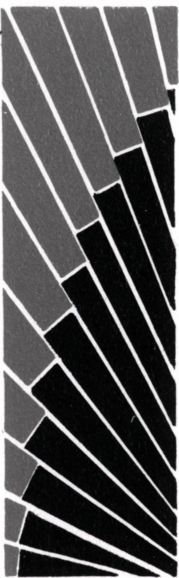
SOX

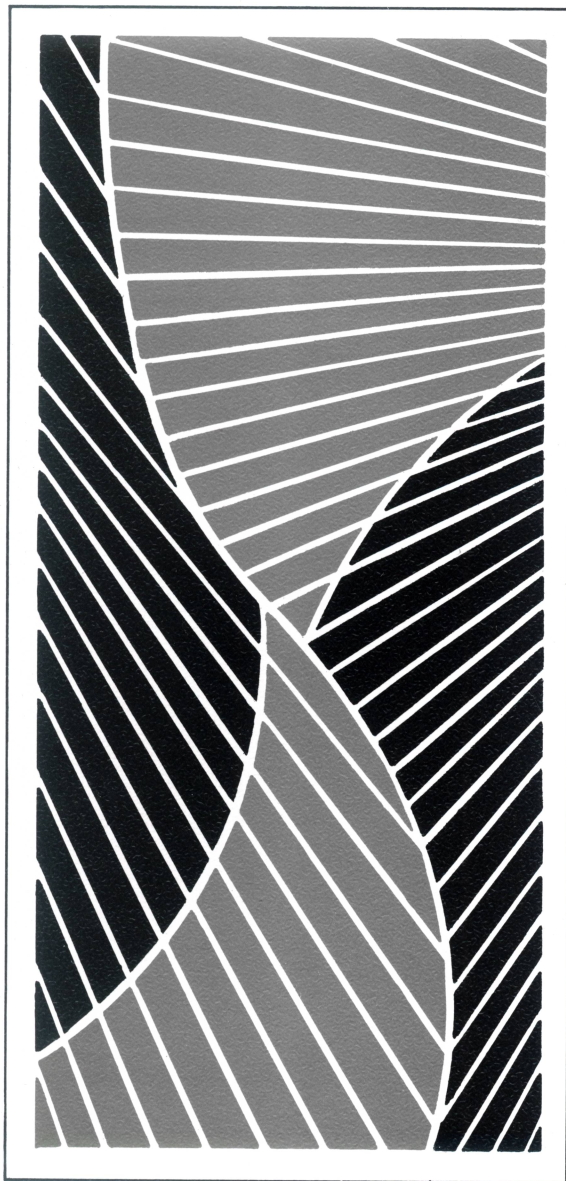
Manual de Operação

CURSES/TERMINFO/SOX



cobra

SOX



cobra

Computadores e Sistemas Brasileiros S/A

MANUAL DE OPERAÇÃO CURSES-TERMINFO/SOX

UX0189-02.0

MANUAL
DE OPERAÇÃO
CURSES-TERMINFO/SOX

2ª Edição

abril/89

UX0189-02.0

Este manual é uma publicação da Divisão de Testes e Integração de Sistemas / Setor de Documentação, aplicável a produtos comercializados pela Cobra.

O Fabricante reserva-se o direito de alterar a qualquer tempo, sem notificação prévia, as características e especificações técnicas do produto retratado nesta documentação, bem como promover, sempre que necessário, alterações e correções na forma e conteúdo desta publicação.

Cobra - Computadores e Sistemas Brasileiros S.A.

Todos os direitos reservados. Esta obra não pode ser comercializada, nem ter qualquer uso diferente do autorizado, nem pode também ser reproduzida, total ou parcialmente, por qualquer meio ou forma, sem prévia autorização escrita concedida pela Cobra.

Av. Comandante Guaranys, 447 - Jacarepaguá

22785 Rio de Janeiro RJ BRASIL

REGISTRO DE ATUALIZAÇÃO

Manual:	Emissão:
Operação CURSES-TERMINFO/SOX	abril/1989

Esta página é uma orientação ao usuário quanto à identificação e aplicação específica deste manual, sua estrutura e o histórico das atualizações aplicáveis ao mesmo. Mantenha esta folha anexa ao início do manual, para controle de sua atualização.

APLICAÇÃO DO MANUAL:

Equipamento: Linha X

Sistema: SOX

Produto: SOX

CONTROLE DE PÁGINAS DO MANUAL:

Módulo 1:

Cap. 1 - 6 págs.
Cap. 2 - 30 págs.
Cap. 3 - 26 págs.
Cap. 4 - 53 págs.
Glossário - 35 págs.
Cap. 5 - 7 págs.
Cap. 6 - 4 págs.
Cap. 7 - 50 págs.
Apêndice A - 12

Módulo 2:

Cap. 1 - 6 págs.
Cap. 2 - 16 págs.
Cap. 3 - 4 págs.
Cap. 4 - 38 págs.

Módulo 3:

Cap. 1 - 3 págs.
Cap. 2 - 3 págs.
Cap. 3 - 190 págs.
Apêndice - 5 págs.
Apêndice B - 23 pags.

HISTÓRICO DAS ATUALIZAÇÕES:

1 - Esta documentação encontra-se em sua segunda edição, em função de uma revisão minuciosa na primeira edição.

2 - O parâmetro el do descritor do terminal TI 300, modo ANSI teve o seu valor alterado para " E OK", para corrigir erro em sua definição (ver Capítulo 1 do Módulo 2).

   Esta correção está disponível nas versões posteriores à A.00 para o SOX-PC e SOX-500 e A.01 para o X-10.

SISTEMA DE ATUALIZAÇÃO DE DOCUMENTAÇÃO:

Com o objetivo de manter os usuários sempre a par das constantes evoluções e melhorias de seus produtos, a COBRA dispõe de um Sistema de Atualização de Documentação (**SAD**), que consiste em um conjunto de arquivos de texto reunidos em um ou mais discos flexíveis denominados **altera-doc**.

Estes discos, que são fornecidos com o sistema operacional SOX, contêm as informações necessárias para a atualização dos manuais disponíveis para a linha de computadores X/SOX, enquanto a documentação correspondente ainda não foi atualizada ou reeditada.

Os arquivos do SAD pertencem ao diretório **/man/altera** e são de dois tipos distintos: os arquivos de atualização, que contêm as alterações a serem aplicadas a cada manual, e um arquivo **índice**, que identifica cada arquivo de atualização existente no diretório.

Os arquivