquivo,string–(I,I)**

Utilize este predicado para abrir um arquivo para leitura (obter informações de um arquivo). Você não precisa se preocupar em fechar um arquivo que apenas foi lido: não existem modificações a serem perdidas.

openappend(SymbolicFileName,DosFileName) **arquivo,string–(I,I)**

Utilize este predicado para abrir um arquivo ao qual quer acrescentar informações.

openmodify(SymbolicFileName,DosFileName) **arquivo,string–(I,I)**

Utilize este predicado para abrir um arquivo para leitura e escrita (enviar informação e obter informação de).

POSICIONAMENTO EM UM ARQUIVO

filepos(SymbolicFileName,FilePosition,Mode)
arquivo,real,inteiro–(I,I,I) (I,O,I)

Existem duas maneiras de utilizar este predicado: para encontrar a posição do arquivo e estabelecer a posição do arquivo. Para encontrar a posição do arquivo, "Mode" tem de ser instanciado (ou ligado) com 0 (veja a lista de modos que segue), "SymbolicFileName" tem de ser instanciado (ou ligado), e "FilePosition" deve estar livre. O predicado vai ligar "FilePosition" à posição atual no arquivo onde a próxima operação de escrita ou leitura ocorrerá. Esta posição avança depois de cada leitura ou escrita, ou pula para a posição estabelecida pela segunda utilização deste predicado.

Para estabelecer a posição do arquivo, todas as três variáveis têm de ser instanciadas. "SymbolicFileName" informa sobre que arquivo acionar, "FilePosition" fornece um valor nu-

mérico para a posição, e "Mode" informa Prolog como interpretar o número "FilePosition". O número pode ser contado do começo do arquivo, da posição atual ou em retrocesso a partir do final do arquivo.

Modo	De onde contar a "FilePosition"
0	Começo do arquivo
1	Posição atual do arquivo
2	Fim do arquivo, retrocedendo

Em cada caso, a primeira posição é a posição 0, e não posição 1.

eof(SymbolicFileName) **file – (I)**

Será bem-sucedido se a posição atual do arquivo no arquivo "SymbolicFileName" for ASCII 26 (também conhecido por CTRL-Z). Este código indica o fim do arquivo.

FECHANDO O ARQUIVO

closefile(SymbolicFileName) **file – (I)**

Fecha o arquivo com o nome "SymbolicFileName". Caso este arquivo não esteja aberto, o predicado será bem-sucedido de qualquer modo. Utilize este predicado para garantir que qualquer arquivo em que se escreva ou se modifique esteja completo e não se vai perder nenhum dado.

OUTROS

filestr(DosFileName,StringVariable) **string,string – (I,O)**

Encontra o arquivo especificado por "DosFileName" e lê caracteres dele até que um limite de 64k seja atingido ou um código de fim-de-arquivo (ASCII 26) seja encontrado. Os caracteres são depois atribuídos à *string* "StringVariable".

Para utilizar o comando Trace com a finalidade de descobrir o que aconteceu, utilize primeiro o menu Files para armazenar novamente o programa como "filetest.pro" no acionador A. Depois, acrescente o predicado "trace" acima da linha "domains". Processe o programa e coloque a meta "file_play". Comece a teclar F10 para acompanhar a ação. Em seguida, verá Prolog chamar a meta e depois se deslocar para a primeira e única cláusula. O primeiro passo para satisfazer aquele regra é chamar o primeiro predicado do lado direito, o predicado "disk". O percurso do acionador de disco será estabelecido em "A:" e este predicado será bem-sucedido. Você o verá retornar; depois, o segundo predicado da di-

reita, o predicado "existfile", será tomado. Pelo fato de ter acabado de arquivar o arquivo "filetest.pro" naquele acionador, você sabe que este predicado também será bem-sucedido. Depois, o predicado "renamefile" será chamado e alterará o nome do arquivo "filetest.pro" para o arquivo "newname.pro". Um predicado verificará o próximo. O acompanhamento até este ponto é indicado na Figura 10-3.

O segundo predicado "existfile" testa para ver se "renamefile" realizou a tarefa. Deveria, tê-lo feito, e este predicado será bem-sucedido. Finalmente, o predicado "deletefile" tornará absolutamente garantido que o arquivo "filetest.pro" original seja cancelado no disco. Isto não é necessário, mas mostra que o predicado "deletefile" pode ser bem-sucedido mesmo que não possa cancelar um arquivo que não esteja presente. Esta parte da pesquisa é mostrada na Figura 10-4. Finalmente, a meta original terá sido conseguida, e "True" aparecerá na janela "Dialog".

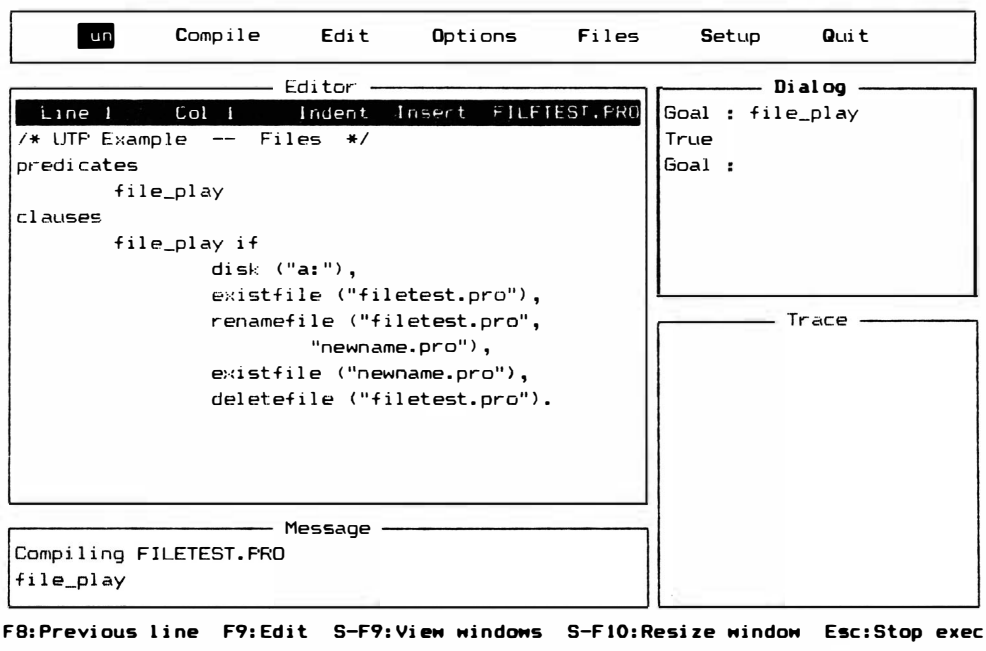


Figura 10-1 Quatro predicados de arquivo em ação

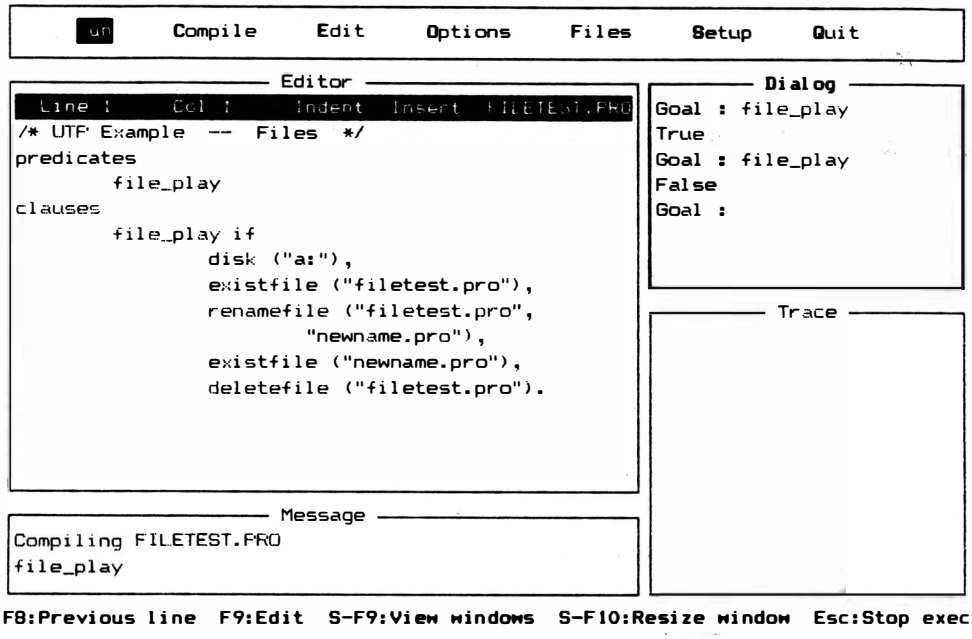


Figura 10-2 O resultado de processar o programa "file_play" uma segunda vez

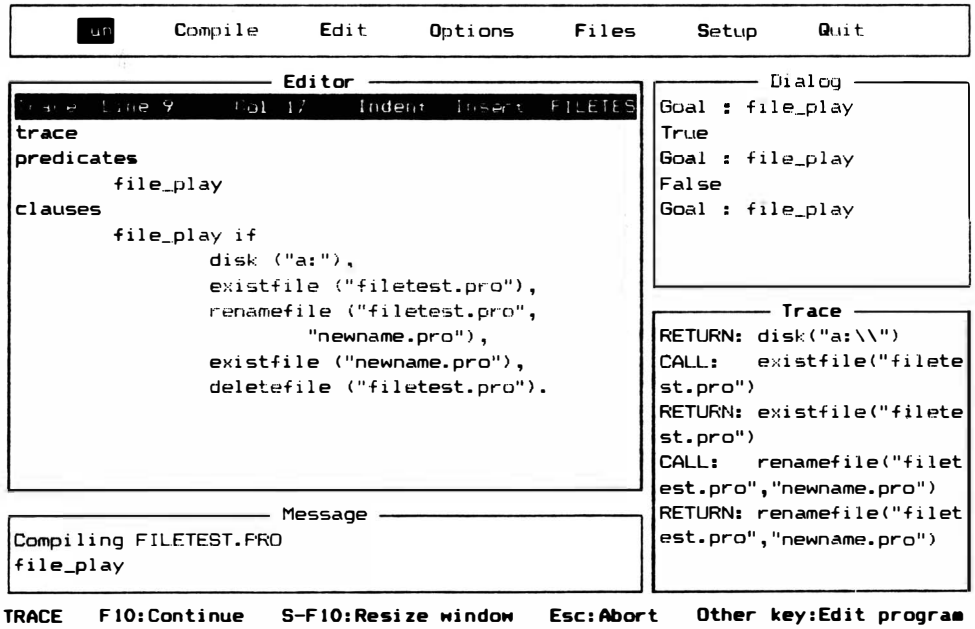


Figura 10-3 Resultado do acompanhamento de "file_play"

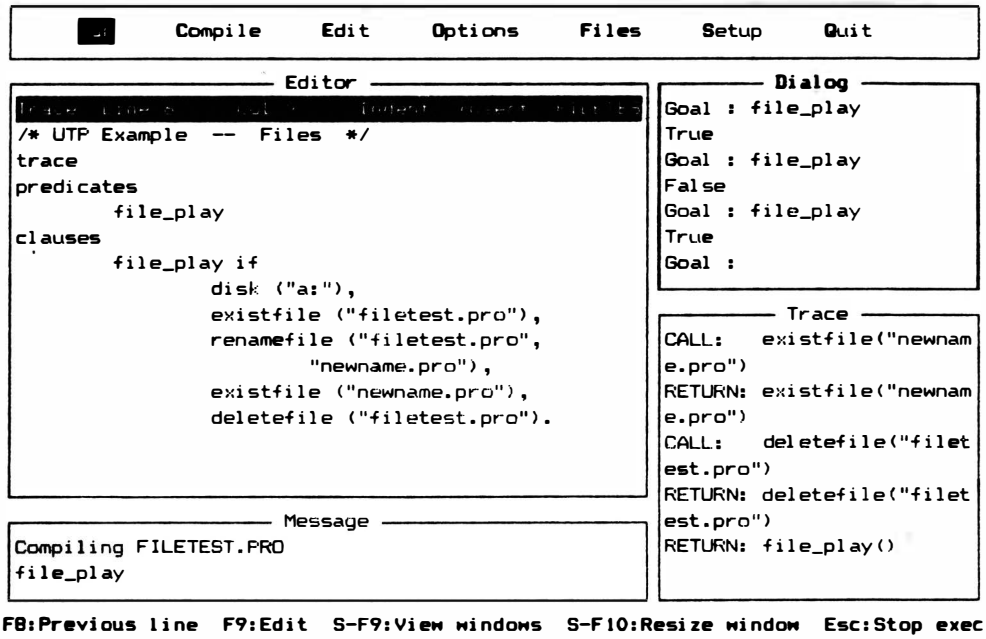


Figura 10-4 Parte do acompanhamento mostrando o funcionamento de "deletefile"

Caso você tente a mesma meta novamente neste ponto, obterá um resultado diferente. Tente. Depois comece a acompanhar o programa com a tecla F10. O predicado "disk" será novamente bem-sucedido, mas assim que Turbo Prolog chegar ao predicado "existfile", encontrará insucesso e responderá "False" na janela Dialog (veja Figura 10-5). Não existe mais um arquivo de nome "filetest.pro" no acionador A.

PREDICADOS PARA ABERTURA DE ARQUIVOS

Se você quiser trabalhar de alguma forma com um arquivo, que não seja apenas verificar sua existência ou alterar seu nome, precisará *abri-lo*. O predicado para abertura do arquivo oferece uma variedade de estilos para o procedimento, dependendo do que se desejar fazer com um arquivo. Oferecendo esta variedade, Turbo Prolog permite-lhe proteger os arquivos contra corrupções como alterações inadequadas em seu conteúdo. Por exemplo, em diversos casos você vai querer que alguma pessoa seja apenas capaz de ler o arquivo sem poder escrever nele. Neste caso, ele pode ser aberto apenas para leitura e não para escrita. Em outro momento você poderá desejar capturar os dados no arquivo

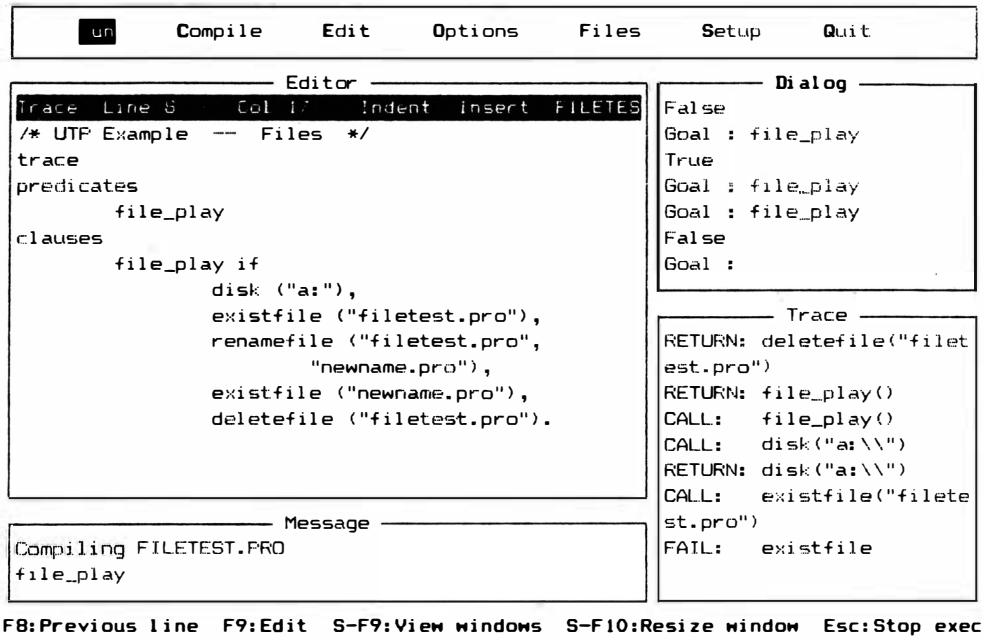


Figura 10-5 Resultados "falsos" mostrados na janela Dialog

mas poderá, ao mesmo tempo, não querer os dados que lá se encontram. Turbo Prolog permite que você abra o arquivo para fazer acréscimos, sem modificar ou escrever de nenhuma outra forma.

OPENWRITE

Antes de ler de um arquivo, você quer gravar um arquivo. Poderia apenas ler de um dos arquivos já existentes no disco. O programa da Figura 10-6 abre um arquivo denominado "clouds.tex" ["nuvens.tex"] e escreve os nomes de algumas nuvens nele. Durante a execução do programa, poderá ouvir o seu acionador de disco começar a funcionar enquanto abre o arquivo.

Depois de processar o programa e utilizar a meta externa "store_clouds" para executar sua solicitação, tecle ESC para encerrar a execução, escolha a opção "O" no submenu Files, e utilize o comando

```
type clouds.tex
```

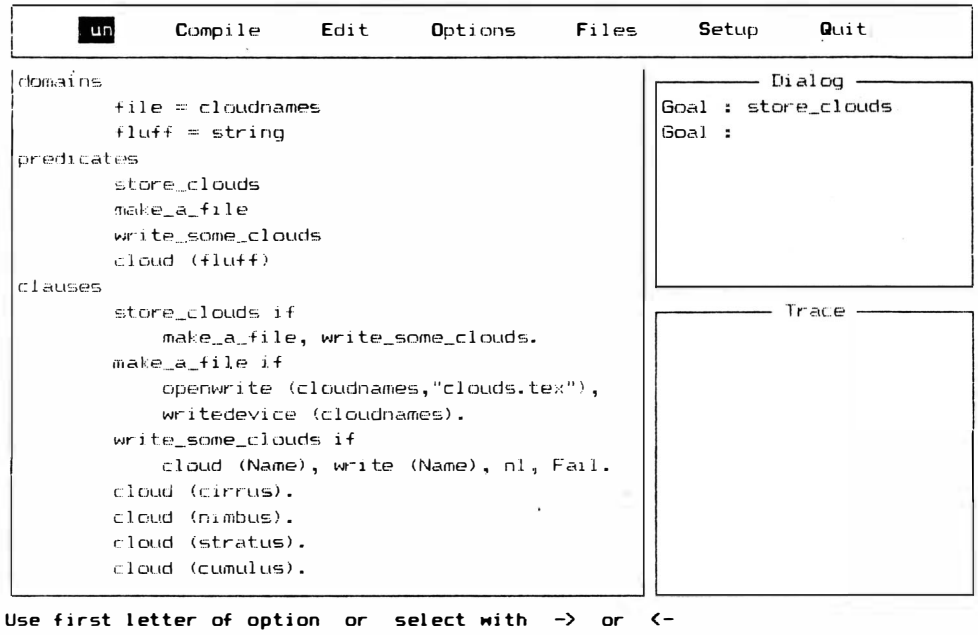


Figura 10-6 Abrindo um arquivo para escrever o nome de "clouds" [nuvens] nele

para ver o conteúdo do arquivo que você acabou de criar (conforme mostrado na Figura 10-7). O "False" no fim vem de "Fail" no predicado "write_some_clouds". Digite Exit e tecle RETURN para voltar ao Turbo Prolog. Se processar novamente o programa com um conjunto diferente de fatos de "clouds" [nuvens], encontrará apenas o novo arquivo "clouds.tex" ao verificar o disco. O arquivo antigo com o mesmo nome é automaticamente cancelado.

OPENREAD

Para ler novamente o arquivo "cloud.tex", poderá utilizar o programa mostrado na Figura 10-8. Este programa mostra tanto a leitura de *string* como de caractere, que depois são escritos na janela Dialog. Observe que a leitura de caractere não começa novamente no início do arquivo. Começa depois do retorno de carro que conclui a leitura da *string*.

```
Type EXIT to return from DOS


The IBM Personal Computer DOS
Version 2.00 (C)Copyright IBM Corp 1981, 1982, 1983

A>type clouds.tex
cirrus
nimbus
stratus
cumulus
False

A>
```

Figura 10-7 Conteúdo do arquivo "cloud.tex"

FILEPOS

Se você quiser especificar em que posição do arquivo ler, poderá utilizar este predicado. De fato, a posição poderá ser contada a partir do começo do arquivo, da posição atual, ou do fim do arquivo, do fim para o começo.

O Programa-Exemplo 44 no disco Biblioteca/Exemplo (mostrado na Figura 10-9) permite que você inspecione, de forma interativa, uma dada posição em qualquer arquivo. A Figura 10-10 mostra o programa de leitura de arquivo, anteriormente desenvolvido, sendo lido de um ponto que não é o começo do arquivo. O predicado "readln" começa a trabalhar na segunda posição (o segundo caractere) e pára de operar quando encontra um retorno de carro. O predicado "readchar" pula duas posições da atual do arquivo (que era o primeiro caractere da segunda *string*) e começa a trabalhar no meio daquela *string*.

O predicado "filepos" poderá também ser utilizado ao contrário (com uma variável "Number" não-instanciada) para descobrir, em relação ao início do arquivo (Modo O), qual é a posição atual do arquivo.

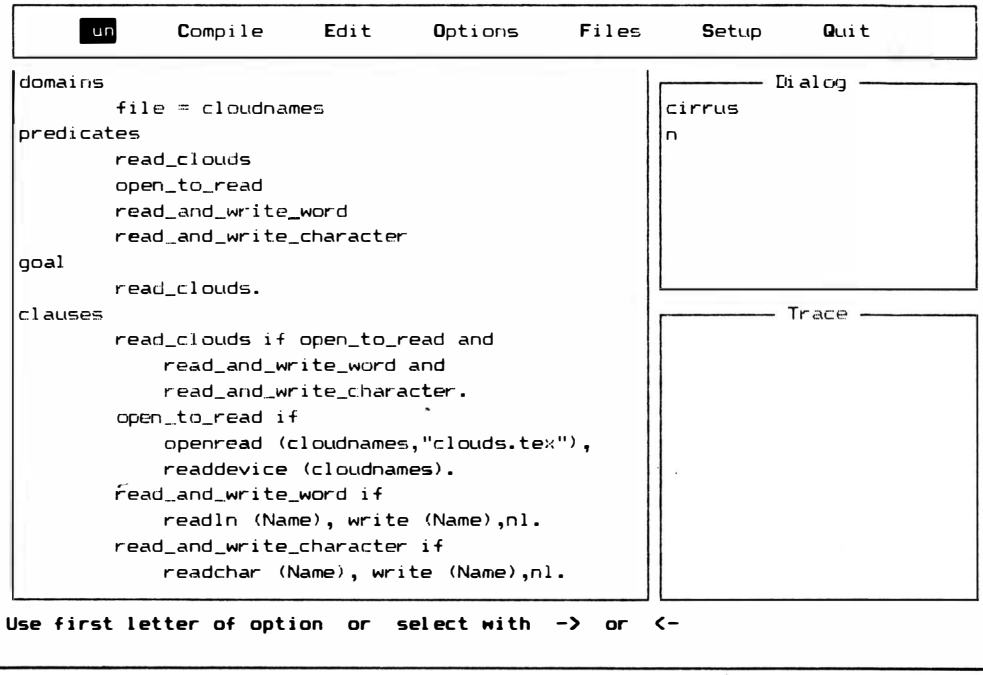


Figura 10-8 Colocando (lendo) de volta o arquivo "clouds.tex"

EOF

Este predicado simples será bem-sucedido se a posição atual do arquivo estiver no fim do arquivo. (O código que indica o fim do arquivo é um decimal ASCII 26. Muitos sistemas produzem este código quando você tecla CTRL-Z.) O predicado "eof" é útil pelo fato de você poder saber quando passou por todo o arquivo. A Figura 10-10 mostra este teste acrescentado ao programa de leitura de arquivo desenvolvido anteriormente neste capítulo. Uma das maneiras mais fáceis de utilizá-lo é encerrá-lo em um predicado "not" e torná-lo parte de uma série de cláusulas "and" em uma regra. A cláusula será bem-sucedida apenas se Turbo Prolog não está ainda no fim do arquivo.

FILE_STR

Isto é tanto um predicado de "string" quanto de arquivo. Lê caracteres de um arquivo até encontrar o código ASCII "eof".

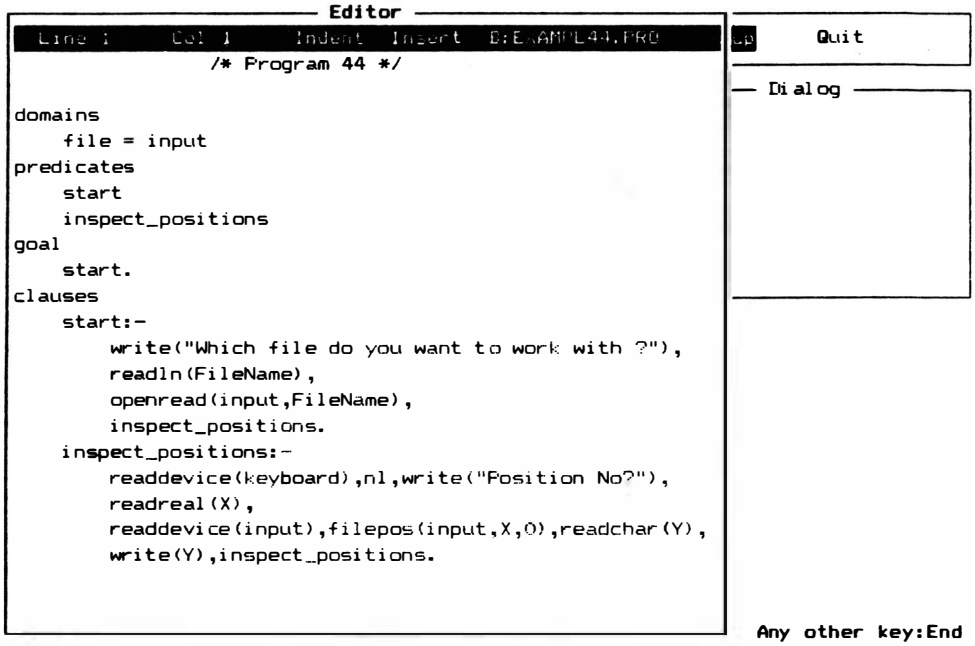


Figura 10-9 Programa-Exemplo 44

OPENMODIFY

Utilize o predicado "openmodify" se quer ler e escrever para o mesmo arquivo.

OPENAPPEND

Caso você queira acrescentar informação ao fim de um arquivo, poderá abri-lo com este predicado. Caso "openappend" não encontre o arquivo especificado. Turbo Prolog fornece-lhe uma mensagem de erro.

CLOSEFILE

Quando tiver terminado a modificação de um arquivo, deverá fechá-lo para ter certeza de que todas as informações chegaram a ele corretamente e que a informação estará ali da próxima vez que precisar dela. Sem o predicado "closefile", haverá o risco de alguma informação ser perdida porque o sistema não sabe que o arquivo está completo.

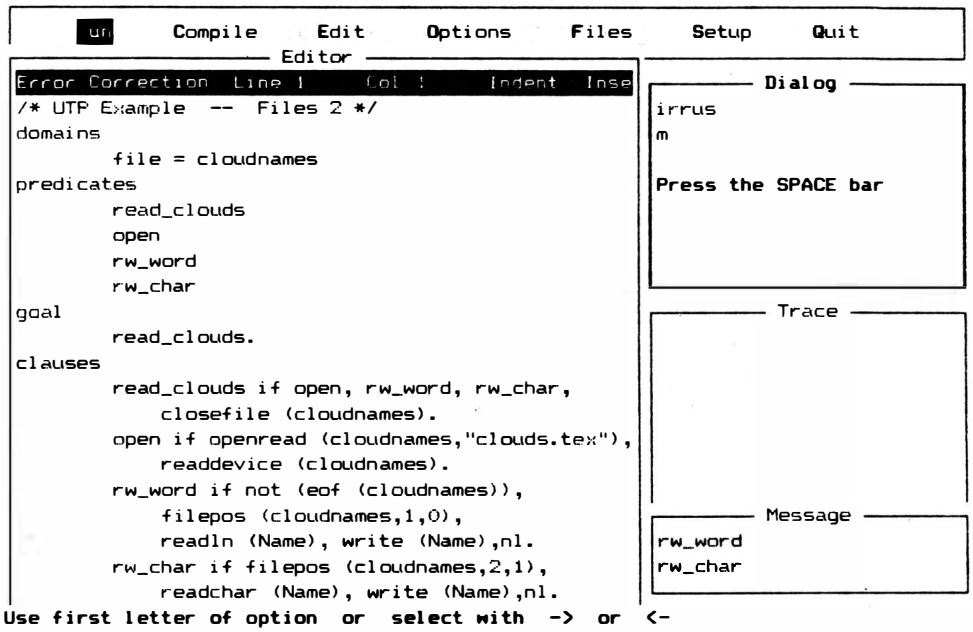


Figura 10-10 Lendo um arquivo a partir de um ponto diferente da origem

FLUSH

Quando seu programa estiver gravando informação, escreverá primeiro em um buffer interno. O predicado "flush" garantirá que todo o conteúdo daquele buffer será gravado na variável "SymbolicFileName". Provavelmente você não vai se utilizar disto com muita freqüência antes de começar a programação avançada.

PREDICADOS DOS

Turbo Prolog permite que você use diretamente os recursos do sitema operacional (DOS). No menu Files existem diversas opções que lhe permitem direcionar, cancelar, copiar, alterar o nome ou imprimir arquivos no disco sem sair do Turbo Prolog.

Adicionalmente, uma opção "Sistema Operacional" permite que você deixe Turbo Prolog temporariamente para executar operações DOS mais complexas. Caso você digite O para escolher esta opção verá que voltou ao sistema operacional com a indicação do acionador de disco na tela. Para voltar do DOS, não é necessário digitar a palavra "prolog" novamente para carregar Turbo Prolog – já está na memória. Tudo o que é preciso

fazer é digitar a palavra "exit" e teclar RETURN. Caso você digite a palavra "prolog", provavelmente obterá uma mensagem de erro informando-lhe de que não há espaço suficiente na memória para carregar o programa.

Turbo Prolog permite também que você chegue ao DOS a partir do programa. Os predicados mostrados na Tabela 10-5 são as ferramentas que lhe dão este recurso.

Tabela 10-5 Predicados-padrões relacionadas ao DOS

dir(Pathname,FileSpecString,DosFileName) **string,string,string–(I,I,O)**

Abre a janela de diretório Turbo Prolog e mostra os arquivos em "Pathname" e em "FileSpaceString" (sendo que ambos têm de estar instanciados ou ligados). O usuário do programa pode assim utilizar as teclas de cursor para iluminar um determinado arquivo, e teclar RETURN; o nome do arquivo será ligado a "DosFileName".

date(Year,Month,Day) **inteiro,inteiro,inteiro – (I,I,I,) (O,O,O)**

Existem duas maneiras de utilizar este predicado: obter a data ou modificá-la. Ambos atuam sobre o relógio interno do PC, que sempre inicia no momento em que o computador é ligado e pode ser modificado pelos comandos DOS externos ao Turbo Prolog. Alguns sistemas têm relógios permanentes que continuam a funcionar mesmo quando a máquina é desligada. O PC não avançado não tem esta opção.

Para obter a data, os três argumentos têm de estar não-instanciados. Cada um será ligado ao valor atual, e o predicado obterá sucesso.

Para modificar a data, os três argumentos têm de estar instanciados e seus valores serão enviados ao relógio.

time(Hours,Minutes,Seconds,Hundreths)
inteiro,inteiro,inteiro,inteiro – (I,I,I,I) (O,O,O,O)

Existem duas maneiras de utilizar este predicado: obter a hora ou modificá-la. Ambos agem sobre o relógio interno do PC, que sempre inicia no momento em que o computador é ligado, e pode ser modificado por comandos DOS externos ao Turbo Prolog. Alguns sistemas têm relógios permanentes que continuam funcionando mesmo quando a máquina está desligada. O PC não melhorado não tem este benfício.

Para obter a hora, os quatro argumentos têm de estar não-instanciados. Assim, cada um será ligado ao valor atual, e o predivado obterá sucesso.

Para modificar a hora, os quatro argumentos têm de estar instanciados, e seus valores serão enviados ao relógio.

beep

Este predicado não tem nenhum argumento. Faz soar o alto-falante do computador.

bios(InterruptNo,RegsIn,RegsOut) inteiro,regdom,regdom – (I,I,O)

Este predicado requer duas variáveis instanciadas (o número de "Interrupt" e os valores do registro a ser utilizados para esta interrupção) e vai ligar a terceira variável ao valor dos registradores depois da interrupção. Registradores são as áreas de armazenamento interno do microprocessador. O domínio "regdom" é definido internamente pelo Turbo Prolog e tem lugar para valores inteiros de todos os registradores:

regdom = reg(AX,BX,CX,DX,SI,DI,DS,ES)

membyte(Segment,Offset,Byte) inteiro,inteiro,inteiro – (I,I,I) (I,I,O)

Se todas as três variáveis estão ligadas, "membyte" vai armazenar o valor "Byte" no endereço de memória calculado com "Segment" e "Offset". Se apenas "Segment" e "Offset" estão ligados, "membyte" vai ligar "Byte" ao valor encontrado no endereço.

memword(Segment,Offset,Word) inteiro,inteiro,inteiro – (I,I,I) (I,I,O)

Se as três variáveis estão ligadas, "memword" vai armazenar o valor "Word" no endereço de memória calculado de "Segment" e "Offset". Se apenas "Segment" e "Offset" estão ligados, "memword" vai ligar "Word" ao valor encontrado no endereço.

portbyte(PortNo,Value) inteiro,inteiro – (I,I) (I,O)

Se tanto, "PortNo" como "Value" estão ligados, este predicado vai enviar (ou escrever) o inteiro "Value" à porta portNo. Caso "PortNo" esteja ligada e "Value" não, "portbyte" lerá um byte da porta dada e vai ligá-lo a "Value".

ptr_dword(StringVar,Segment,Offset)
string,inteiro,inteiro – (I,O,O) (O,I,I)

Se a única variável ligada das três é "StringVar", este predicado vai procurar a memória e devolver o endereço daquela *string*. O endereço vem em duas partes: "Segment" e "Offset". Caso variável estiver não-ligada e o endereço ligado, o predicado vai àquele endereço e ligar "StringVar" à *string* que encontrar lá. Os valores ASCII dos bytes encontrados, começando naquele endereço e processados até que O seja encontrado, serão traduzidos para caracteres e tornar-se-ão "StringVar".

storage(StackSize,HeapSize,TrailSize) real,real,real – (O,O,O)

Este predicado pode ler os tamanhos disponíveis correntes das variáveis.

system(DosCommandString) string – (I)

Envia o valor instanciado da variável ao DOS para execução.

DIR

O predicado "dir" (diretório) não vai diretamente ao DOS. Em seu lugar utiliza o diretório de arquivo do Turbo Prolog com o qual você está familiarizado no menu Files. O diretório do arquivo utiliza os recursos do diretório DOS. Pela especificação de um percurso, uma extensão de nome-de-arquivo (como .PRO), e um nome de arquivo, poderá ver os arquivos que estão no disco.

DATE E TIME

Os predicados "date" e "time" são os mais simples do grupo DOS. Cada um trabalha com o relógio do PC e cada um instancia a variável livre ao valor no relógio ou impõem valores ligados ao relógio. Todas as variáveis relevantes têm de ser instanciadas ou todas têm de estar livres antes de um destes predicados ser chamado. A Figura 10-11 mostra como estes e o predicado beep funcionam.

BEEP

Um predicado que compete muito para ser o predicado mais simples, "beep" apenas faz o alto-falante do computador emitir um som.

BIOS, MEMBYTE, MEMWORD, PORTBYTE, PTR_DWORD, STORAGE

Você não vai precisar destes predicados a não ser que esteja interessado em programar o computador ao nível do assembler. Você pode utilizar "bios" para inspecionar ou impor os valores dos registradores do microprocessador no coração do PC. Os predicados "membyte", "memword" e "portbyte" permitem que você leia ou escreva dados da memória ou porta de E/S. O predicado "ptr_dword" trabalhar com os endereços de segmento e offset de uma variável *string*, ao passo que storage lê os tamanhos das áreas stack, heap e trail.

SYSTEM

O predicado "system" envia uma *string* ao DOS para ser executado. É comando de uso geral do DOS a ser utilizado a partir dos programas Turbo Prolog. Por exemplo, se você tivesse um programa no acionador A chamado WORK.PRO e quisesse copiá-lo no acionador B, tudo o que teria de fazer seria acrescentar a cláusula

```
system ("copy a:work.pro b:")
```

ao seu programa; o resultado seria obtido durante a execução do programa. Você poderia utilizar também a cláusula

```
system (A)
```

se a variável "A" fosse instanciada no momento em que o predicado fosse chamado.

CONVERSÃO DE TIPO

Algumas vezes os dados com os quais você está trabalhando não estão no tipo de domínio necessário. Certos predicados (indicados na Tabela 10-6) convertem dados de um tipo para outro. A conversão entre os tipos de símbolo/*string* e tipos inteiro/real é automática nos predicados padrões.

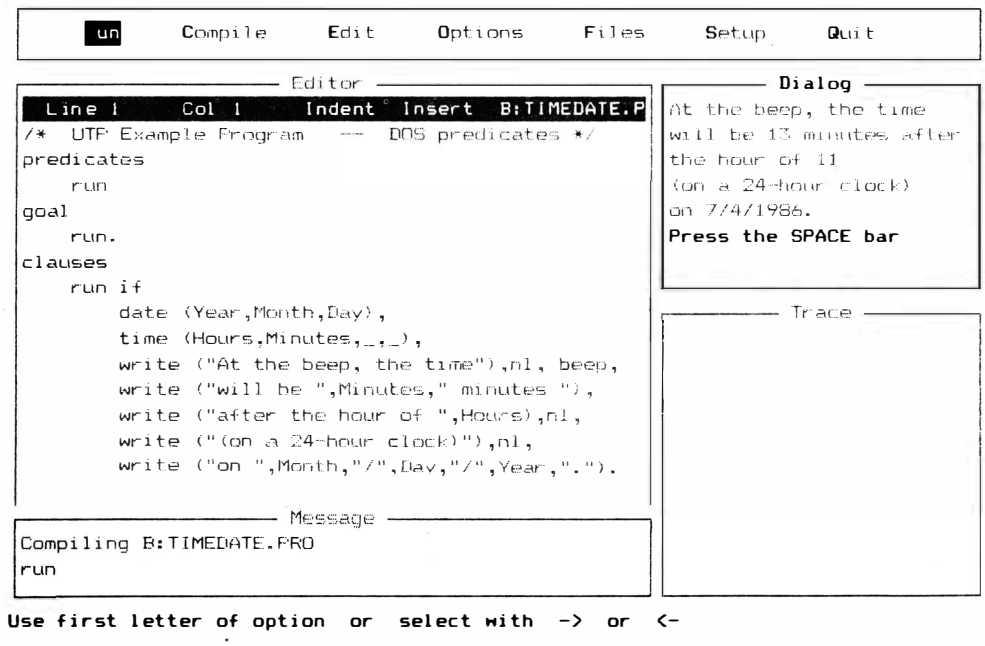


Figura 10-11 Os predicados "time", "date" e "beep" em operação

CHAR_INT (CHAR, PARAM, INTPARAM)

Se as duas variáveis estiverem ligadas, o predicado char_int será bem-sucedido quando os valores forem equivalentes; isto é, é bem-sucedido quando "IntParam" é o valor inteiro que é o código ASCII decimal (procure no Apêndice A o caractere em "CharParam"). Funciona respectivamente nos domínios "char" e "int". Caso uma das variáveis esteja não-ligada, ela será instanciada ao valor convertido da variável ligada.

Tabela 10-6 Predicado da conversão-de-tipo padrão

char_int
str_char
str_int
str_real
upper_lower

STR_CHAR (STRINGPARAM, CHARPARAM)

Caso as duas variáveis estejam ligadas, "str_char" obterá sucesso quando os valaores ligados forem equivalentes. Este predicado funciona nos domínios "string" e "char", respectivamente. "StringParam" deve estar ligado a um único caractere. Se uma das variáveis for não-ligada, a outra será ligada ao seu valor correspondente, convertido.

STR_INT (STRING, LENGTH)

Este predicado trabalha nos domínios "string" e "int". Se as duas variáveis estiverem ligadas, o predicado será bem-sucedido se "String" tiver "Length" caracteres de comprimento. Caso "Length" esteja não-ligado, será instanciado com o comprimento de "String".

STR_REAL (STRINGPARAM, REALPARAM)

Se as duas variáveis estiverem ligadas, o predicado str_real será bem-sucedido quando os valores forem equivalentes; isto é, será bem-sucedido quando "RealParam" for o equivalente binário do valor inteiro decimal ao qual "StringParam" estiver ligado. Funciona em "string" e "real", respectivamente. Caso uma das variáveis esteja não-ligada, ela será instanciada ao valor convertido da variável ligada.

UPPER_LOWER (STRINGIFUPPERCASE, STRINGINLOWERCASE)

Este predicado funciona com duas *strings* e é útil no processamento de linguagem natural. Se as duas variáveis *string* estiverem instanciadas, o predicado será bem-sucedido quando representam duas versões da mesma *string* diferindo apenas no uso de maiúsculas ou minúsculas. Caso uma das variáveis esteja não-ligada, será instanciada com a versão maiúscula da outra se esta for minúscula e vice-versa.

CAPÍTULO **11**

GRÁFICOS E SOM

GRÁFICOS

Os gráficos tornaram-se um elemento muito popular em muitas linguagens de programação. Turbo Prolog não tem um, mas dois métodos poderosos para a criação de gráficos em um programa.

O primeiro método consiste em utilizar os predicados de ponto e linha, que lhe permitem traçar linhas e pontos em qualquer uma das cores oferecidas pelo PC. O segundo método que você pode utilizar é denominado *gráfico da tartaruga* porque lhe permite utilizar predicados que movimentam uma "tartaruga" imaginária na tela. Se você já utilizou Logo, saberá tudo a respeito de gráficos da tartaruga.

Caso ainda não queira utilizar nenhum gráfico em seu programa, pode pular este capítulo. Entretanto, o capítulo é curto, e poderá valer a pena saber o que a linguagem oferece. Caso você compreenda Logo, poderá passar rapidamente pela seção sobre gráficos com tartaruga.

GRÁFICOS DE LINHAS E PONTOS

Os predicados padrões que lhe permitem criar gráficos de pontos e linhas em Turbo Prolog estão listados na Tabela 11-1. Dois dos quatro predicados listados ("graphics" e "text") servem apenas para estabelecer o cenário de gráficos e o seu cancelamento; você tem de se satisfazer com um único predicado para criação de pontos e um único predicado para criação-de-linha.

Construindo regras próprias Uma regra geral que você deveria ter em mente

a respeito do Turbo Prolog, e a respeito da maioria das linguagens de programação, é que não precisa ter um comando para cada situação. De fato, uma linguagem pode ser lenta e extremamente difícil de ser aprendida caso esteja carregada com comandos especializados que podem ter utilidade apenas uma vez ou outra.

Tudo o que você precisa para o comando de gráficos de uma linguagem é a chave que vai em cada porta. A partir dos predicados intrínsecos, você pode construir rotinas e regras mais complexas. Em Turbo Prolog, você pode fazer os predicados de ponto e de linha servirem como cláusulas em regras maiores que podem executar exatamente o que você precisa. Depois, pode gravar estas regras para serem usadas em outros programas.

Tabela 11-1 Predicados padrões para gráficos com pontos e linhas

graphics (ModeParam, Palette, Background)
text
dot (Row, Column, Color)
line (Row 1, Col 1, Row 2, Col 2, Color)

Utilizando o modo gráfico Para começar a fazer gráficos, é preciso que o Turbo Prolog e o PC estejam no modo gráfico. O predicado "graphics" (ModeParam, Palette, Background) serve para fazer isto. Ele limpa a tela, coloca o cursor no canto superior esquerdo, estabelece a resolução, escolhe as cores de primeiro plano disponíveis e estabelece as cores de fundo. Mas não desenha nada. Você utiliza os predicados "dot" e "line" para fazer isto. Ao terminar de fazer gráficos e querendo voltar ao modo texto, basta usar o predicado "text". Este predicado sem argumentos recoloca a tela no modo texto. (Caso o utilize durante o modo texto, não faz absolutamente nada.)

A primeira coisa a fazer com o predicado "graphics" é certificar-se de que os três argumentos estão instanciados no ponto do programa em que será chamado. Você pode colocar constantes nos argumentos explicitamente, ou pode montar estruturas que estabeleçam os valores certos nos momentos certos. Você instancia estes valores da mesma forma que faria com outros em Turbo Prolog. A Figura 11-1 mostra o predicado "graphics" estabelecendo o modo gráfico. Você deveria, agora, examinar cada um dos argumentos do predicado "graphics".

ModeParam O PC da IBM oferece diversos estilos de tela. O monitor monocromático original oferece 80 colunas por 25 linhas de caracteres em preto e branco (ou âmbar ou verde) utilizados até agora neste livro. O conjunto de caracteres incluem algumas formas gráficas para realizar desenhos rudimentares. Existem ainda placas para monitores monocromáticos alternativos (placas de circuito impresso diferentes) que oferecem letras de forma diferente e resolução melhor que o equipamento IBM original. Muitos

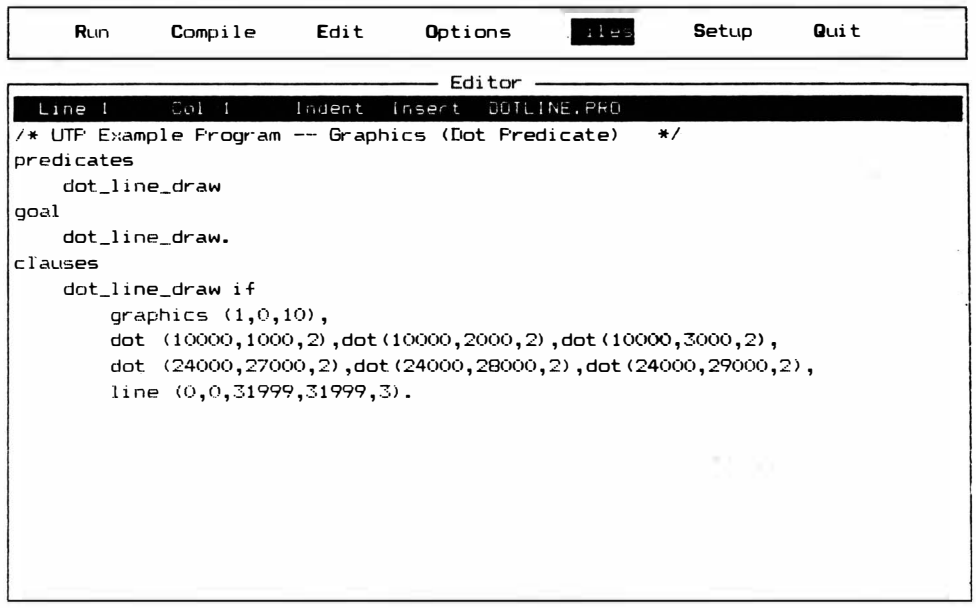


Figura 11-1 O predicado "dot"

computadores compatíveis com o PC da IBM têm este tipo de placa de vídeo inerentes à máquina.

Ligando outras placas para vídeo, será possível acrescentar até cor à sua tela (apesar de precisar também de um monitor colorido para ver uma imagem colorida). Os padrões de vídeo colorido mais populares da IBM são o CGA (*Color Graphics Adapter*) e o EGA (*Enhanced Graphics Adapter*). Muitas outras empresas fazem adaptadores para vídeo que imitam estes, e alguns fazem placas baseadas em outros padrões. Tanto o CGA como o EGA têm uma variedade de modos gráficos (isto é, uma única placa tem condições de fornecer diferentes estilos de vídeo). A razão principal desta multiplicação de estilos é que quando você estabelece um vídeo colorido, precisa colocar os números das cores na paleta (as cores que poderá escolher para desenhar) *versus* a resolução (o número de pontos na tela).

Turbo Prolog trabalha com alguns dos modos comuns tanto de CGA como de EGA. A Tabela 11-2 mostra estes modos juntamente com a sua resolução e número de cores.

Tabela 11-2 Modos gráficos e valores "ModeParam"

Valor ModeParam	Descrição	Resolução Colunas	Linhas	Número das Cores	CGA ou EGA
1	médio	320	200	4	CGA
2	alto	640	200	B/P*	CGA
3	médio	320	200	16	EGA
4	alto	640	200	16	EGA
5	melhorado	640	350	13	EGA

* Preto e branco, ou monocromático

Palette Como você pode ver na Tabela 11-2, o CGA permite apenas os modos 1 e 2. Você pode ter um vídeo de alta resolução em preto e branco ou pode ter quatro cores. O modo 1 do CGA permite que voce utilize quatro cores, mas uma destas cores está reservada para o fundo. Para as três cores do primeiro plano, existe uma escolha entre duas paletas, conforme indicado na Tabela 11-3.

Tabela 11-3 Escolhas de paleta de resolução média de CGA (Cores de primeiro plano)

Valor da Paleta	Cor 1	Cor 2	Cor 3
0	verde	vermelho	amarelo
1	verde azulado	magenta	branco

Segundo Plano Assim que tiver escolhido um modo gráfico (para o argumento "ModeParam") e cores (para o argumento "Paleta"), você precisa selecionar a cor de fundo. Isto não é tratado por "palette". A Tabela 11-4 mostra as possibilidades para as cores do segundo plano. Você sempre pode escolher entre 16 cores para isto, e os valores são os mesmos vistos para os predicados de janelas no Capítulo 7.

FAZENDO ALGUMAS MARCAS

Assim que você tiver passado para o modo gráfico e escolhido as cores, estará pronto para realmente colocar alguma coisa na tela. Suas duas ferramentas iniciais são os predicados "dot" e "line".

dot (Row, Column, Color) A utilização do predicado é bastante direta. Este predicado pode funcionar de duas formas. Pode colocar um ponto de uma determinada cor na tela, ou pode ler o valor da cor no ponto indicado por "Row" e "Column". Não lerá a cor do ponto aqui, porque o ponto poderá ter a mesma cor do fundo e poderá não parecer um ponto para você. Mesmo assim, todo ponto na tela tem um valor de cor.

Tabela 11-4 Escolha das cores de fundo para o predicado "Graphics"

Valor	Cor	Valor	Cor
0	Preto	8	Cinza
1	Azul	9	Azul claro
2	Verde	10	Verde claro
3	Verde azulado	11	Verde azulado claro
4	Vermelho	12	Vermelho claro
5	Magenta	13	Magenta claro
6	Marron	14	Amarelo
7	Branco	15	Branco intenso

Para traçar um ponto com um dado valor "Color", você precisa instanciar os três argumentos do predicado. "Row" e "Column" colocam o ponto em uma grade imaginária da tela. Esta grade tem o ponto zero no canto superior esquerdo. Agora, você tem a oportunidade de digitar qualquer valor inteiro de coordenada de 0 até 31999 para cada argumento, não importa em que modo de vídeo você se encontra. Você realmente vai colocar um ponto apenas se ele tiver um valor de cor diferente do fundo.

Para ler o valor da cor de um ponto na tela, instacie "Row" e "Column" para valores legítimos (0 até 31999). O predicado "dot" obterá sucesso ligando "Color" ao valor de cor apropriado. O programa da Figura 11-1 mostrará as duas utilizações de "dot" (bem como os predicados "graphics" e "text") em ação. A Figura 11-2 mostra o resultado em preto e branco daquela utilização. Os pontos estão nos setores superior esquerdo e inferior direito da tela.

line (Row 1, Col 1, Row 2, Col 2, Color) Ao contrário de "dot", o predicado "line" poderá ser utilizado apenas para traçar na tela, e não para ler o valor de uma cor na tela. Isto é compreensível, já que uma linha não é como um ponto, pelo fato de não ter necessariamente um único valor de cor.

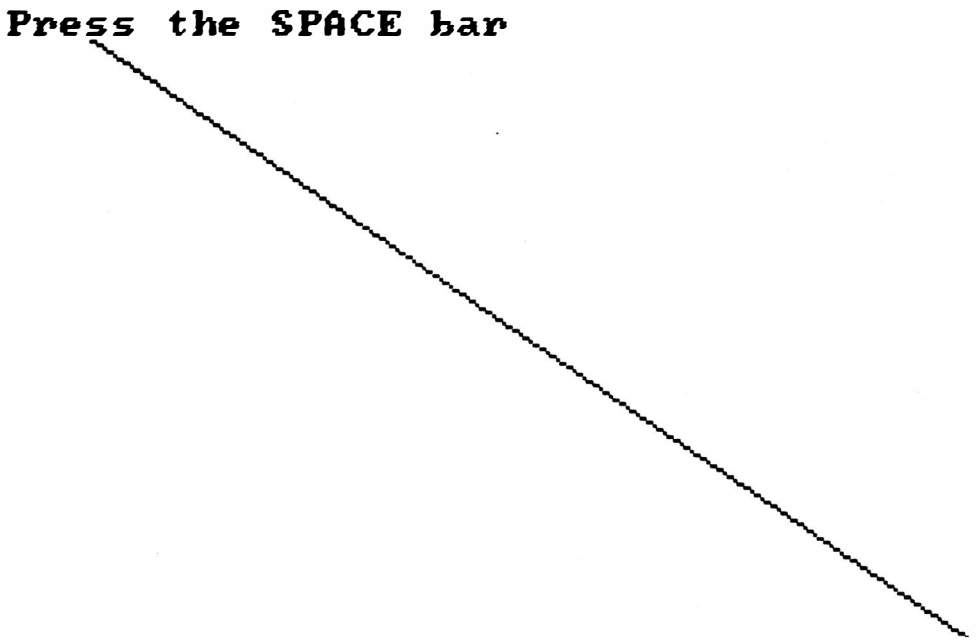


Figura 11-2 Resultado da utilização dos predicados "graphics" e "text"

O predicado "line" precisa de cinco variáveis instanciadas. Quatro são especificações de linhas e colunas que estabelecem as extremidades da linha. A última estabelece a cor da linha. Para ser bem-sucedido, o predicado traça uma linha entre (e incluindo) as duas extremidades. Novamente, da mesma forma que com o predicado "dot", os valores legítimos "Row" e "Column" variam de 0 até 31999. O programa na Figuraa 11-1 também usa o predicado "line". A Figura 11-2 mostra o resultado do processamento do programa.

Os modos gráficos não impedem que você utilize o predicado "write" que aprendeu. A Figura 11-3 mostra alguma escrita acrescentada a um programa gráfico. A Figura 11-4 mostra o resultado.

Caso você tenha uma placa gráfica Hercules ou um adaptador de vídeo monocromático Hercules, não poderá utilizar os predicados gráficos padrões – Turbo Prolog não os permitirá naquele hardware.

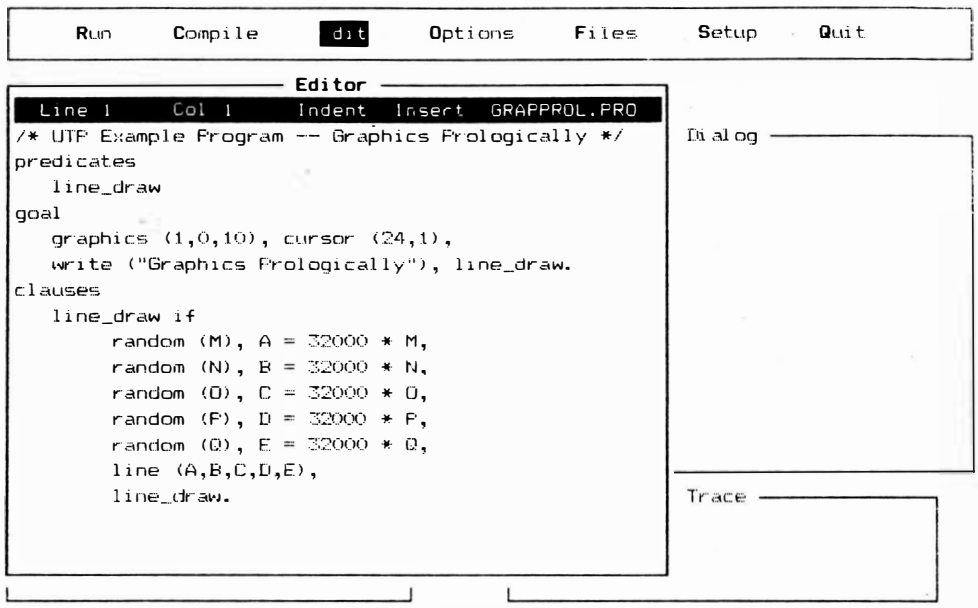


Figura 11-3 "Write" acrescentado a um programa gráfico que traça linhas aleatórias

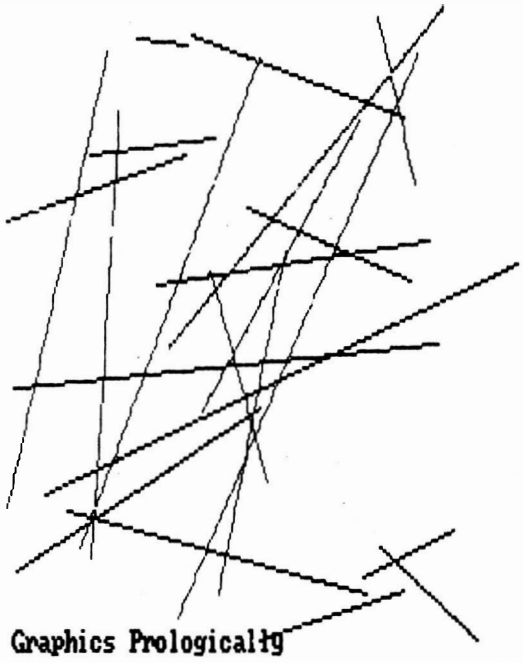


Figura 11-4 Resultado do programa da Figura 11-3

CONSTRUA AS SUAS PRÓPRIAS FUNÇÕES GRÁFICAS

Você pode criar regras gráficas mais complexas a partir dos dois comandos gráficos fundamentais. Da mesma forma, você pode utilizar os predicados matemáticos e operadores em conjunto com os dois comandos gráficos básicos para criar regras que terão sucesso ao traçar qualquer outra forma. Outras regras faça-o-você-mesmo podem executar funções como o preenchimento de uma determinada área.

GRÁFICOS EM JANELAS

Você pode colocar gráficos em janelas também. De fato, você pode colocar gráficos em um número de janelas no mesmo programa, bem como na tela.

GRÁFICOS TARTARUGA

Gráficos tartaruga é o outro método principal para a colocação de linhas e figuras na tela em Turbo Prolog. A idéia de uma "tartaruga" para a geração de gráficos representa os fundamentos da linguagem de computação Logo e foi adotada por várias outras linguagens.

A "tartaruga" é um *robot* imaginário para traçados na tela. Ele pega suas instruções do programa na forma de predicados, conforme a Tabela 11-5. Os predicados lhe informam se deve se movimentar para a frente ou para trás, virar à esquerda ou à direita,

Tabela 11-5 Predicados gráficos da tartaruga

Controle de pena	
pencolor (Color)	inteiro – (I)
pendown	
penup	
Movimento	
forward (Step)	(inteiro) – (I)
back (Step)	(Inteiro) – (I)
Rotação	
left (Angle)	(inteiro) – (I) ou (O)
right (Angle)	(inteiro) – (I) ou (O)

pressionar uma pena (contra a tela, como se a tela fosse papel de desenho) ou levantar a pena. A tartaruga do Turbo Prolog é ainda mais talentosa do que isto. Pode ainda obedecer instruções referentes à cor de pena que deve utilizar.

Dando vida à tartaruga Quando você entra no modo gráfico, chamando o predicado "graphics", a tartaruga está parada no centro da tela (centralizada horizontal e verticalmente), com a pena levantada e o nariz apontado para a parte superior da tela. A tartaruga nunca será vista. A única coisa que você verá será trilha deixada pelo seu lápis. Caso você opte por utilizar apenas os predicados "dot" e "line", não verá nem o rastro da tartaruga. Mas você pode misturar gráficos da tartaruga com gráficos de pontos e linhas se desejar.

A tartaruga espera suas ordens. A Figura 11-5 mostra um programa que dá à tartaruga algo a fazer.

Controle da pena O primeiro passo depois de entrar no modo gráfico neste programa é abaixar a pena. Se você quisesse uma imagem gráfica que começasse em outro ponto que não fosse o centro da tela, poderia mover primeiro a tartaruga para outra posição inicial

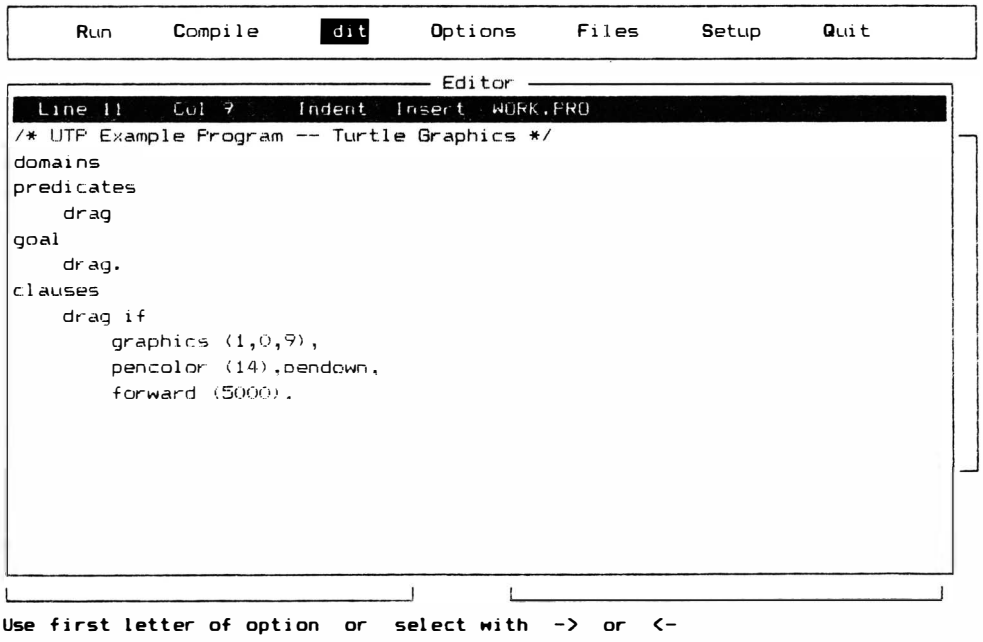


Figura 11-5 Trabalhando com a tartaruga

(com a pena para cima), abaixar a pena e traçar. Mas neste programa, a tartaruga começa a traçar imediatamente. Antes dela começar, você deveria estabelecer a cor que o desenho deveria utilizar.

pencolor (Cor) O predicado "Epencolor" atribui à cor da pena utilizada pela tartaruga o valor anteriormente instanciado de "Color". Para o modo O de CGA (que é o que foi estabelecido no programa-exemplo), as escolhas de cor estão limitadas pelas palettes aos valores inteiros mostrados na Tabela 11-3.

pendown Depois de escolher a cor da pena, você ordena à tartaruga abaixar a pena sobre a tela. O predicado "pendown" (sem argumentos) é satisfeito colocando o sistema de tal forma que movimentos subseqüentes pela tartaruga deixam um rastro da cor da pena escolhida na tela.

penup O predicado "penup" também não tem argumentos. Qualquer movimento da tartaruga, feito depois que este predicado foi chamado, não deixará nenhum rastro na tela.

forward (Step), back (Step) Neste ponto do programa, a tartaruga está pronta para o processamento (o modo gráfico foi determinado com o predicado "graphics") com uma determinada pena de cor (do predicado "pencolor") e com a pena encostada na tela (do predicado "pendown"). Tudo o que falta é movimentar a tartaruga para ver como desenha. Os dois predicados de movimento são "forward" e "backward". A direção é no sentido para o qual o nariz da tartaruga está apontando. Inicialmente está apontando para a parte superior da tela. Posteriormente, você aprenderá como alterar esta direção.

Cada predicado e movimento requer uma única variável instanciada: "Step". O valor de "Step" pode ser qualquer inteiro de 1 até 32000. O tamanho de cada passo depende do modo gráfico. Caso um passo para a frente ou para trás tire a tartaruga da tela, não será permitido; o traço não será feito e o predicado falhará.

A Figura 11-5 mostra um programa que traça uma linha reta, utilizando os predicados "pendown" e "forward" . A Figura 11-6 mostra os efeitos gráficos daquele programa.

left (Angle), right (Angle) Você logo vai se cansar de fazer linhas retas. Os predicados de rotação "left" e "right" fornecem mais possibilidades. Cada um deles pode funcionar de duas formas: girando a tartaruga ou lendo o seu ângulo.

Caso "Angle" esteja instanciado (com um inteiro), a tartaruga será girada deste número de graus na direção indicada. O predicado "right" gira a tartaruga em sentido

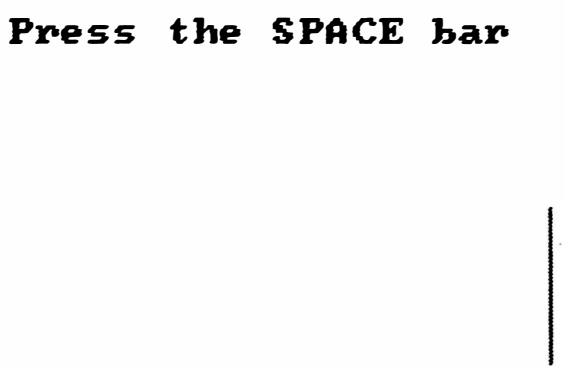


Figura 11-6 Efeitos gráficos de programa, utilizando predicados "pendown" e "forward"

horário; "left" gira a tartaruga em sentido anti-horário. Qualquer rotação que tenha mais do que 360 graus simplesmente movimenta a tartaruga mais do que uma volta. Depois da rotação, qualquer movimento será feito ao longo da linha da nova direção do nariz.

Caso "Angle" esteja livre, o predicado "left" ou "right" va ligá-lo ao ângulo atual. Este ângulo é medido relativamente a um ponto de origem diretamente para cima. Por exemplo, um valor de 180 para "Angle" significaria que o nariz aponta para baixo em linha reta no sentido da parte inferior da tela.

A Figura 11-7 mostra o programa da tartaruga com o acréscimo de diversas voltas. A Figura 11-8 mostra os efeitos gráficos do programa. Observe que este programa gráfico mostra também um texto gerado com o predicado "Write" padrão. A posição do cursor para escrever não é a mesma que a posição atual da tartaruga.

SONS

Você pode empregar som em seus programas Turbo Prolog, apesar do próprio hardware do PC da IBM não estar projetado para a produção de música particularmente complexa ou interessante. Porém o PC tem um alto-falante, e Turbo Prolog é capaz de exercer algum controle sobre o alto-falante. De fato, pode realmente tocar notas por intermédio dele. Caso você não queira utilizar a habilidade sonora de Prolog, ou se o predicado "beep", que é descrito no Capítulo 10, seja-lhe suficiente em termos sonoros, não leia o restante deste capítulo.

DOIS PREDICADOS

Os dois únicos predicados padrões que emitem som são "beep" e "sound". O

```
Run     Compile     dit     Options     Files     Setup     Quit

                        Editor
 Line 1     Col 1     Indent  Insert  TURTLES2.PRO
/* UTP Example Program -- Turtle Graphics  */
predicates
    drag
goal
    drag.
clauses
    drag if
        graphics (1,0,9), cursor (0,24), write ("Text and Turtles"),
        pencolor (14),
        pendown,
        forward (5000), right (90),forward (2000),right (90), forward (2000),
        penup,
        left (90), back (4000), pencolor (13), right (30),
        pendown,
        back (10000),
        penup,
        forward (15000),
        pendown,
        forward (10000).
```

Figura 11-7 Programa da tartaruga com diversas voltas.

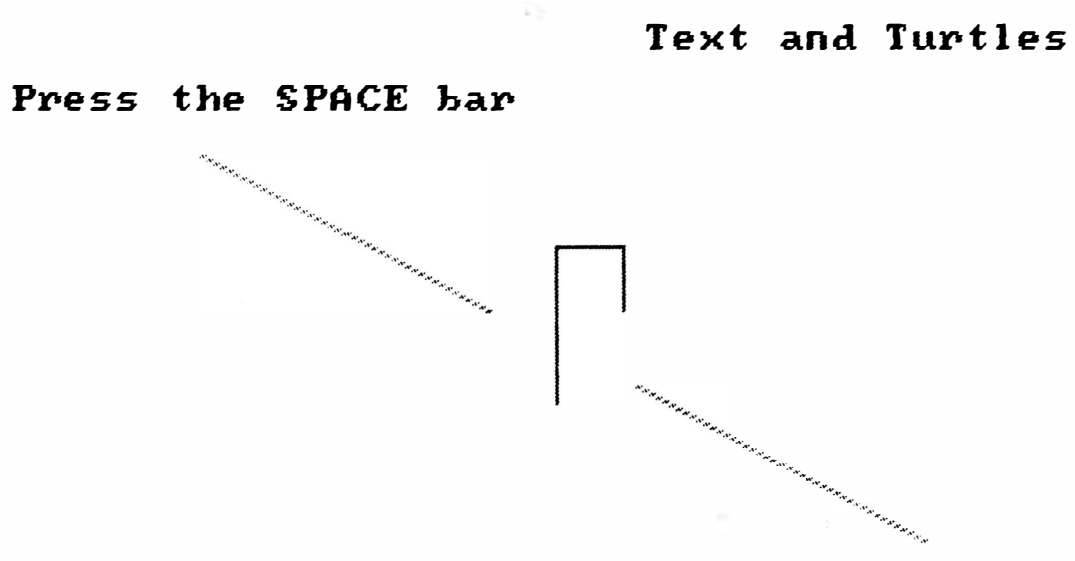


Figura 11-8 Efeitos gráficos do programa da tartaruga

predicado "beep" não lhe permite muito controle realmente. O predicado "sound" lhe permite um controle direto da duração e altura da nota.

beep Este predicado apenas emite um som rápido do tipo que muitos programas fazem quando você pressiona a tecla errada. Provavelmente a utilização mais freqüente do beep seja justamente isto: avisar erros de entrada. O predicado beep é utilizado no programa na Figura 11-9.

sound (Duration, Frequency) [Duração, Freqüência] Caso você queira realmente tocar notas, este é o predicado que deve ser utilizado. Para utilizar "sound" você deve instanciar tanto "Duration" como "Frequency" com valores inteiros (ligados).

O valor "Duration" será interpretado como um centésimo de segundo. O valor "Frequency" será interpretado como Hz (ciclos por segundo). Você pode ir desde 41 Hz até 17500Hz. Uma pequena modificação poderia facilmente transformar este código em um programa rudimentar de teste auditivo.

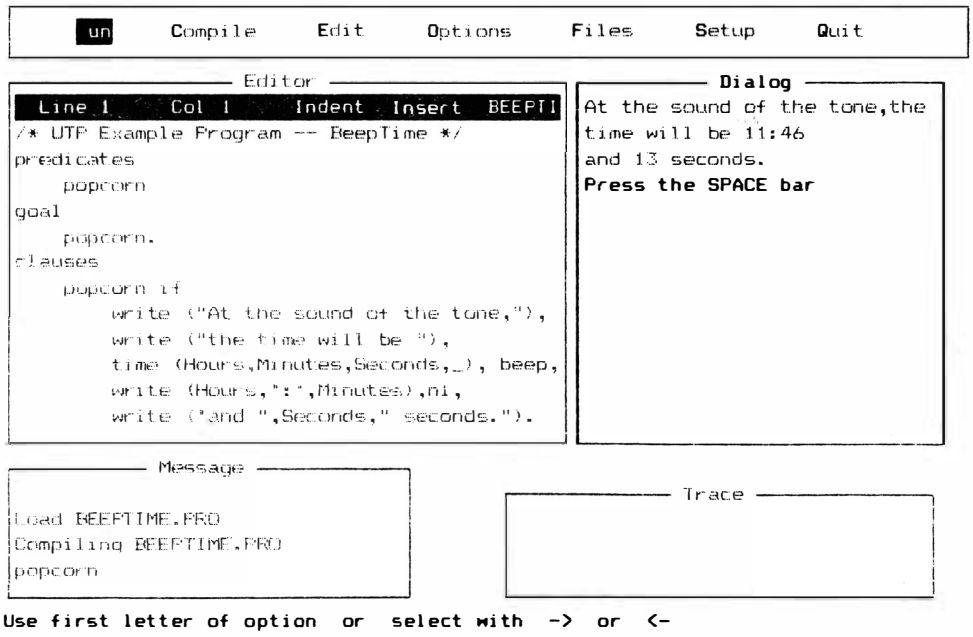


Figura 11-9 Um programaa utilizando o predicado "beep"

A capacidade de especificar a freqüência em Hz significa que você pode utilizar tabelas padrões das freqüências das escalas diatônicas para produzir música. A Figura 11-10 mostra o predicado do "sound" em funcionamento. Você pode expandir este programa para tocar *string* ou arquivos inteiros. Para fazer isto, poderá querer utilizar alguns dos predicados de conversão de tipo.

REGRAS SONORAS DEFINIDAS PELO USUÁRIO

Como em qualquer outra esfera de operação do Turbo Prolog, você pode construir regras que se comportam como predicados feitos sob medida. Para a geração de som, poderia utilizar o predicado "sound" e outros predicados padrões como "time", os diversos predicados "read", e os predicados matemáticos para a construção de regras sonoras mais complexas.

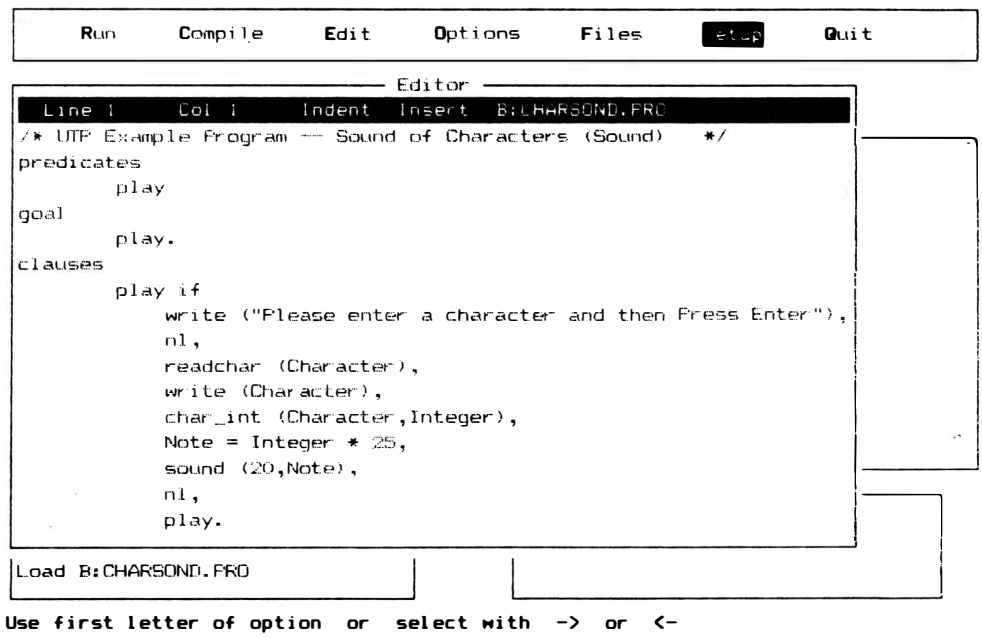


Figura 11-10 Um programa utilizando o predicado "sound"

CAPÍTULO 12

TÉCNICAS AVANÇADAS DE PROGRAMAÇÃO

Turbo Prolog fornece uma quantidade de recursos que lhe permitem construir programas mais complexos do que os mostrados até aqui neste livro. Este capítulo vai descrever rapidamente estes recursos e mostrar alguns deles em ação. Caso queira maiores detalhes, leia as descrições técnicas na seção "Guia do Programador" do manual da Borland, verifique um dos livros mais avançados a respeito de Turbo Prolog e dê uma olhada em GeoBase (descrito no Capítulo 13). GeoBase utiliza algumas das técnicas descritas aqui.

Não existe a "melhor maneira" para estruturar este tipo de tópicos avançados. Com freqüência referem-se um ao outro, e a maioria é opcional em qualquer programa dado. Este capítulo apresenta primeiro as diretrizes do compilador para lhe dar uma sensação de controle sobre os detalhes de compilação; depois, examina as outras maneiras de compilar um programa: em disco, com arquivos "include" ou em módulos.

Este capítulo é útil para duas finalidades: compreender melhor GeoBase e escrever programas mais complexos. Caso apenas deseje brincar com GeoBase antes de compreender seu funcionamento interno, então vá para o Capítulo 13. Se você não planeja escrever longos programas mas quer experimentar os predicados e operadores básicos, não precisa deste capítulo ainda. Mas a informação que ele contém ao menos permitirá que você veja o que mais Turbo Prolog pode oferecer e as possibilidades do que ficou para ser explorado.

DIRETRIZES DE COMPILAÇÃO

Quando vê tenta compilar um programa, Turbo Prolog verifica primeiro se o código de programa segue as regras de sintaxe da linguagem. Depois, verifica se os argu-

mentos usam corretamente seus domínios. Turbo Prolog é um compilador Prolog, mas o seu sistema de menus e janelas permite o desenvolvimento de programas de forma interativa. Cada vez que Turbo Prolog encontra um erro de sintaxe ou domínio, ele aponta este erro com o cursor (na janela de Editor), fornece-lhe uma mensagem explicando o problema, e coloca-o de volta no modo Editor de modo que você possa acertar o problema. Depois de cada acerto, basta teclar F10 novamente para tentar a compilação.

Assim que você tiver passado pela verificação de sintaxe e domínio, terá um programa que pode ser compilado. Se vai fazer o que se espera dele, quando quiser, é outro assunto. Caso tenha uma meta interna, você brevemente verá alguns sinais externos de seu progresso. Mas mesmo assim não pode ter certeza sobre o que o código vai fazer se mudar a meta interna. Adicionalmente, caso o programa dependa de uma meta externa, será preciso fornecer metas de teste imediatamente. Qualquer tipo de teste deveria incluir tantos casos padrões como extremos. Esta é uma prática regular em programação com qualquer linguagem.

Enquanto está estudando o programa, para ver o que faz e por que, poderá recorrer às diversas diretrizes do compilador. São instruções (relacionadas na Tabela 12-1) que você acrescenta ao arquivo do programa-fonte para informar ao compilador como tratar a compilação. Algumas diretrizes permitem compilações complexas, de multiarquivos, e outras apenas ajudam a tentar descobrir o que um programa está fazendo. Eles não são predicados. São instruções ao sistema Turbo Prolog.

SEGUINDO DIRETRIZES E PREDICADOS

Caso seu programa não esteja respondendo às suas metas de forma esperada, poderá utilizar o acompanhamento para descobrir o que está fazendo e por que. A maneira mais simples de fazer isto é acrescentar uma diretriz "trace" ao arquivo do programa-fonte. Mas esta não é a única maneira de se conseguir um acompanhamento. Existem outras opções, conforme mostrado na Tabela 12-2.

ACOMPANHAMENTO

Você já viu a utilização, neste livro, da diretriz "trace" básica. Esta palavra simples encontra-se no começo de um programa (depois de um comentário, mas antes da seção domínio). Quando esta diretriz é acrescentada, o programa é examinado passo a passo, chamando um predicado, fazendo uma combinação ou escrevendo uma linha por vez. No final de cada passo, ele pára, descreve o que foi feito, e aguarda que você lhe dê a próxima instrução. A Tabela 12-3 mostra os diversos tipos de mensagens que você obtém da diretriz "trace".

Tabela 12-1 Diretrizes de Compilação

check_cmpio	Verifica se algum predicado utiliza padrões de fluxo composto.
check_determ	Verifica se algum predicado tem cláusulas não-determinísticas.
code=nnnnn	Estabelece o tamanho máximo na memória para o código. A contagem é feita em parágrafos. Cada parágrafo equivale a 16 bytes. O tamanho default é 1k parágrafos (o valor nnnnn é igual a 1.024, neste caso.)
diagnostics	Imprime dados de diagnóstico do compilador.
include"filename"	Permite que você compile programas que se estendem por diversos arquivos. O arquivo nomeado será incluído na compilação e poderá ele próprio conter uma diretriz "include".
nobreak	Periodicamente, Turbo Prolog varre o teclado verificando se alguma tecla foi operada. Isto permite que você utilize CTRL-BREAK para escapar de programas que poderão ter entrado em ciclos infinitos. Esta diretriz elimina este tipo de verificação. Utilize-a e poderá ser confrontado com um programa que apenas pode ser interrompido, recomeçando o DOS.
nowarnings	Impede que Turbo Prolog lhe envie avisos.
shorttrace	Veja Tabela 13-2.
shorttrace p1, p2...	Veja Tabela 13-2.
trace	Veja Tabela 13-2.
trace, p1, p2...	Veja Tabela 13-2.
trail=nnn	Especifica o tamanho da área "trail" (em bytes). A área "trail" é a área utilizada para registrar a ligação e não-ligação de variáveis de referência. O tamanho padrão é O.

Tabela 12-2 Diretrizes de acompanhamento, operadores e predicados

trace	Liga o acompanhamento para todo o programa.
trace p1, p2...	Liga o acompanhamento para determinados predicados.
trace (ligado) trace (desligado)	Isto é um predicado, não uma diretriz. Liga ou desliga o acompanhamento (dependendo do valor de argumento utilizado). Pode ser utilizado apenas se a diretriz "trace" ou "shorttrace" estiver no início do programa. Observe que se a diretriz "shorttrace" está no começo do programa, então este será o tipo de acompanhamento que este predicado vai ligar e desligar. Pode ser utilizado diversas vezes para ligar o acompanhamento apenas onde é necessário.
shorttrace	Liga o acompanhamento para todo o programa, mas utiliza a otimização do compilador e, portanto, fornece informações de acompanhamento menos detalhada em algumas circunstâncias.
shorttrace p1,p2...	Liga "shorttrace" para predicados especiais.
ALT-T	Esta combinação de teclas pode ser utilizada para ligar e desligar o acompanhamento. Operando ALT e T, você pode ligar e desligar o acompanhamento enquanto segue um programa passo a passo.

Tabela 12-3 Mensagem durante o acompanhamento

CALL	Aparece cada vez que um predicado é chamado. Mostra tanto o predicado quanto o valor presente de seu argumento.
RETURN	Aparece quando uma cláusula foi satisfeita e a lógica do programa volta a um predicado anterior. Um asterisco depois de RETURN significa que há outras cláusulas que poderiam combinar com os argumentos de entrada, e que este ponto precisa e já foi marcado para retrocesso futuro.
FAIL	Aparece quando uma cláusula falhou; isto é, a cláusula não pode ser combinada ou Turbo Prolog não consegue executar seu trabalho.
REDO	Aparece quando o programa está retrocedendo. Mostra o nome do predicado rechamado e o valor presente de seu argumento.

A diretriz "trace" coloca o cursor no ponto de ação na janela Editor e mostra as metas e submetas que está tentanto combinar na janela Trace. A Figura 12-1 mostra este tipo de acompanhamento em um programa do disco Biblioteca/Exemplo. A janela Trace foi alargada para mostrar a mensagem obtida em cada passo. Depois que cada mensagem foi colocada na janela Trace, a tecla F10 foi acionada para que se veja o próximo passo. Esta figura é, em essência, uma forma de seqüência de fotografias que mostra como as duas listas foram concatenadas.

SHORTTRACE

Esta diretriz é utilizada da mesma forma que "trace", mas fornece um pouco menos de informação. Isto é porque ela permite ver o que acontece com a otimização do compilador implementada. Quando Turbo Prolog compila algo, normalmente utiliza rotinas de *otimização* especiais para aumentar a velocidade e encurtar caminhos nos locais em que isto pode ser feito, sem alterar o significado do programa. Uma das mais óbvias delas é a eliminação da recorrência final. Experimente uma diretriz "trace" em uma recorrência e depois tente "shorttrace" (veja a Figura 12-2). Você pode ver que uma menor quantidade de mensagens de janela Trace foi gerada pelo mesmo processo de concatenar duas listas.

OUTRAS OPÇÕES DE ACOMPANHAMENTO

Tanto "trace" como "shorttrace" oferecem uma especialização. Você pode escolher o predicado que deve ser acompanhado e deixar que todos os outros sejam proces-

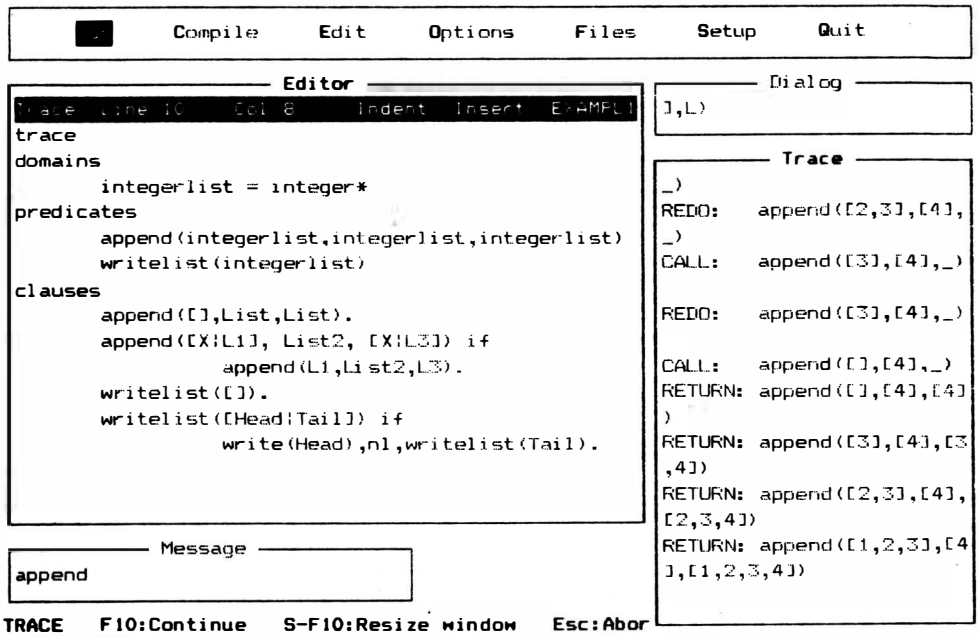


Figura 12-1 Programa de acompanhamento do disco Biblioteca/Exemplo

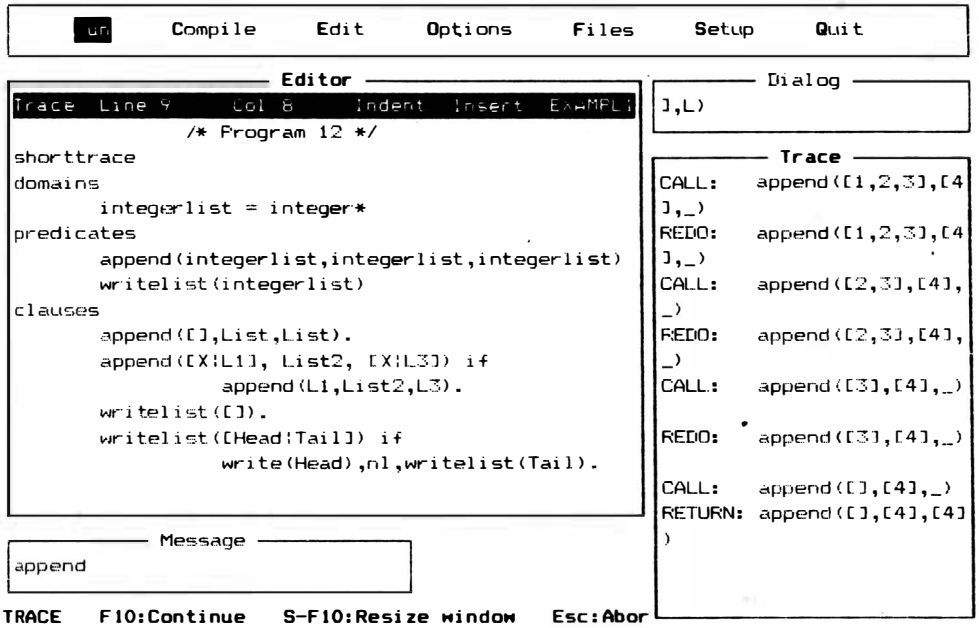


Figura 12-2 Utilizando a diretriz "shorttrace"

sados normalmente. Para fazer isto, digite os nomes dos predicados depois da diretriz, e separe cada nome do próximo com uma vírgula. A Figura 12-3 mostra um exemplo disto no programa EXAMPL16.PRO. Para gerar esta saída, carregue o programa e acrescente a seguinte linha acima das declarações domínio:

```
trace person, common_interest
```

Processe o programa e coloque a meta

```
findpairs
```

Tecle F10 dez vezes. A janela Trace agora deverá estar cheia de informações.

O PREDICADO "TRACE"

O último exemplo de acompanhamento e controle de acompanhamento que você

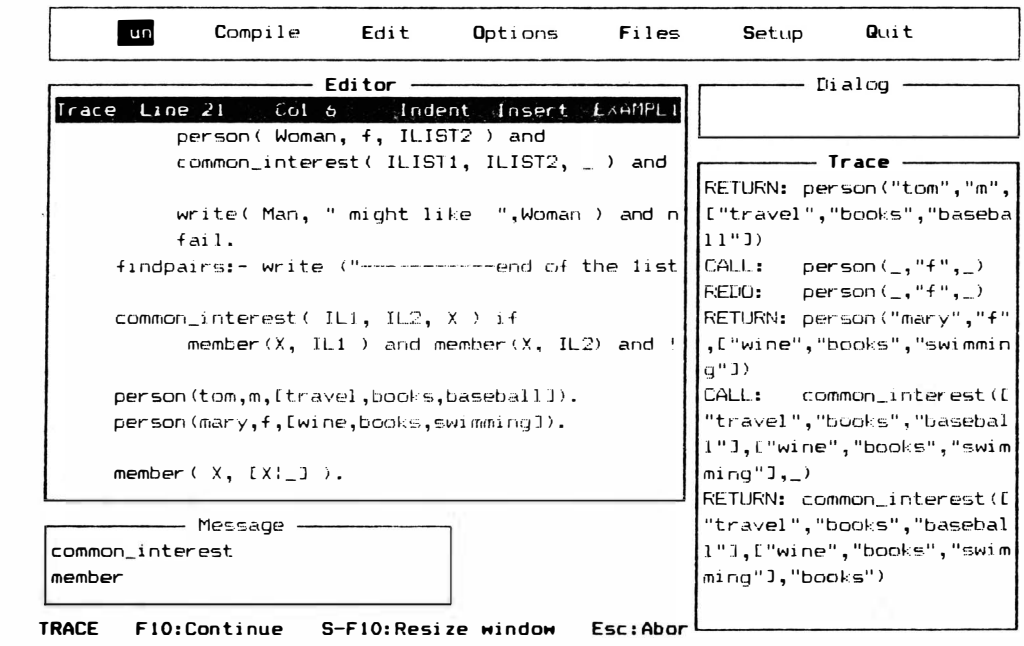


Figura 12-3 Utilizando a diretriz "trace" para predicados especiais

vai ver aqui diz respeito ao predicado "trace". Tem a mesma escrita e utilização básica que a diretriz "trace".

Entretanto, pelo fato de ser um predicado, você o coloca na seção cláusula e não no começo do programa. O predicado "trace" tem um único argumento que pode ter um dos dois valores: ligado e desligado. Para utilizá-lo, você também precisa ter a diretriz "trace" ou "shorttrace" no topo do programa. Incluir também o predicado é mais um meio de manter o acompanhamento confinado apenas àquelas áreas em que é necessário. Isto não é muito significativo aqui, mas imagine percorrer passo a passo um programa grande em que você opte por ligar o acompanhamento apenas na parte que o preocupa e depois desligá-lo.

```
                    /* Program 18 */
trace
predicates
      solve(real,real,real)
      reply(real,real,real)
      mysqrt(real,real,real)
      equal(real,real)
clauses
      solve(A,B,C) :-
                trace (off),
                D = B*B-4*A*C, reply(A,B,D), nl.

      reply(_,_,D) :- D < 0, write("No solution"), !.
      reply(A,B,D) :-
                D = 0, X=-B/(2*A), write("x=",X), !.
      reply(A,B,D) :-
                mysqrt(D,D,SqrtD),
                X1 = (-B + SqrtD)/(2*A),
                X2 = (-B - SqrtD)/(2*A),
                write("x1 = ",X1," and x2 = ",X2).

      mysqrt(X,Guess,Root) :-
               trace (on),
               NewGuess = Guess-(Guess*Guess-X)/2/Guess,
               not(equal(NewGuess,Guess)),!,
               trace (off),!,
               mysqrt(X,NewGuess,Root).
      mysqrt(_,Guess,Guess).

      equal(X,Y) :-
               X/Y > 0.99999   , X/Y < 1.00001.
```

Figura 12-4 Listagem para colocar a diretriz "trace" no topo com o acompanhamento ligado apenas para parte de uma única regra

As Figuras 12-4 e 12-5 mostram isto em ação. Desta vez, o programa-exemplo da Borland de número 18 é utilizado. Você quer ver o que acontece em uma pequena parte do cálculo, de modo que deve colocar a diretriz "trace" no topo, desligar o acompanhamento no começo (conforme mostrado na Figura 12-4), para depois ligá-lo naquela parte que deseja compreender (mostrado na Figura 12-5). Para processar o programa, coloque a meta

```
solve(1,-3,2)
```

e tecle F10 até o aparecimento da solução final na Figura 12-5. Aparece uma boa quantidade de acompanhamento, apesar da área acompanhada ser pequena. Isto se deve ao fato da área acompanhada chamar outros predicados, que retornam respostas de submetas. Estes outros predicados parecem estar fora da área de acompanhamento, mas não estão – são chamados antes do predicado "trace (off)" ser chamado.

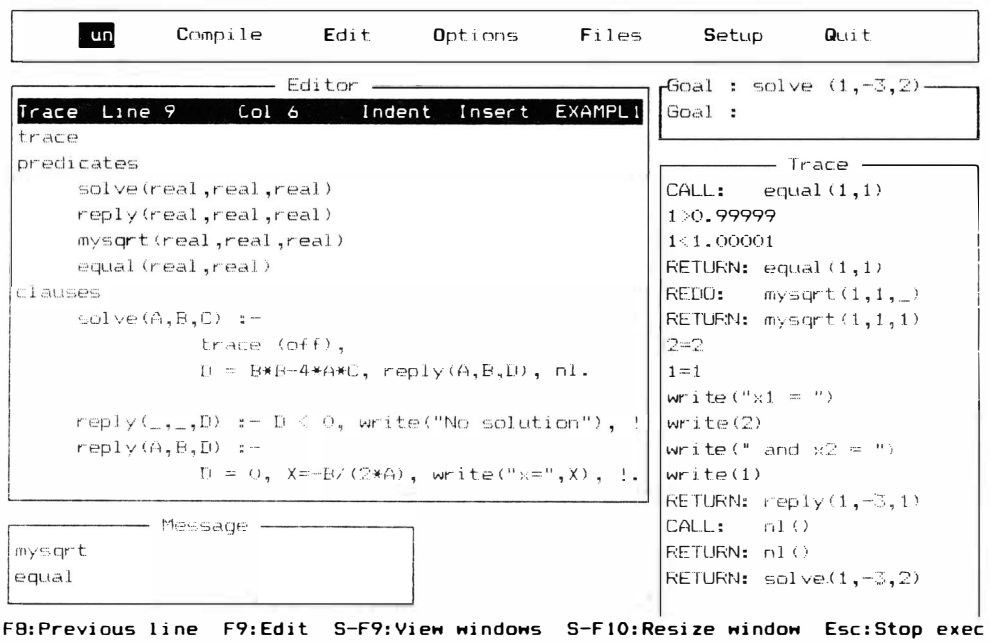


Figura 12-5 Acompanhamento ativado

EXCEÇÕES DURANTE O ACOMPANHAMENTO

Nem todos os predicados são iguais para o acompanhamento. Os CALL e RETURN dos predicados "write", por exemplo, não são vistos pela janela Trace porque têm um número indeterminado de argumentos. Você pode ver isto na Figura 12-1 e 12-2, bem como na Figura 12-5. Os outros predicados e operadores que são casos excepcionais de acompanhamento são mostrado na Tabela 12-4.

DETERMINISMO E CHECK_DETERM

Um predicado tem cláusulas *não-determinísticas* se aquelas cláusulas são capazes de encontrar soluções múltiplas quando ocorre o retrocesso. Apesar de alguns Prolog terem problemas de utilização de memória com claúsulas não-determinísticas, Turbo não os tem – procura estas cláusulas antes do processamento. Uma diretriz especial está disponível se você quer estar ainda mais seguro de que uma cláusula não-determinística não vai piorar a performance do programa.

Tabela 12-4 Operadores e predicados que recebem tratamento de acompanhamento especial

OPERADORES DE COMPARAÇÃO		PREDICADOS
Operador	**Significado**	asserta
=	igual a	assertz
< >	diferente de	bound
> <	diferente de	findall
>	maior que	free
> =	maior que ou igual a	not
<	menor que	readterm
< =	menor que ou igual a	retract
		write
		writef

A diretriz "check_determ" pode ser inserida no começo do programa, da mesma forma como se inseriu a diretriz "trace" em diversos programas deste livro. Caso "check_determ" esteja inserido, Turbo vai avisá-lo sempre que encontrar cláusulas determinísticas. Você pode tratar este tipo de cláusulas não-determinísticas de duas formas. Primeiro, basta teclar F10 para ignorar o aviso e continuar com o processamento do programa (procurando combinar uma determinada meta). Acione qualquer outra tecla, e o sistema vai terminar o processamento do programa. Segundo, em muitos casos você pode

```
                /* Program 23 */
check_determ
domains
    person      = p(name,age,telno,job)
    age         = integer
    telno ,name,job = string
predicates
    readperson(person)
    run
goal
    run.
clauses
    readperson(p(Name,Age,Telno,Job)):-
        write("Which name ? "),readln(Name),
        write("Job ?"),readln(Job),
        write("Age ?"),readint(Age),
        write("Telephone no ?"),readln(Telno).
    run:-
        readperson(P),nl,write(P),nl,nl,
        write("Is this compound object OK (y/n)"),
        readchar(Ch),Ch='y'.
    run:-
        nl,nl,write("Alright, try again"),nl,nl,run.
```

Figura 12-6 A diretriz "check_determ" instalada

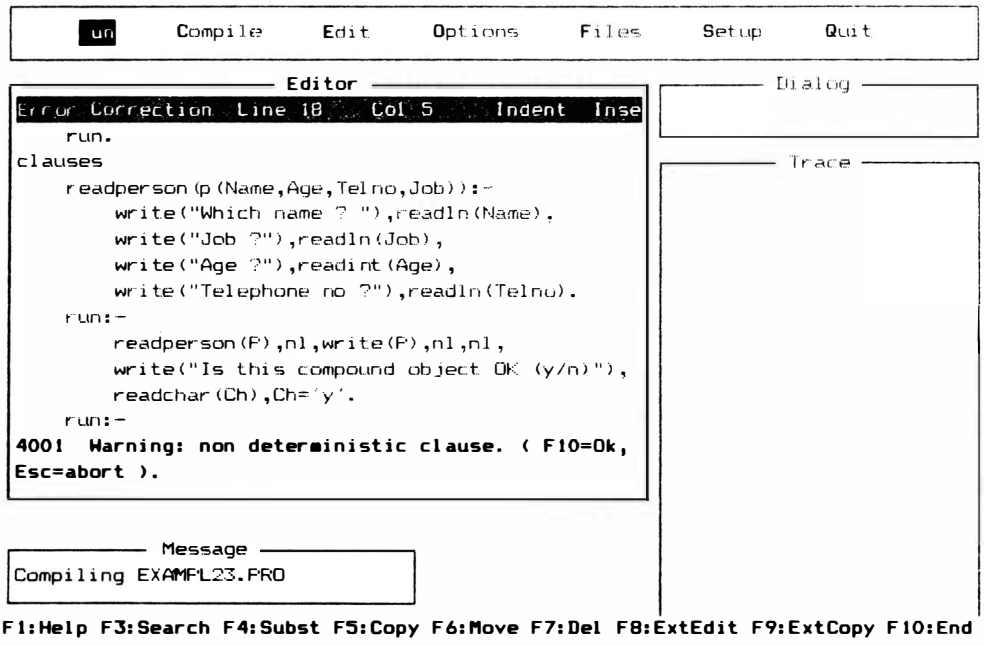


Figura 12-7 Mensagem do compilador com cláusulas não-determinísticas

transformar as cláusulas não-determinísticas em determinísticas, inserindo um comando "Cut"(!). Pelo fato de "Cut" proibir o retrocesso, se for colocado na posição correta, encerra o problema não-determinístico por definição. A Figura 12-6 mostra a diretriz "check_determ" instalada no Programa-Exemplo 23 da Borland. A Figura 12-7 mostra a mensagem que este compilador, solicitado pela diretriz "check_determ", mostra, com exemplos de cláusulas não-determinísticas que o cursor aponta à janela Editor. A mensagem sai pelo lado direito da janela de Editor. A Figura 12-8 mostra o mesmo programa de uma forma limpa. Um programa "Cut" foi inserido para transformar as cláusulas difíceis, e "check_determ" não encontra mais um problema. O programa será compilado corretamente e aguardará uma meta.

PADRÕES DE FLUXO

Outro problema que pode surgir vem de diversas utilizações possíveis de muitos dos predicados. Por exemplo, você já viu que vários predicados intrínsecos têm uma ação quando variáveis ligadas são utilizadas e outra quando variáveis livres são utilizadas. Ou-

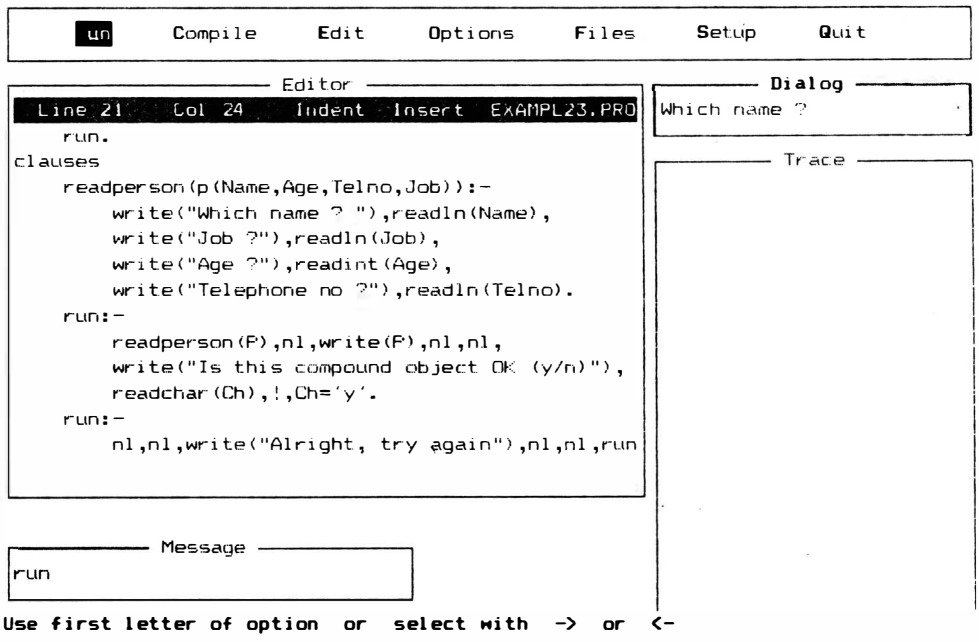


Figura 12-8 Programa-Exemplo 23 com "Cut" inserido

tros que têm mais de uma variável têm utilizações potenciais em quantidade ainda maior, juntamente com suas várias combinações de valores ligados e livres. Predicados definidos pelo usuário podem ter os mesmos recursos.

Em Turbo Prolog, refere-se a isto como *padrões de fluxo compostos.* O fluxo move-se nestas direções: uma variável ligada (ou parâmetro) fornece entrada ao computador (para o programa), ao passo que uma variável livre (ou parâmetro) fornece saída do computador (ao programa). Este padrão misturado de entrada/saída (E/S) requer um código de programa mais longo e é, portanto, algumas vezes útil saber onde tais predicados estão para serem acompanhados e testados. Pela colocação da diretriz "check_cmpio" no começo do programa, você pode fazer com que o compilador o alerte quanto a tais padrões de fluxo.

JUNTANDO TUDO

A diretriz "diagnostics" vai fornecer-lhe uma tabela estática que contém um resumo da informação fornecida pelas diretrizes do compilador mencionadas anteriormente ("check_determ" e "check_cmpio"), bem como informação de domínio e tamanho de código. A Figura 12-9 mostra a diretriz "diagnostics" instalada no Programa-Exemplo 29 da Borland. A Figura 12-10 mostra o resultado da compilação deste programa com a diretriz "diagnostics" – uma grande quantidade de informação na janela Message. De fato, a tela toda é utilizada como uma janela de mensagem, de modo que você possa ver a extensão da tabela de diagnóstico. Nenhum destes predicados está em um banco de dados. Estão todos na área de cláusulas regulares. Todos, exceto um, são determinantes. Ocupam de 76 a 913 bytes e têm uma variedade de domínios e padrões de fluxo.

Você deveria saber também que pode utilizar todas ou qualquer combinação das diretrizes do compilador de uma vez. Foram apresentados separadamente para enfatizar suas ações específicas. Basta colocá-los antes da seção de domínios. Uma exceção é a diretriz "include", que pode ser colocada em qualquer lugar onde pode ir uma das palavras de seção ou divisão.

SUPRIMINDO AVISOS E ESCAPES

Caso "diagnostics" for o oposto do que você quer, e em seu lugar desejar limitar as mensagens de aviso por estar certo do que está fazendo, existem duas diretrizes de compilação que vão ajudá-lo.

```
                /* Program 29 */
diagostics
domains
     list=symbol*
     scores=integer
predicates
     member(symbol,list)
     run
     continue(list,scores)
     yes_no_count(symbol,list)
     guessword(scores,list)
     word(list,integer)
     read_as_list(list,integer)
goal
     makewindow(1,7,0,"",0,0,25,80),
     makewindow(2,7,135,"Counting",1,20,4,34),
     makewindow(3,112,112,"YES",5,5,7,30),
     makewindow(4,112,112,"NO",5,40,7,30),
     makewindow(5,7,7,"",14,20,10,34),
     run.
clauses
     run:- word(W,L),
           shiftwindow(1),clearwindow,
           write("The word has ",L," letters"),
           shiftwindow(2),clearwindow,
           shiftwindow(3),clearwindow,
           shiftwindow(4),clearwindow,
           continue(W,0),fail.
     continue(L,R):-
           shiftwindow(5),clearwindow,
           write("Guess a letter :"),
           Total=R+1,readln(T),yes_no_count(T,L),
           shiftwindow(5),clearwindow,
           guessword(Total,L),continue(L,Total).
     yes_no_count(X,List):-
           member(X,List),shiftwindow(3),write(X),
                          shiftwindow(2),write(X),!.
     yes_no_count(X,_):-
            shiftwindow(4),write(X),
            shiftwindow(2),write(X).
     guessword(Count,Word):-
            write("Know the word yet? Press y or n"),
            readchar(A),A='y',cursor(0,0),
            write("Type it in one letter per line \n"),
            word(Word,L),read_as_list(G,L),
            G=Word,clearwindow,window_attr(112),
            write("Right! You used ",Count," guess(es)"),
            readchar(_),window_attr(7),!,fail.
     guessword(_,_).
     word([b,i,r,d],4). word([p,r,o,l,o,g],6).
     word([f,u,t,u,r,e],6).
     member(X,[X|_]):-!.
     member(X,[_|T]):-member(X,T).
     read_as_list([],0) :-!.
     read_as_list([Ch|Rest],L) :-
            readln(Ch),L1=L-1,read_as_list(Rest,L1).
```

Figura 12-9 A diretriz "diagnostics" instalada no Programa-Exemplo 29

NOBREAK

Em um processamento regular, Turbo Prolog verifica no teclado a existência de CTRL-C ou CTRL-BREAK antes de cada nova chamada de predicado. Caso encontre um deles, encerra o processamento do programa. Em um programa em ciclo, poderá ser a única maneira de sair. Por exemplo, o Exemplo 41 do manual do Turbo Prolog poderá ser interrompido apenas com CTRL-BREAK. Caso você acrescente a diretriz "nobreak" no começo deste programa, conforme indicado na Figura 12-11, a única maneira de sair é reinicializar o sistema operacional – CTRL-BREAK não resolve o problema. Tudo o que você armazenou em RAM perdeu-se. Mesmo assim, você poderá querer utilizar "no-break" algumas vezes, porque o procedimento de interrupção diminui a velocidade do programa.

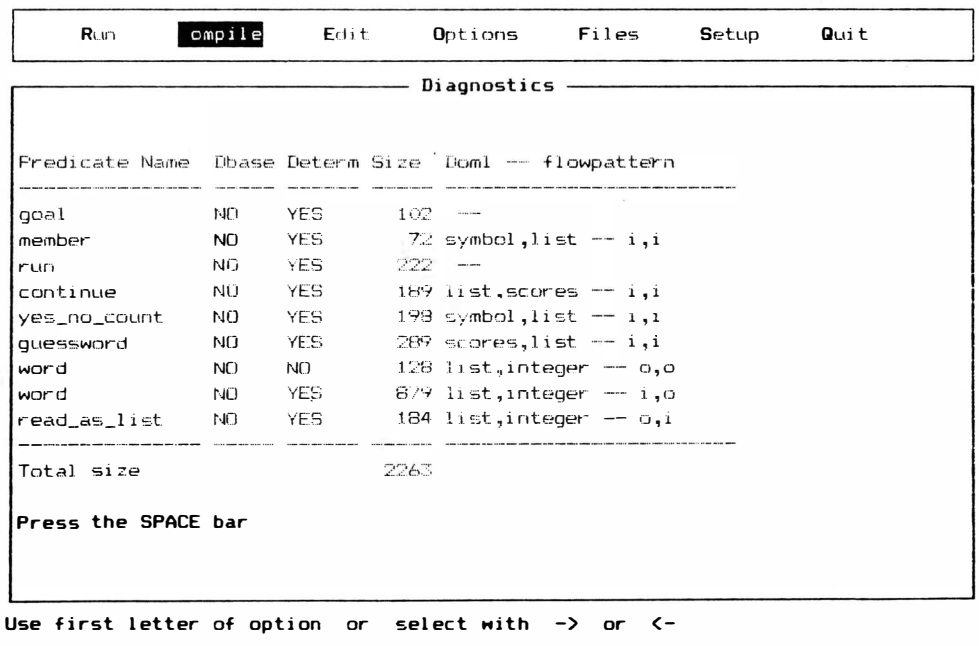


Figura 12-10 Compilando o programa-exemplo 29, usando a diretriz "diagnostics"

NOWARNINGS

Você está familiarizado com avisos do tipo "Variable occurs only once in a clause" (Uma variável ocorre apenas uma vez em uma cláusula), e "Variable not bound on return from clause" (Variável não ligada de volta à cláusula) que aparece durante diversas aplicações. Você não vai vê-los se utilizar a diretriz "nowarnings". É claro, o programa continua tendo estas condições. Você apenas não será avisado. Caso tenha certeza do que está fazendo, elimine as sinalizações, e ganhará um pouco de velocidade.

ABRINDO ESPAÇO

As diretrizes "code" e "trail" informam ao sistema a quantidade de memória que deve manter disponível, para o código do programa e o registro de variáveis de referência, respectivamente. Estes são explicados rapidamente na Tabela 12-1. Não se preocupe com eles antes de passar para aplicações realmente avançadas.

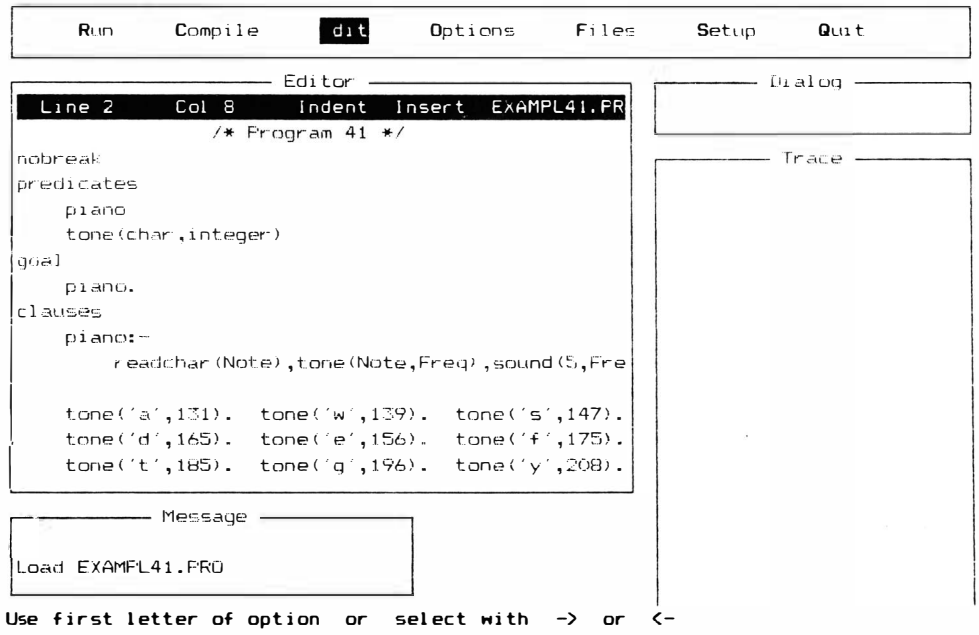


Figura 12-11 Acrescentando uma diretriz "nobreak"

INCLUINDO OUTROS ARQUIVOS

Em muitos momentos, você poderá querer incluir outros arquivos na compilação. Estes arquivos poderiam conter códigos de outra linguagem de programação, ou poderiam ser outros arquivos de Turbo Prolog que você não quer reescrever. A diretriz "include" permite utilizar o código "como é" em alguns casos.

A diretriz "include" não precisa ser colocada no começo de um programa como as outras. Em vez disto, pode ser colocada em qualquer local onde uma palavra-chave pode aparecer (um delimitador de seção como "clauses", "goal" ou "global"). Você a utiliza para solicitar ao Turbo Prolog para compilar outros arquivos como um código-objeto único. Um arquivo incluído poderá conter uma meta, domínio ou predicado, declarações ou cláusulas. Você a escreve da seguinte forma:

```
include "DosFileName"
```

onde "DosFileName" é o nome do arquivo em disco que quer incluir.

Aqui está um exemplo de uma diretriz "include" em funcionamento. O programa da Figura 12-12 não tem toda informação necessária – o predicado "paid_enough" não está totalmente pronto. Mas em vez de construir mais texto-fonte neste arquivo, você pode referir-se a outro arquivo feito antes. Este arquivo (o programa mostrado na Figura 12-13) detalha o predicado "paid_enough". Pelo fato do primeiro programa ter a linha "include" referindo-se ao segundo programa, a combinação dos dois deveria ser compilada corretamente.

Tente você mesmo. Pegue o primeiro programa na tela (como mostrado na Figura 12-12) e depois tecle C para compilar. Você verá a compilação começar, e verá mesmo o Turbo Prolog carregar automaticamente o segundo programa (o *include file*) na janela Editor. A primeira vez que tentei isto, o compilador acabou parando em um erro tipo domínio. George foi listado como ganhando $45000 por ano, que é um inteiro grande demais para ser processado por Turbo Prolog. Turbo, portanto, pulou de volta ao modo de edição. Alterei "45000" para "30000" (lembre-se de não utilizar vírgulas no meio deste tipo de números), e depois teclei F10. Normalmente, a tecla tenta novamente uma compilação que falhou. Faz isto aqui, também, mas automaticamente grava a versão consertada do arquivo incluído com o qual você acabou de trabalhar. Depois, volta ao primeiro arquivo, começa a compilá-lo, chama o arquivo incluído, acrescenta-o ao conjunto e termina. Você está pronto para o processamento, conforme indicado na Figura 12-14.

```
success (george) and success (phil)
```

obterá a resposta "False". Por que? Tente segui-lo. Acrescente a diretriz "trace" ao início do programa INCLUDE1. Depois, recompile e reprocesse. Estabeleça uma meta, e percorra o programa com a tecla F10. Você não verá apenas um acompanhamento no programa INCLUDE1, verá o acompanhamento pular para o arquivo incluído (include2.pro) e continuar ali (como mostrado na Figura 12-15).

REFERENCIANDO OUTRO ARQUIVO

O predicado "include" permite que você especificamente declare que outro arquivo colocar nesta compilação. Os arquivos chamados poderão também conter uma diretriz "include" que chama mais um arquivo. Entretanto, você não deve ter diretrizes "include" que formem ciclos (chamando outra vez um dos arquivos já incluídos).

A utilização da diretriz "include" não é a única maneira de se trabalhar com arquivos múltiplos para uma única compilação. Existe ainda programação modular. Mas antes de entrar nisto, examine algumas das opções, ainda não discutidas, que estão disponíveis para compilação de arquivos simples.

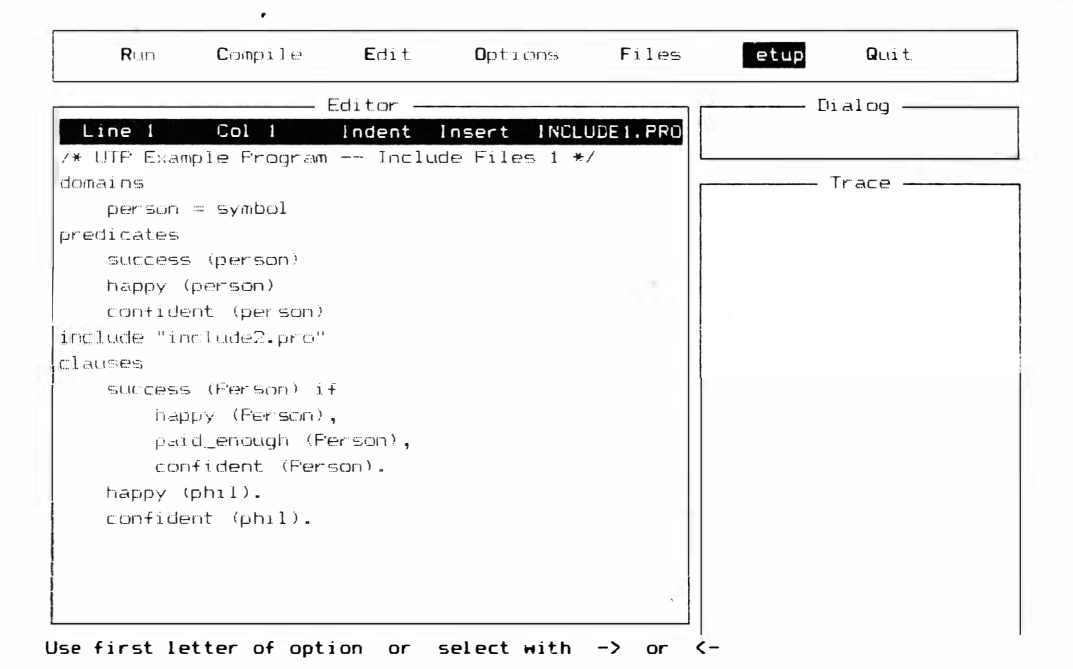


Figura 12-12 Um programa com o predicado "paid_enough" não inteiramente concluído

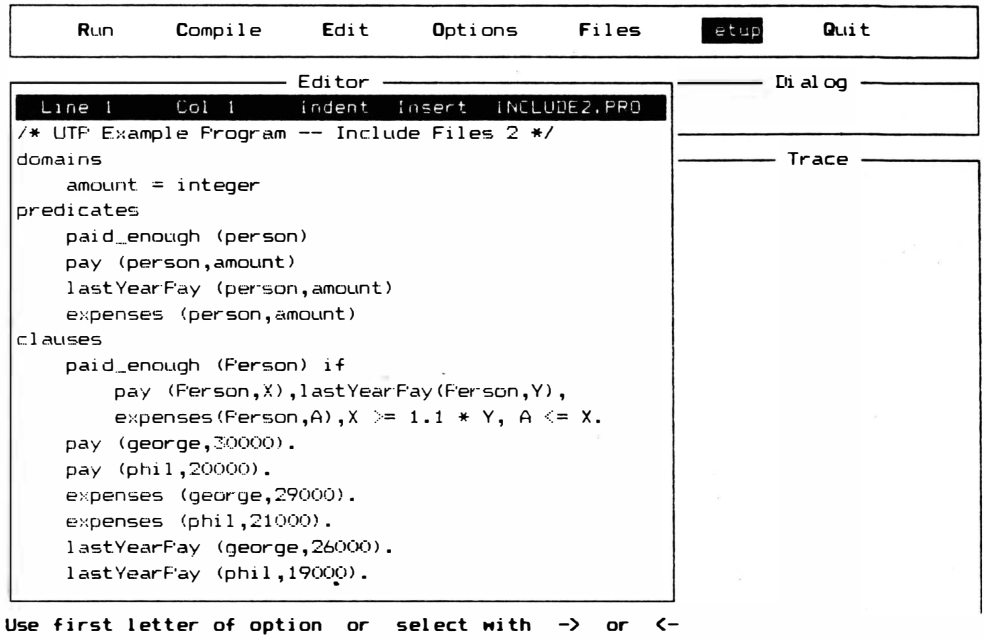


Figura 12-13 Arquivo detalhando o predicado "paid_enough"

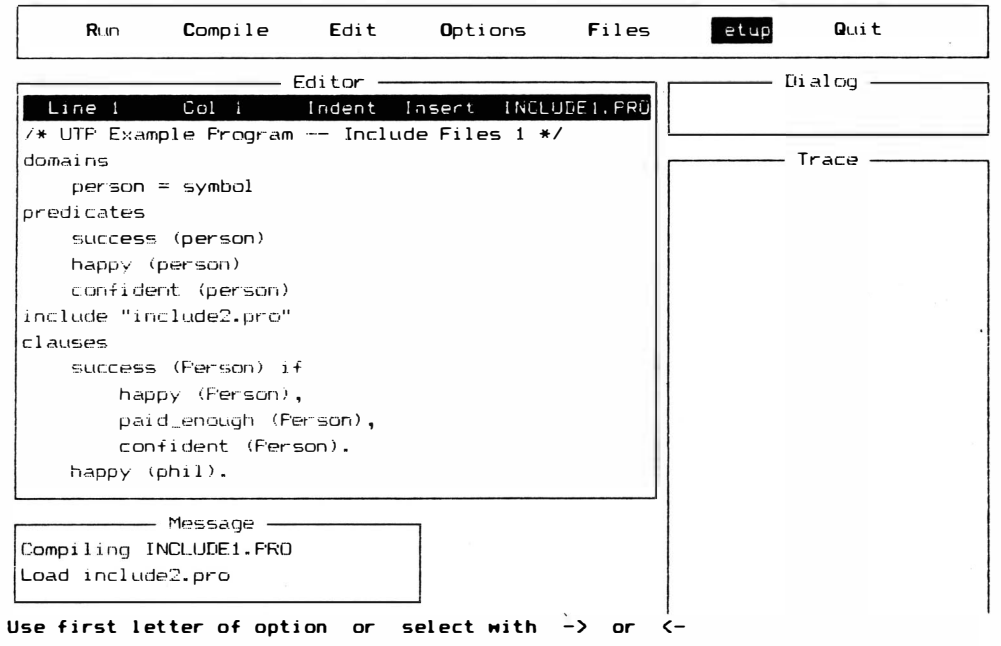


Figura 12-14 Programa recompilado com arquivo "include"

ALTERNATIVAS DE COMPILAÇÃO

Quando você utiliza o procedimento de compilação padrão, está compilando seu código diretamente em RAM e processando-o dali. Isto é muito rápido, mas limita severamente o tamanho de seu programa. Para conseguir acesso a mais dados e códigos e fazer programas que não são completamente dependentes do sistema de desenvolvimento Turbo Prolog com menus e janelas para ser processado, você pode compilar em disco.

COMPILANDO EM DISCO

Todos os programas utilizados até aqui neste livro foram compilados na memória (RAM). Quando você compila um programa em disco, pode processá-lo mais tarde sem iniciar o sistema de desenvolvimento Turbo Prolog. Tudo o que tem a fazer é digitar o nome do programa e teclar RETURN.

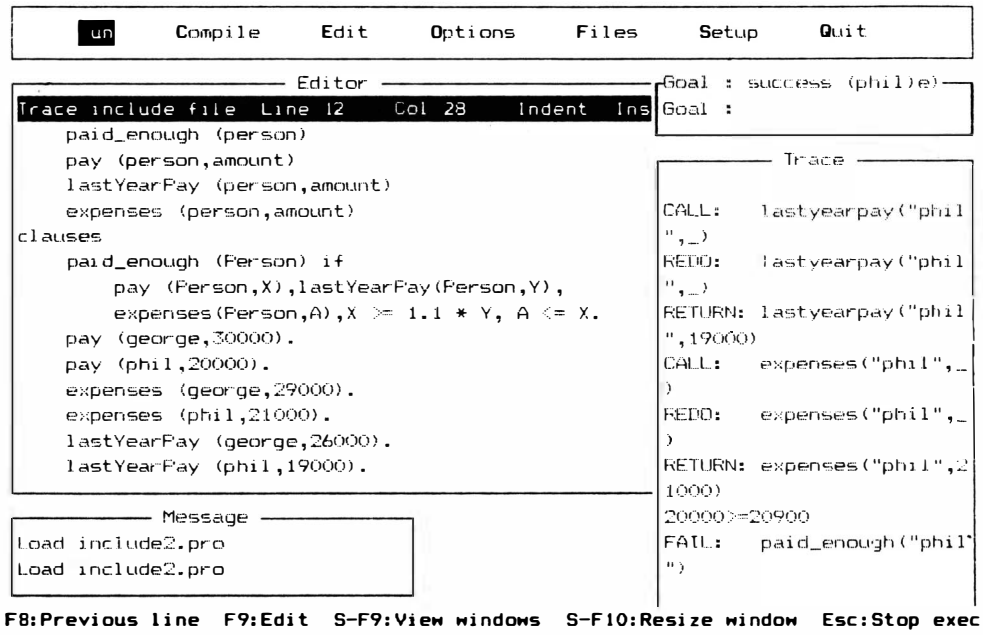


Figura 12-15 Acompanhamento no programa INCLUDE1

EXE

A maneira mais simples de fazer a compilação para o disco é escolher a opção EXE no menu Options (conforme mostrado na Figura 12-16). No momento em que a escolheu, o menu desaparece novamente, de modo que deve ter fé em ter pego a parte certa do menu. Depois de escolher EXE, você manda Turbo Prolog compilar. Você vai criar um arquivo OBJ e depois "linkar" (conectar) aquele arquivo para fazer um arquivo EXE.*Linkar* é o processo de acrescentar código padrão ao código desenvolvido por você. O arquivo que faz esta "linkagem" é PLINK.BAT. Quando você compila um arquivo EXE em disco, não precisa pensar na "linkagem". Isto é feito automaticamente pelo compilador. Para que um programa EXE funcione adequadamente, você precisa ter certeza de que vários arquivos estão no diretório adequado.

O arquivo COMMAND.COM (parte do sistema operacional) está no diretório DOS especificado pelo menu setup. O arquivo PLINK.BAT tem que estar no diretório OBJ especificado pelo menu setup. O arquivo LINK.EXE tem que estar no diretório OBJ especificado pelo menu setup ou em um diretório especificado por um percurso no arquivo

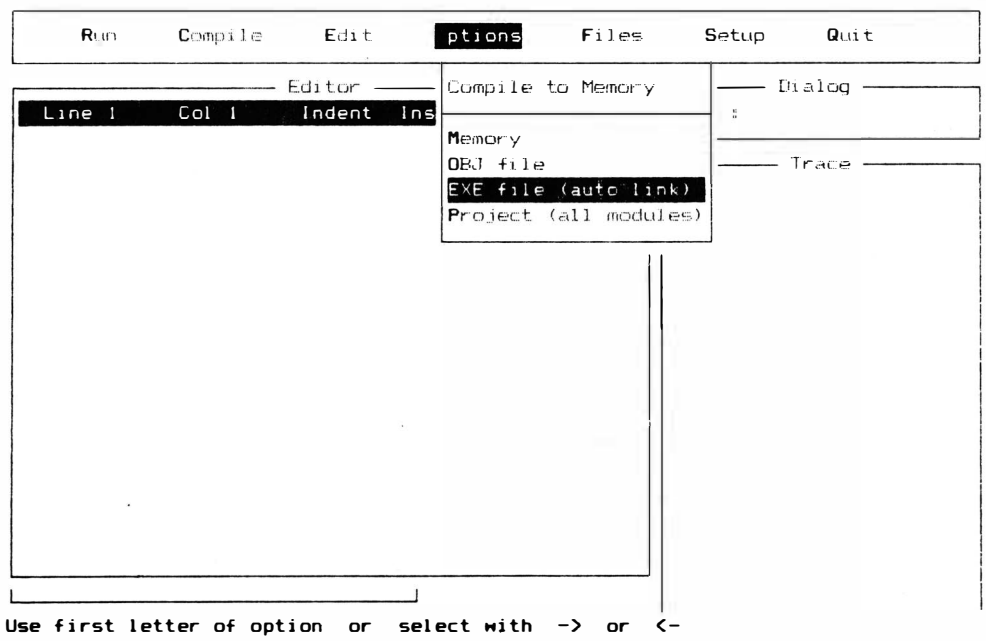


Figura 12-16 Seleção .EXE do menu Options

PLINK.BAT. Você pode mover os arquivos para o diretório adequado ou alterar as especificações do menu setup. Será preciso garantir também que tenha memória suficiente (RAM) em seu computador para todo o programa "linkado". Caso isto esteja em discussão, e você queira fazer uma versão EXE independente de seu programa, deverá escolher a opção OBJ do menu Options.

OBJ

Caso você escolha compilar em um arquivo OBJ, terá de seguir a compilação com uma "linkagem". Para isto, tudo o que tem a fazer é chamar apropriadamente o arquivo PLINK.BAT. Escolha a opção OBJ no menu Options. Depois, tecle C para fazer a compilação. Depois da compilação, verifique o diretório da área que tem o arquivo OBJ. Deverá haver, também, um arquivo SYM que contenha a tabela de símbolos. Vá ao diretório que contém o arquivo PLINK.BAT, e depois digite "plink" seguido por um espaço e pelo nome do arquivo OBJ que deseja "linkar". Você pode também acrescentar dois outros parâmetros: o percurso para o programa EXE e o percurso ao diretório Turbo.

Quando você processa a operação de "linkagem" no arquivo OBJ, transforma-o em um arquivo EXE ou em um programa. Este tipo de programa pode ser processado simplesmente digitando seu nome e teclando RETURN. Para especificar onde deveria estar o arquivo EXE no disco, basta pular um espaço depois da descrição do arquivo OBJ e depois digitar o nome do percurso que deseja utilizar. O arquivo PLINK.BAT precisa dos arquivos INIT.OBJ e PROLOG.LIB. Caso não estejam no mesmo diretório que PLINK você tem de pular outro espaço depois da especificação do percurso EXE e digitar o percurso à parte do disco onde estes podem ser encontrados. A Tabela 12-5 resume estas opções.

Tabela 12-5 Parâmetros PLINK.BAT

O arquivo linkado é chamado com esta sintaxe:

plink.bat %1 %2 %3

%1	Refere-se ao nome do projeto ou módulo a ser "linkado".
%2	Refere-se ao acionador e percurso onde o arquivo EXE será gravado.
%3	Refere-se ao acionador e percurso no qual PLINK.BAT vai procurar os arquivos INIT.OBJ e PROLOG.LIB.

PROGRAMAÇÃO MODULAR

Existe ainda mais uma maneira de compilar. Permite que você utilize arquivos-

fontes de diversas linguagens, em uma variedade de percursos de disco. A *programação modular* constrói um programa final (ou "project") a partir de alguns ou vários módulos. Todos os módulos em um projeto têm de compartilhar uma tabela de símbolos interna. Portanto, cada módulo precisa conhecer o projeto ao qual está destinado. Todo módulo precisa de uma diretriz "project" juntamente com um nome de projeto na primeira linha do código.

MÓDULOS DE PROGRAMA

Você pode criar programas a partir de módulos que são escritos, editados e compilados separadamente. Existem três vantagens em se utilizar este enfoque em programas longos; duas são geralmente verdadeiras em diversas linguagens de programação, e a terceira é específica ao Prolog.

SIMPLIFICAR A DEPURAÇÃO

Tanto maior um programa, mais difícil a sua depuração. Existe cada vez mais um maior número de combinações de coisas que podem interagir de uma forma que você não quer e que poderá não perceber. Uma resposta a este problema é trabalhar com módulos. Cada módulo é prático e pequeno, e cada um cuida de uma única tarefa. Um módulo pode ser rigorosamente testado e depurado antes de ser acrescentado ao programa todo. Depois, quando você precisar modificar ou atualizar um programa, poderá facilmente modificar apenas alguns módulos em lugar de todo programa. Isto é verdade em muitas linguagens de computação.

INTERFACE COM OUTRAS LINGUAGENS

A programação modular também facilita a tarefa de incluir códigos de programas escritos em outra linguagem. Turbo Prolog pode combinar módulos em outras linguagens e "linká-los" em um programa C, Pascal ou FORTRAN e assembler. A Borland espera atualizar Turbo Pascal, de modo que os módulos Turbo Prolog possam ser incluídos no programa.

RETER VARIÁVEIS LOCAIS

Todos os nomes de predicados e domínios são locais em Turbo Prolog (a não ser que você utilize as declarações "global" especiais do Turbo Prolog). Isto significa que você pode reutilizar um nome de predicado ou domínio em um módulo diferente, juntar os

módulos, e as diversas partes do código permanecem distintas e não interferem com outras. Coisas diferentes poderão ter os mesmos nomes em módulos diferentes, mas não serão confundidas pelo código ou parecerão a mesma coisa. Isto é particularmente impressionante quando se sabe que Turbo Prolog mantém os nomes das variáveis do próprio programador ao compilar (nem todos os Prolog o fazem). Isto facilita a análise dos programas Turbo Prolog e a compreensão do que está acontecendo.

REQUISITOS DA PROGRAMAÇÃO MODULAR

Existem quatro coisas que devem ser feitas para fazer funcionar um programa modular: estabelecer uma definição de projeto, declarar variáveis globais, garantir que pelo menos um módulo tenha uma meta interna, e garantir que os arquivos certos para "linkagem" estejam nos diretórios corretos (descrito anteriormente na seção "Compilar em disco"). O módulo com a meta é o módulo principal. Depois de definir o projeto e declaradas as variáveis globais, pode-se começar a compilar cada módulo, um por um, utilizando a opção OBJ ou EXE. Caso você utilize a opção OBJ, não ligue estes arquivos até que o último módulo seja compilado. Quando você chegar ao último módulo, poderá utilizar a opção OBJ, sair do Turbo Prolog e chamar o linker você mesmo (PLINK.BAT); ou poderá escolher uma opção EXE, observar o módulo final compilar, e depois "linkar" com todos os outros.

DEFINIÇÃO DE PROJETO

Para utilizar programação modular, será preciso especificar que módulos deverão ser utilizados no programa compilado, final. Você cria este arquivo escolhendo a opção Module List do menu Files (conforme mostrado na Figura 12-17), fornecendo o nome de projeto àquela lista, e depois editando a lista para que contenha todos os nomes de módulos. O nome não precisa conter a extensão (tipo de arquivo) e deve ser seguido por um sinal (+). O arquivo do projeto terá a extensão ".PRJ".

A definição do projeto faz duas coisas. Primeiro, permite ao compilador estabelecer uma tabela de símbolos com o nome do projeto. Esta tabela será compartilhada por todos os módulos e será armazenada como um arquivo em disco (com o nome de projeto e tipo de arquivo SYM) no diretório OBJ. Segundo, o arquivo PLINK.BAT poderá se referir ao arquivo PRJ no momento da "linkagem" para ver que arquivos precisam estar "linkados".

Para cada módulo saber qual arquivo SYM utilizar, você precisa colocar a dire-

triz do compilador "project" no começo do código do módulo e seguida de um espaço e do nome do projeto.

PREDICADOS E VARIÁVEIS GLOBAIS

Estes já foram mencionados diversas vezes neste livro. As declarações permitem ao Turbo Prolog testar erros de sintaxe e domínio, mesmo entre módulos. Apesar de ser prático o fato das variáveis Prolog serem locais, existem ocasiões em que você quer que um módulo tenha o mesmo significado que outro ao referir-se a um nome específico de variável. Você pode conseguir isto declarando a variável como sendo *global*.

Existem duas partes para este tipo de declaração: o predicado e o domínio.

PREDICADOS GLOBAIS

Os predicados podem ser declarados como globais na seção de *predicados globais*. Entretanto, você poderá ter apenas uma seção deste tipo em todos os módulos (para compilação) agrupados. Uma diferença entre estas declarações e declarações normais de predicado é que os padrões de fluxo de predicados globais têm de ser especificados antecipadamente. Um predicado global poderá ter mais do que um padrão de fluxo, e diversos padrões poderão ser mostrados em uma mesma linha. O predicado é seguido pelos argumentos, um hífen, e depois os padrões legítimos sintaticamente, entre parênteses. Todo argumento tem que ter um "(i)" (entrada ou livre) ou um "(o)" (saída ou ligado).
Aqui está um exemplo:

```
global predicates
        glob_pred_1 (dom1, dom2, dom2) – (i,o,o)
```

DOMÍNIOS GLOBAIS

Os domínios precisam ser globalmente declarados a não ser que sejam padrões. Isto é feito na seção de domínios globais. Estas declarações parecem o mesmo que outras declarações de domínio. Aqui está um exemplo do predicado anterior:

```
dom1 = symbol dom2, dom3 = integer
```

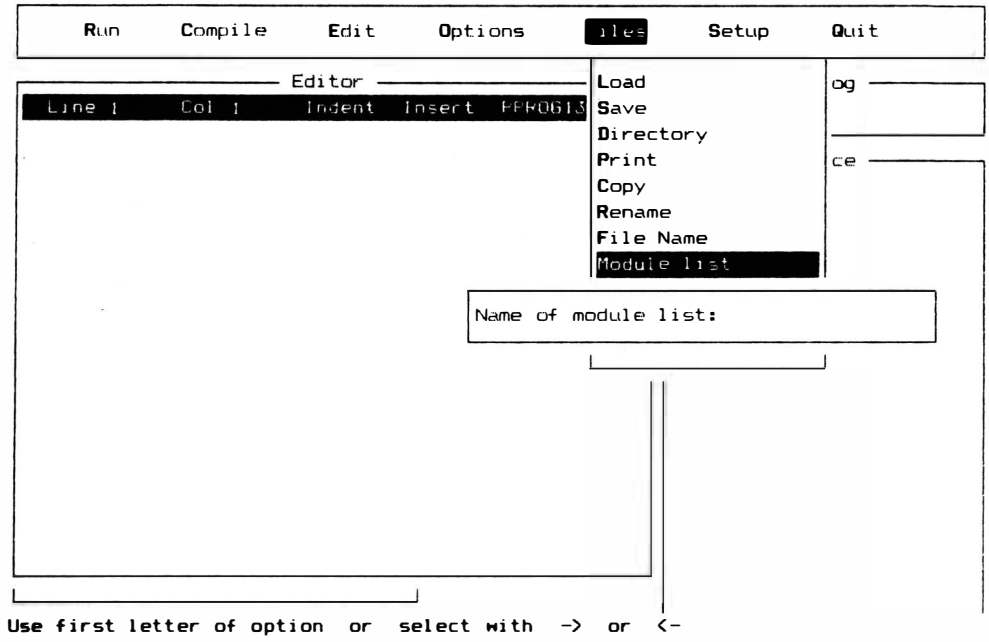


Figura 12-17 Opção Module List do menu Files

UTILIZANDO UMA DIRETRIZ "INCLUDE" PARA DECLARAÇÕES GLOBAIS

Assim que tiver escrito as suas declarações globais, a maneira mais fácil de garantir que todo e qualquer módulo a ser compilado tenha acesso a eles é fazer um arquivo separado e colocar uma diretriz "include" no texto de cada módulo.

DIVISÕES DE PROGRAMA

Existem mais divisões de programa possíveis do que aquelas apresentadas neste livro. Muitas são utilizadas em GeoBase, o aplicativo descrito no Capítulo 13. Estas divisões, na ordem em que aparecem, são as seguintes:

- Diretrizes de compilador
- Domínios
- Domínios globais
- Banco de dados
- Predicados
- Predicados globais

- Meta
- Claúsulas.

A maioria destas divisões é opcional. Como você já viu, um programa pode consistir totalmente de uma só meta, ou pode ter um predicado incorporado em uma seção de cláusulas e depender de uma meta externa. Apesar disto, estas outras divisões permitem que você declare outros tipos de informação que podem ser muito práticas em programas longos e complexos.

DIRETRIZES DO COMPILADOR

Foram discutidos anteriormente neste capítulo.

DOMÍNIOS

Você já está familiarizado com esta seção, que descreve o tipo de dados utilizados pelos predicados.

DOMÍNIOS GLOBAIS

Esta seção permite que você especifique domínios que operarão para mais de um módulo.

BANCO DE DADOS

Outro tipo de arquivo com o qual Turbo Prolog pode trabalhar é um *banco de dados dinâmico*. A lista padrão de cláusulas (fatos e regras) é um banco de dados, e Turbo Prolog tem um motor interno para pesquisar este banco de dados. A combinação e retrocesso que ocorre automaticamente quando uma meta é estabelecida é uma maneira eficiente e rigorosa de pesquisar um banco de dados. Entretanto, algumas vezes é muito útil acrescentar ou subtrair fatos e regras enquanto se estabelecem metas. Na verdade, estes são predicados externos que você quer considerar ao inquirir o banco de dados. Este tipo de banco de dados dinâmico (alterável) é declarado em uma seção separada no começo do programa e pode ser gravado em um arquivo separado pela utilização do predicado "save". Este e outros predicados que podem cuidar de bancos de dados dinâmicos estão relacionados na Tabela 12-6. A declaração do banco de dados vem antes da outra declaração de predicado. Declara os predicados para o banco de dados que podem ser feitos por cláusu-

las digitadas ou cláusulas trazidas de um arquivo em disco pelo predicado "consult".

Três predicados padrões permitem que você construa um banco de dados dinâmico: "asserta", "assertz" e "retract". Os dois primeiros acrescentam fatos ao banco de dados; o último cancela fatos. O predicado "asserta" coloca o fato novo antes de todos os fatos anteriores do mesmo predicado, ao passo que o predicado "assertz" coloca o novo fato depois dos anteriores. A ordem tem importância, porque Prolog procura de cima para baixo, e da esquerda para a direita, em uma lista.

Tabela 12-6 Predicados tratando bancos de dados dinâmicos

asserta (< fato >) dbasedom–(I)

Acrescenta um fato ao banco de dados armazenado em RAM. O fato é colocado antes de qualquer outra cláusula do mesmo predicado.

assertz (< fato >) dbasedom–(I)(O)

Acrescenta um fato ao banco de dados armazenado em RAM. O fato é colocado depois de qualquer outra cláusula do mesmo predicado.

retract (< fato >) dbasedom–(I)

Este é o oposto dos predicados "asserta" e assertz". Cancela o primeiro fato no banco de dados que combina com o " < fato > " no argumento.

consult (DosFileName) string–(I)

Coloca um arquivo de banco de dados na memória. O arquivo será um arquivo de texto: se qualquer uma das linhas dele não vai de encontro aos padrões de sintaxe do Turbo Prolog, o predicado "consult" falha. Com freqüência o arquivo de texto será um que se coloca no disco com o predicado "save".

save(DosFileName) string–(I)

Grava as cláusulas em disco com o nome "DosFileName". O arquivo de texto terá uma cláusula em cada linha e poderá ser editado diretamente. Poderá também ser chamado à memória por um predicado "consult" em um programa.

PREDICADOS

Nesta seção você pode declarar oficialmente todos os predicados definidos pelo usuário que serão encontrados na seção cláusulas. Ao declará-los, você informa ao Turbo Prolog coisas como que tipo de dados os predicados vão utilizar e quantos argumentos possuem. Isto permite que o compilador verifique a existência de erros de sintaxe e tipos.

PREDICADOS GLOBAIS

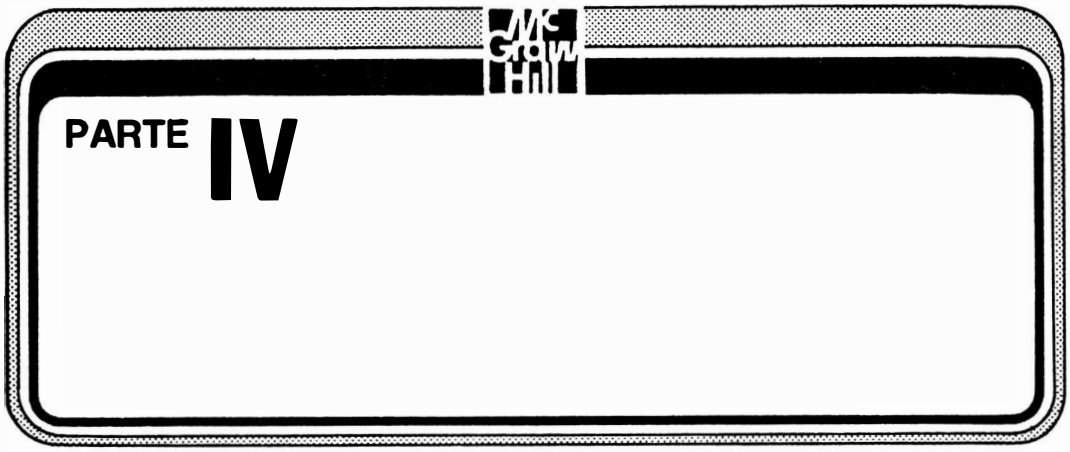
Você pode também declarar predicados que podem ser bons para mais de um módulo. Apenas uma seção de predicados globais pode ser utilizada para uma compilação. Em outras palavras, quaisquer outros arquivos incluídos não podem ter a sua seção de predicados globais próprios se o compilador já leu um.

META

Os Prolog Mellish e Clocksin padrões operam inteiramente com metas externas. Você tem que processar o programa e depois informar-lhe a meta que deseja atingir. Turbo Prolog permite que você utilize uma meta interna, de modo que o sistema não esperará outra meta, trabalhando imediatamente no sentido da meta especificada.

CLÁUSULAS

As cláusulas são os fatos e regras que Prolog pesquisa para atingir a sua meta.



Turbo Prolog – Guia do Usuário apresentou-lhe Prolog de forma geral e de forma específica aos princípios do Turbo Prolog. Agora, você deve estar pronto para problemas Prolog de maior desafio, de modo que esta seção apresenta "GeoBase", o banco de dados inquisitivo, bastante poderoso, de linguagem natural da Borland. GeoBase é tão interessante quanto divertido, e pelo fato de ter sido escrito em Turbo Prolog, deverá dar-lhe uma idéia da força das aplicações escritas nesta linguagem.

CAPÍTULO 13

UTILIZANDO GEOBASE

Quando você adquire Turbo Prolog, obtém mais do que uma ferramenta de desenvolvimento de software. Obtém também GeoBase, um exemplo de programa aplicativo, escrito em Prolog. GeoBase é um banco de dados em linguagem natural; isto é, combina um banco de dados (informações a respeito da geografia dos Estados Unidos) com uma interface que pode interpretar perguntas escritas no inglês do dia-a-dia. Em um determinado sentido, GeoBase é constituído por duas das aplicações mais importantes do Prolog: sistemas especialistas e linguagem natural.

Além de obter este aplicativo em Turbo Prolog você obtém o código-fonte completo e comentado. Isto torna GeoBase um laboratório maravilhoso para sua pesquisa Prolog. Depois de compilar, processar e fazer perguntas ao programa, você pode examinar o código, modificá-lo, melhorá-lo, ou tirar pedaços dele para outros programas que deseja escrever. GeoBase utiliza muitas das facilidades do Turbo Prolog e permite que você experimente características avançadas como compilação EXE e modificação de banco de dados.

Se você quer utilizar GeoBase, este capítulo não é de seu interesse. Mas lembre-se de que este código (funcionando) e os dados aguardam sua inspeção.

O QUE É NECESSÁRIO PARA COMPILAR GEOBASE

A Figura 13-1 mostra os arquivos necessários para compilar e processar GeoBase. Caso você compile em disco, poderá processar GeoBase posteriormente a partir de um único arquivo.

Você encontrará três obstáculos principais ao compilar e processar um programa

grande como GeoBase: pouco espaço em disco, percurso de diretório errado, e pouco espaço RAM.

POUCO ESPAÇO EM DISCO

Formate novos discos e copie apenas os arquivos relacionados na Figura 13-1. Assim terá espaço para qualquer versão modificada dos arquivos que possa querer fazer e para os arquivos que o compilador vai gerar. Você provavelmente já viu como estão cheios tanto o disco Sistema/Programa como o disco Biblioteca/Exemplo, com pouco espaço de sobra para qualquer coisa nova.

```
PROLOG.LIB
PROLOG.EXE
PROLOG.SYS
PROLOG.ERR
PROLOG.HLP
PLINK.BAT
INIT.OBJ

GEOBASE.PRO
GEOBASE.INC
GEOBASE.HLP
GEOBASE.DBA
```

Figura 13-1 Arquivos necessários à compilação de GeoBase

Este obstáculo e o seguinte são mais fáceis de vencer se você tiver um disco rígido, apesar de precisar ter muito mais cuidado com os percursos escolhidos em um sistema de disco rígido.

PERCURSOS DE DIRETÓRIO

Você pode encontrar mensagens de erro dizendo que um determinado arquivo não pode ser aberto. Isto normalmente significa que o arquivo não está no disco onde Prolog espera que esteja. Você apenas precisa utilizar o menu Setup para alterar as condições do diretório, e, no pior caso, sair para o sistema operacional para copiar arquivos em locais mais convenientes.

RESTRIÇÕES DA RAM

Você vai precisar de muita memória para processar GeoBase. Primeiro, tente com o sistema da forma que está. Depois, caso tenha problemas com mensagens de erro, terá de deixar Turbo Prolog para conseguir mais memória livre. Você pode remover alguns programas residentes em RAM, que está utilizando (como SideKick ou Ready!), ou comprar uma placa de memória com mais integrados de memória acrescentados ao seu PC.

INSPECIONANDO O ARQUIVO PRO

Para experimentar GeoBase, dispare Turbo Prolog e carregue o arquivo GEOBASE.PRO na janela Editor. Utilize o menu setup para ampliar a janela Editor, de modo que possa ver a linha toda do código do programa.

A Figura 13-2 mostra o que você vai ver quando a seção título de GeoBase está na janela Editor. Passe para o Editor e utilize as teclas de cursor para se movimentar pelo código. Observe as seções de declaração do começo; observe também os comentários ex-

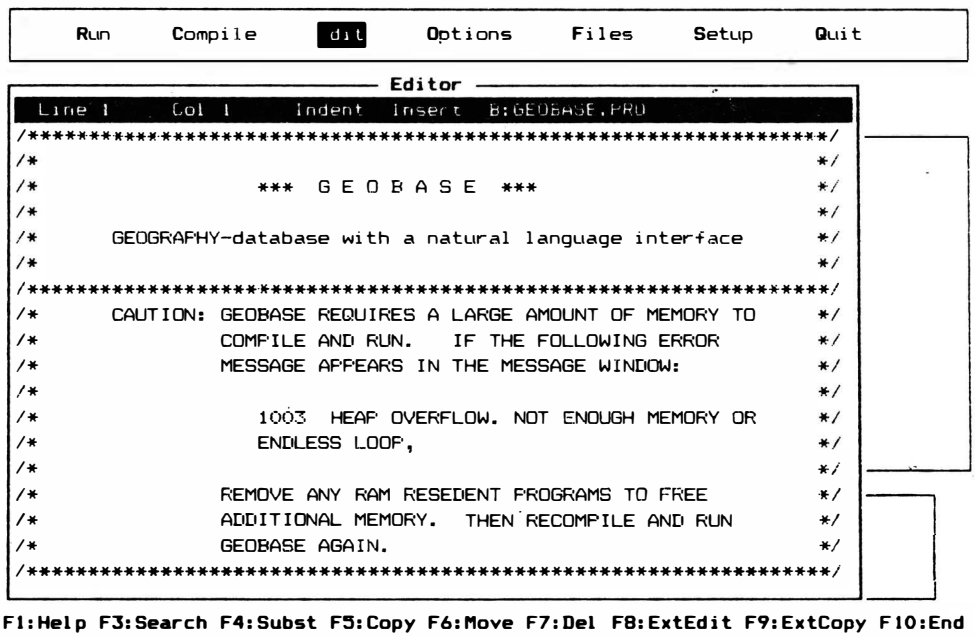


Figura 13-2 Seção título de GeoBase na janela Editor

plicando o que é cada linha. A Figura 13-3 mostra parte deste código. Você nem consegue ver a linha toda do código.

Em seguida você verá as cláusulas que acessam o banco de dados em si (que é um arquivo diferente). Algumas são mostradas nas Figuras 13-4 e 13-5. Finalmente, você vai chegar à linha "INCLUDE" (mostrada na Figura 13-6) que se refere a outro arquivo importante de GeoBase.

COMPILANDO E PROCESSANDO GEOBASE

Tudo o que você precisa fazer agora é sair do editor e teclar C para compilar o programa na memória. Esteja preparado para esperar um pouco pelos resultados. O arquivo GEOBASE.PRO tem que trazer o outro código, e assim há muita compilação que deve ser feita. Ainda, se já não o fez, poderá precisar ir até o menu Setup antes de compilar, digitar a opção Directories, e passar o diretório PRO para o acionador B ou qualquer acionador que contenha o arquivo GEOBASE.PRO.

```
                                    Editor
  Line 37    Col 1      Indent  Insert  B:GEOBASE.PRO
code=2500

DOMAINS
  LIST = SYMBOL*                          /* The domain LIST is a list of words

DATABASE
  /* The Language Database */
  relat(SYMBOL,LIST)                      /* Relation name and a list of attribu
  schema(SYMBOL,SYMBOL,SYMBOL)            /* Entity network: entity-assoc-entity

  entitysize(SYMBOL,SYMBOL)               /* Which attribute gives the size of a
  relop(LIST,SYMBOL)                      /* Example: relop([greater,than],gt] *
  assoc(SYMBOL,LIST)                      /* Alternative assoc names */
  synonym(SYMBOL,SYMBOL)                  /* Synonyms for entities */
  ignore(SYMBOL)                          /* Words to be ignored */
  min(SYMBOL)                             /* Words stating minimum */
  max(SYMBOL)                             /* Words stating maximum */
  big(SYMBOL,SYMBOL)                      /* big, long, high .... */
  unit(SYMBOL,SYMBOL)                     /* Unit for population, area ... */

  /* The Real database */
F1:Help F3:Search F4:Subst F5:Copy F6:Move F7:Del F8:ExtEdit F9:ExtCopy F10:End
```

Figura 13-3 Parte do código GeoBase

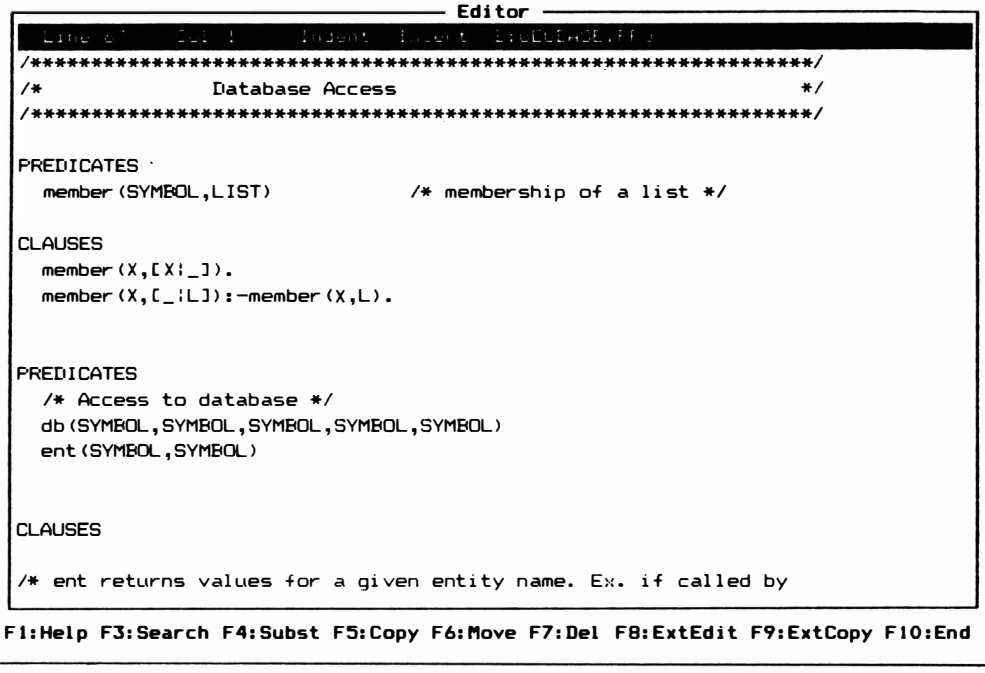


Figura 13-4 Cláusulas para acessar o banco de dados GeoBase

Figura 13-5 Cláusulas adicionais que acessam o banco de dados GeoBase

OPÇÕES DE COMPILAÇÃO

Você pode utilizar também o submenu Options para escolher um método de compilação diferente para GeoBase. Conforme explicado no Capítulo 12, compilar GeoBase em disco como um arquivo OBJ ou EXE permite que você o utilize independentemente do ambiente de desenvolvimento Turbo Prolog – você poderá processá-lo como um programa independente. Digite O para obter o submenu Options, e depois mova a iluminação para OBJ,EXE ou Project.

COMPILANDO EM DISCO

Quando você compila na memória, seu programa só pode ser processado junto com o compilador. Você pode compilar qualquer programa, inclusive GeoBase, em um arquivo em disco que pode ser processado sozinho, sem auxílio adicional do compilador. Caso você escolha esta opção, assegure-se de que o disco contendo os diretórios OBJ e EXE (no menu Setup) tenha espaço suficiente para novos arquivos.

```
    Run      Compile      Edit      Options      File      Setup      Quit

  /* Relationships about mountainss */
  db(mountain,in,state,MOUNT,STATE):-   mountain(STATE,_,MOUNT,_).
  db(state,with,mountain,STATE,MOUNT):- mountain(STATE,_,MOUNT,_).
  db(height,of,mountain,HEIGHT,MOUNT):- mountain(_,_,MOUNT,H1),str_int(HEIGHT,

  /* Relationships about lakes */
  db(lake,in,state,LAKE,STATE):-        lake(LAKE,_,LIST),member(STATE,LIST).
  db(state,with,lake,STATE,LAKE):-      lake(LAKE,_,LIST),member(STATE,LIST).
  db(area,of,lake,AREA,LAKE):-          lake(LAKE,A1,_),str_real(AREA,A1).

  /* Relationships about roads */
  db(road,in,state,ROAD,STATE):-        road(ROAD,LIST),member(STATE,LIST).
  db(state,with,road,STATE,ROAD):-      road(ROAD,LIST),member(STATE,LIST).

  db(E,in,continent,VAL,usa):-          ent(E,VAL).
  db(name,of,_,X,X):-                   bound(X).

INCLUDE "geobase.inc" /* Include parser + scanner + eval + Menusystem */

Use first letter of option  or  select with  ->  or  <-
```

Figura 13-6 Linha "INCLUDE" referindo-se ao arquivo geobase.inc

Caso você tenha muito espaço de memória RAM em seu computador, pode compilar diretamente um arquivo EXE. Isto é fácil, porque a compilação chama automaticamente o linker. Entretanto, caso você não tenha memória suficiente, poderá utilizar a opção OBJ, chamar o linker, esperar o arquivo OBJ ser gravado em disco, sair do Turbo Prolog e depois digitar.

```
plink geobase
```

Quando você rodar o compilador, verá as mesmas mensagens de compilação que vê ao compilar em memória (descritas na próxima seção), mas no fim verá também mensagens a respeito dos arquivos finais gerados. Assim que tiver um arquivo GEOBASE.EXE poderá processar GeoBase simplesmente digitando

```
geobase
```

e RETURN.

COMPILANDO EM MEMÓRIA

A primeira coisa que o programa fará será carregar o arquivo GEOBASE.INC. Depois de aproximadamente 15 segundos, você verá uma seqüência de predicados rolar pela tela na janela Message. Depois de aproximadamente um minuto, aparecerão o menu principal e a janela. Isto é mostrado na Figura 13.7. (*Nota:* Não utilize a opção Save the Database on File ainda.)

As próximas seções deste capítulo detalham as opções apresentadas pelo menu principal.

(INTRODUÇÃO) TUTORIAL

Se você quer ver a introdução neste momento (movendo o cursor até a opção Tutorial e teclando RETURN), GeoBase vai rapidamente se referir ao arquivo GEOBASE.HLP. A primeira tela (conforme mostrado na Figura 13-8) explica o tipo de perguntas que GeoBase pode responder e a sintaxe necessária.

Você pode utilizar os comandos de teclas mostrados na parte inferior da tela ou pode utilizar a janela Editor padrão para se movimentar nesta tela Help. Aparecem mais informações à medida que você rola para baixo. Lembre-se de que mesmo esta tela é parte do código de programa. Você pode mergulhar no código para ver como as teclas de controle Editor foram implementadas em uma janela definida pelo usuário.

O restante da introdução GeoBase explica onde é mantida a informação para análise gramatical das questões em GeoBase. Isto permite que você acrescente as sua próprias regras ao conjunto ou altere as regras já existentes lá. A Figura 13-9 mostra parte desta seção.

DOS COMANDOS (COMANDOS DOS)

Esta opção no menu principal permite que você saia para o DOS e use os utilitários padrões do DOS. Da mesma forma que a opção OS no menu Files de Turbo Prolog, você volta ao programa simplesmente digitando

```
exit
```

Caso você utilize a opção Comandos DOS para voltar ao sistema operacional, verifique em qual acionador se encontra ao voltar, utilizando o comando Exit. Este é o acionador de-

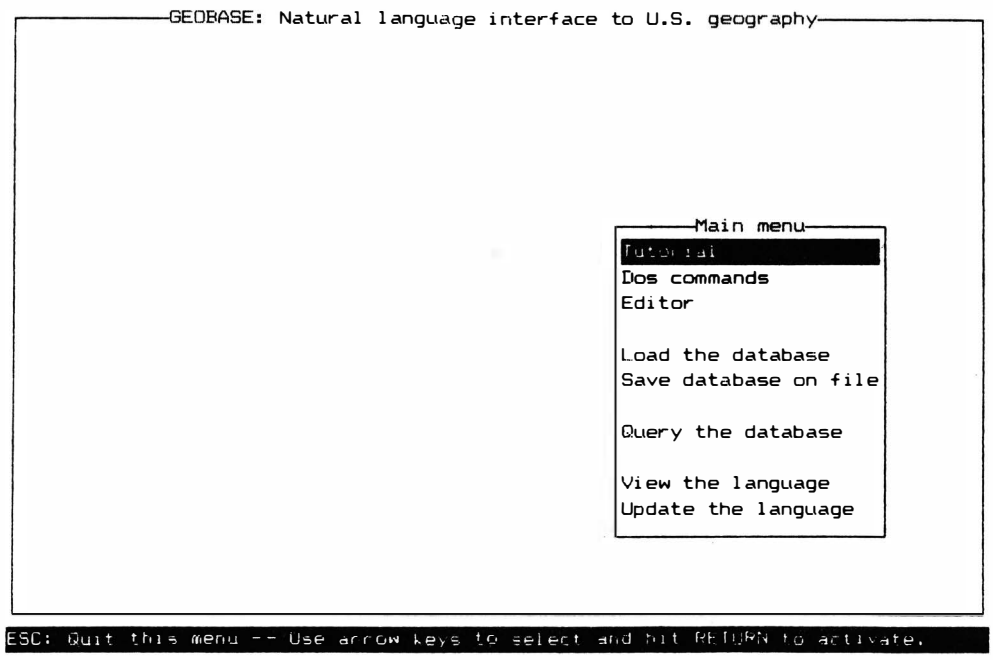


Figura 13-7 Menu e janela principal para GeoBase

fault, que Turbo Prolog pesquisa para encontrar arquivos como bancos de dados. Caso o banco de dados esteja em outro acionador de disco não será encontrado, e você terá que voltar ao DOS novamente ou retroceder ao menu princpal do Prolog para alterar o diretório.

EDITOR

Escolha este menu para obter uma janela Editor pronta para processamento de texto.

VIEW THE LANGUAGE (Ver a linguagem)

Caso escolha esta opção, você receberá outro menu, conforme mostrado na Figura 13-10. Este menu permite que você veja as definições de linguagem natural e regras já existentes no programa.

```
GEOBASE: Natural language interface to U.S. geography

                        G E O B A S E
                        *************


Geobase is a natural language interface to U.S. geography containing
the following information:

Information about states:
  Area of the state                          In square miles
  Population of the state                    In citizens
  Capital of the state
  Which states border a given state
  Rivers in the state
  Cities in the state
  Highest and lowest point in the state      In feet

Information about rivers:
  Length of river                            In miles

Information about cities:

F2:Goto line  F3:Search  S-F9:View windows  S-F10:Resize window  F10:End
```

Figura 13-8 Tela de auxflio para GeoBase

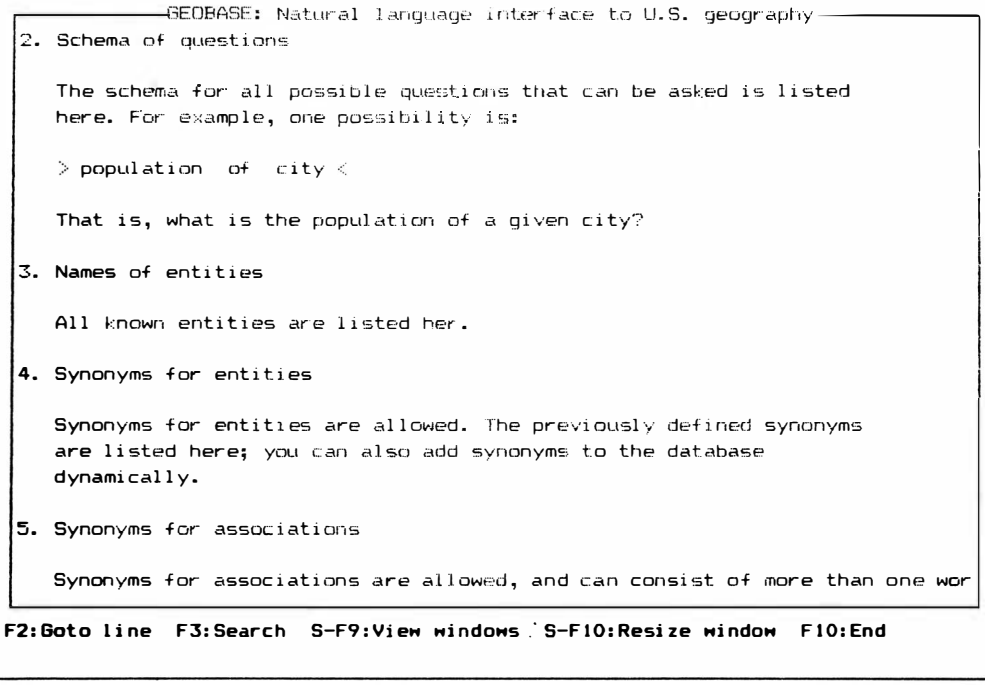


Figura 13-9 Parte da seção de análise de questões de GeoBase

UPDATE THE LANGUAGE (Atualize a linguagem)

Esta escolha oferece outro submenu (conforme indicado na Figura 13-11). Com este menu você poderá acrescentar novos sinônimos, novas alternativas de palavras associativas, ou novas palavras para serem ignoradas no texto de questões. Você pode retroceder um nível do menu por vez teclando ESC.

CARREGAR O BANCO DE DADOS

Antes de formular perguntas ao programa, é preciso carregar o banco de dados na memória. Enquanto carrega o banco de dados, prepare-se para esperar por alguma coisa além de ruído do disco. E prepare-se para uma possível declaração "out of memory" (faltou memória). Para resolver o problema de falta de memória, retroceda ao Turbo Prolog e coamece novamente, sem os programas residentes na memória.

Você não precisa carregar o banco de dados para utilizar o programa em sua

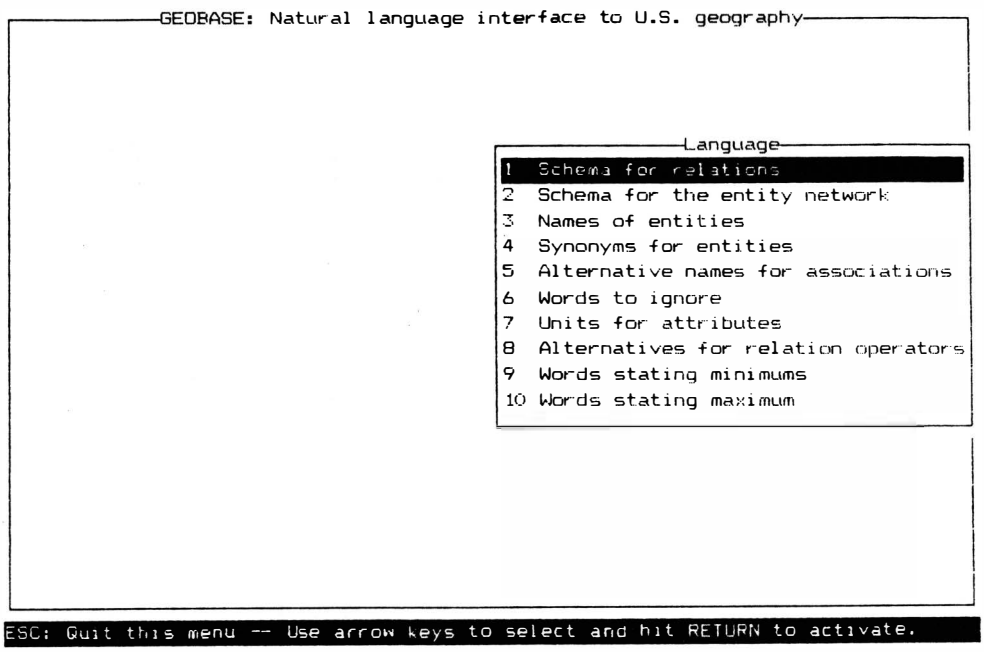


Figura 13-10 Menu apresentado pela opção View the Language (Ver a linguagem)

forma mais fundamental: fazendo perguntas. Tudo o que precisa fazer é escolher a próxima opção.

INQUIRINDO O BANCO DE DADOS

Esta escolha não requer memória adicional nem demora muito para ser processada. Assim que você teclar RETURN, ser-lhe-á solicitado fazer uma pergunta. Primeiro, tente alguns dos exemplos de perguntas que Borland sugere, conforme indicado na Figura 13-12 (estes são tomados da introdução). Como em qualquer outra posição em GeoBase, você pode voltar, teclando ESC.

GRAVANDO O BANCO DE DADOS

Tenha muito cuidado com esta opção. Caso você a escolha inadvertidamente e não tenha carregado um banco de dados para ser modificado, poderá gravar um banco de

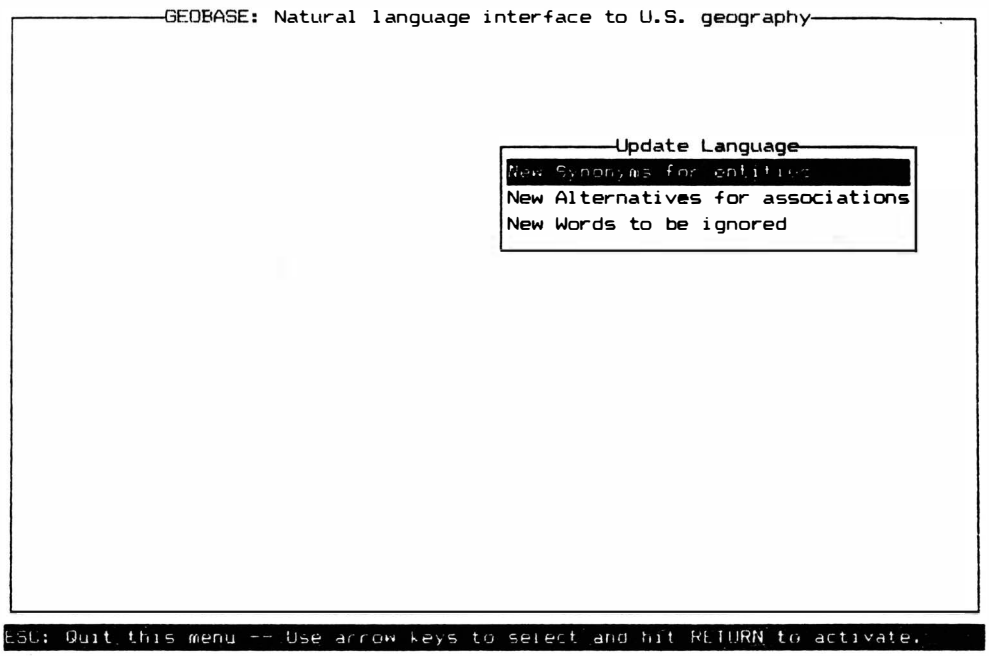


Figura 13-11 Submenu para atualizar a linguagem GeoBase

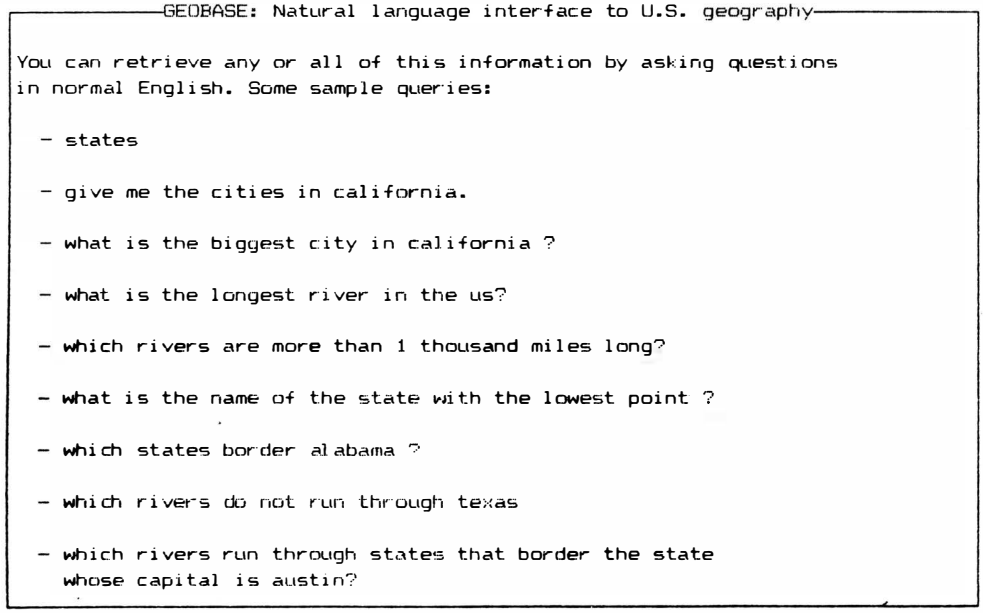
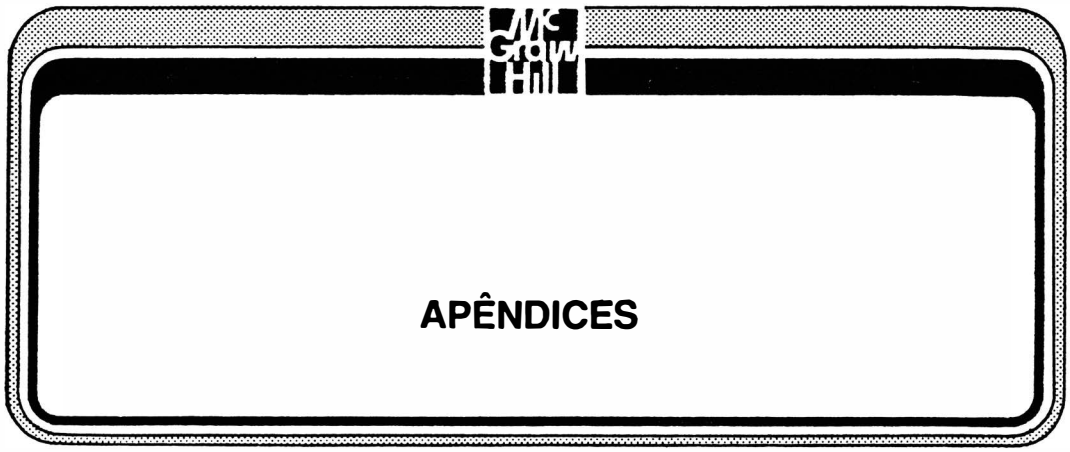


Figura 13-12 Quesitos-amostra para o banco de dados

dados que tem 0 bytes de comprimento e transformar o banco de dados anterior em um arquivo BAK.GeoBase não será capaz de processar sem os dados no banco de dados, e não vai encontrar aqueles dados se o arquivo tiver uma extensão BAK. Pior ainda, se fizer isto duas vezes, poderá cancelar completamente o banco de dados. Isto prova a utilidade dos discos de reserva.

MODIFICANDO O BANCO DE DADOS

Tente colocar questões até encontrar diversas que a linguagem não compreenda. Depois, acrescente definições às rotinas da linguagem que lhes permitam atender às perguntas. Volte novamente fazendo as mesmas perguntas e compare as respostas. Quanto você pode melhorar a capacidade de GeoBase compreender inglês?



APÊNDICES

APÊNDICE A

CÓDIGOS ASCII

Valor ASCII	Caractere	Pode ser Impresso	Efeito Especial em Turbo Prolog
1	CTRL-A	Não	Não
2	CTRL-B	Não	Não
3	CTRL-C	Não	Para execução
4	CTRL-D	Não	Não
5	CTRL-E	Não	Não
6	CTRL-F	Não	Não
7	CTRL-G	Não	Não
8	CTRL-H	Não	Retrocesso
9	CTRL-I	Não	Tabulação
10	CTRL-J	Não	Não
11	CTRL-K	Não	Não
12	CTRL-L	Não	Não
13	CTRL-M	Não	Retorno de carro
14	CTRL-N	Não	Não
15	CTRL-O	Não	Não
16	CTRL-P	Não	Ligar/desligar ECO a impressora
17	CTRL-Q	Não	Continua a mostra
18	CTRL-R	Não	Não
19	CTRL-S	Não	Pausa ou para a mostra
20	CTRL-T	Não	Liga/desliga o acompanhamento
21	CTRL-U	Não	Não
22	CTRL-V	Não	Não
23	CTRL-W	Não	Não

Valor ASCII	Caractere	Pode ser Impresso	Efeito Especial em Turbo Prolog
24	CTRL-X	Não	Não
25	CTRL-Y	Não	Não
26	CTRL-Z	Não	Sinaliza EOF (fim de arquivo)
27	ESC	Não	Escape
28	nenhum	Não	Não
29	nenhum	Não	Não
30	nenhum	Não	Não
31	nenhum	Não	Não
32	SPACE	Sim	Caractere espaço em branco
33	!	Sim	Não
34	"	Sim	Não
35	#	Sim	Não
36	$	Sim	Não
37	%	Sim	Não
38	&	Sim	Não
39	'	Sim	Não
40	(	Sim	Não
41	)	Sim	Não
42	*	Sim	Não
43	+	Sim	Não
44	,	Sim	Não
45	–	Sim	Não
46	.	Sim	Não
47	/	Sim	Não
48	0	Sim	Não
49	1	Sim	Não
50	2	Sim	Não
51	3	Sim	Não
52	4	Sim	Não
53	5	Sim	Não
54	6	Sim	Não
55	7	Sim	Não
56	8	Sim	Não
57	9	Sim	Não
58	:	Sim	Não
59	;	Sim	Não
60	<	Sim	Não
61	=	Sim	Não
62	>	Sim	Não
63	?	Sim	Não

Valor ASCII	Caractere	Pode ser Impresso	Efeito Especial em Turbo Prolog
64	@	Sim	Não
65	A	Sim	Não
66	B	Sim	Não
67	C	Sim	Não
68	D	Sim	Não
69	E	Sim	Não
70	F	Sim	Não
71	G	Sim	Não
72	H	Sim	Não
73	I	Sim	Não
74	J	Sim	Não
75	K	Sim	Não
76	L	Sim	Não
77	M	Sim	Não
78	N	Sim	Não
79	O	Sim	Não
80	P	Sim	Não
81	Q	Sim	Não
82	R	Sim	Não
83	S	Sim	Não
84	T	Sim	Não
85	U	Sim	Não
86	V	Sim	Não
87	W	Sim	Não
88	X	Sim	Não
89	Y	Sim	Não
90	Z	Sim	Não
91	[	Sim	Não
92	\	Sim	Não
93	]	Sim	Não
94	^	Sim	Não
95	_	Sim	Não
96	`	Sim	Não
97	a	Sim	Não
98	b	Sim	Não
99	c	Sim	Não
100	d	Sim	Não
101	e	Sim	Não
102	f	Sim	Não
103	g	Sim	Não
104	h	Sim	Não
105	i	Sim	Não

Valor ASCII	Caractere	Pode ser Impresso	Efeito Especial em Turbo Prolog
106	j	Sim	Não
107	k	Sim	Não
108	l	Sim	Não
109	m	Sim	Não
110	n	Sim	Não
111	o	Sim	Não
112	p	Sim	Não
113	q	Sim	Não
114	r	Sim	Não
115	s	Sim	Não
116	t	Sim	Não
117	u	Sim	Não
118	v	Sim	Não
119	w	Sim	Não
120	x	Sim	Não
121	y	Sim	Não
122	z	Sim	Não
123	{	Sim	Não
124	|	Sim	Não
125	}	Sim	Não
126	~	Sim	Não

APÊNDICE B

BIBLIOGRAFIA

TEXTO GERAL

Clocksin, W.F., and C.S. Mellish. *Programming in Prolog.* 2d ed. New York: Springer-Verlag, 1984.

Este é a bíblia da matéria. Caso você vá consultar apenas mais um livro, que seja este. Faz um excelente trabalho de passar por uma versão do Prolog, demonstrando suas características e algumas das bases teóricas.

Burnham, W. D., and A. R. Hall. *Prolog Programming and Applications.* New York: Halsted Press (division of John Wiley & Sons), 1985.

Este é o segundo livro mais importante. Apesar de ser pequeno, contém uma passagem completa pelos predicados e sintaxe do Prolog.

Bharat, B. *An Introduction to Prolog.* Summit, Pa.: TAB Books, 1986.

Este livro está escrito na forma de perguntas e respostas. Apesar de ser um pouco difícil de seguir, e de sua sintaxe estar no estilo micro-Prolog, a informação é apresentada de forma lúcida juntamente com cláusulas demonstrativas e regras. Não cobrem os produtos Prolog disponíveis para microcomputadores, informação de base a respeito de sistemas especialistas e IA, e algo a respeito da teoria e da lógica. Um extenso apêndice contém uma introdução à implementação Prolog 86 do Prolog.

MICRO-PROLOG NAS ESCOLAS

de Saram, H., *Programming in micro-Prolog*. New York: Halsted Press (division of John Wiley & Sons), 1985.

Tanto este livro como o próximo estão fortemente orientados para o micro-Prolog, a versão para microcomputador mais popular do Prolog até o aparecimento do Turbo Prolog. A sintaxe do micro-Prolog difere substancialmente daquela do Turbo Prolog, mas seus predicados fundamentais são os mesmos e têm o mesmo significado. Este livre é um tanto "divertido", com um personagem de história em quadrinhos tentando combinar cláusulas e programas-exemplo como "Fox, Chicken e Grain" (Raposa, Galinha e Trigo). Hugh de Saram traz muito de sua experiência no ensinamento do Prolog para as crianças na Inglaterra.

Ennals, R. *Iniciando micro-Prolog*. New York: Harper & Row, 1984.

Este livro também deriva da experiência de ensinar Prolog para crianças em idade escolar. Inclui respostas a exercícios e seria um ótimo texto para um ginasiano que estivesse utilizando micro-Prolog. Infelizmente, as diferenças de sintaxe entre micro-Prolog e Turbo Prolog o tornam útil principalmente como um suplemento do professor em qualquer aula que utiliza Turbo.

PROGRAMAÇÃO LÓGICA: HISTÓRIA E TEORIA

Clark, K.L., e S. A. Tarnlund, eds. *Programação Lógica*. New York: Academic Press Inc., 1982.

Este é o livro mais famoso em programação lógica. Inclui contribuições de Colemerauer (o homem que inventou Prolog) e outras autoridades em Prolog, incluindo Pereira, Kowalski e Mellish. O nível técnico é bastante elevado.

Van Caneghem, M., D.H.D. Warren, eds. *Programação Lógica e suas aplicações*. Norwood, N.J.: Ablex Publishing Corp., 1986.

Este é uma compilação de documentos de pesquisa que variam de "Otimização de Recorrência de Cauda" e "Ensinando Lógica como uma linguagem de computação nas escolas" até "Uma simulação Prolog de Tomada de Decisão Migratória em um País Menos Desenvolvido". Estão presentes a teoria e a aplicação.

Campbell, J.A., ed. *Implementações de Prolog*, New York: Halsted Press (divisão da John Wiley & Sons), 1984.

Este é um livro a respeito do estado da arte na pesquisa Prolog. Tem documentos a respeito da história da linguagem (daqueles que estavam envolvidos), assntos atuais em Prolog, questões teóricas e direcionamentos futuros.

FUNDAMENTOS GERAIS

Feigenbaum, E.A., e P. McCorduck, *A Quinta Geração: Inteligência Artificial no Desafio Computacional do Japão ao Mundo*. Reading Mass: Addison–Wesley, 1983.

A Borland diz que Turbo Prolog traz a genialidade da quinta geração e Inteligência Artificial ao seu PC. Este é o livro que apresentou ao público americano o conceito de computadores de quinta geração.

MARCAS REGISTRADAS

Apple ®	Apple Computer, Inc.
AT™	International Business Machines Corp.
GeoBase™	Borland International, Inc.
Hercules™	Hercules Computer Technology
IBM ®	International Business Machines Corp.
Macintosh™	Macintosh is a trademark of McIntosh Laboratory, Inc., licensed to Apple Computer, Inc., and is used with express permission of its owner.
MS-DOS™	Microsoft Corp.
MultiMate ®	MultiMate International Corp.
PC-DOS™	International Business Machines Corp.
ProKey™	RoseSoft, Inc.
SideKick™	Borland International, Inc.
Turbo Pascal®	Borland International, Inc.
Turbo Prolog™	Borland International, Inc.
WordStar ®	MicroPro International Corp.

ÍNDICE ANALÍTICO

Composição e Arte-Final:
JAG Composição Editorial e Artes Gráficas Ltda.
Praça F. Roosevelt, 208 - 8º andar
Tel. (011) 255-5694 - São Paulo

Impressão e acabamento
(com filmes fornecidos):
EDITORA SANTUÁRIO
Fone (0125) 36-2140
APARECIDA - SP

OUTROS LIVROS NA ÁREA

Andersen	– dBase III – Dicas e Truques®
Andersen	– PC-DOS – Dicas e Truques®
Archer	– LAN – Redes de Microcomputadores – IBM PC
Baras	– Lotus 1-2-3 Guia do Usuário 2/e. Inclui Versão 2.0
Baras	– Symphony – Guia do Usuário
Baras	– Lotus 1-2-3 Avançado
Byers	– dBase III Banco de Dados p/Todas as Aplicações
Compucenter	– MS/DOS Guia do Operador 2/e. Inclui Versões 2.0 – 2.1 – 3.0 – 3.1 – 3.2
Curtis	– WordStar – Guia do Usuário – IBM PC e Compatíveis
Fishback	– Framework – Aplicações em Finanças, Administração e Negócios
Flast	– Lotus 1-2-3 Macros
Garcia	– dBase III – Guia do Operador – Comandos Básicos
Harrison	– Framework para Principiantes
Hester	– OPEN ACCESS – Gerenciador de Informações
Hoffman	– WORD – Guia do Usuário
Hoffman	– MS/DOS Guia do Usuário 2/e. Inclui Versões 2.0 – 2.1 – 3.0 – 3.1 – 3.2
Intercorp	– Lotus 1-2-3 Guia do Operador – Inclui Versão 2.0
Jones	– dBase III – Guia do Usuário
Jones	– dBase III PLUS – Guia do Usuário
Mainis	– PRO/DOS – Guia do Usuário
Moura	– Processador de Textos no MS/DOS e UNIX
Nicholas	– Symphony Macros
Renzetti	– Turbo Pascal – Guia do Operador – Comandos Básicos
Sampaio	– SOX – Conceitos Básicos
Sheldon	– PC-DOS/MS-DOS Guia do Usuário – Inclui Versão 3.0
Shimizu	– Programação Assembler para Micros 68000, 68010 e 68020
Slemer	– Contas a Receber Contas a Pagar em dBase III PLUS
Slemer	– Bancos e Fluxo de Caixa em dBase III PLUS
Slemer	– Contabilidade e Ativo Fixo em dBase III PLUS
Zuccolo	– WordStar – Guia do Operador – IBM PC e Compatíveis
Zuccolo	– WordStar CP/M – MS/DOS – Dicas e Truques
Wood	– SuperCalc – Guia do Usuário – SuperCalc2, SuperCalc3, SuperCalc4
Wood	– SuperCalc3 e SuperCalc4 – Guia do Operador – Comandos Básicos

0-07-450377-4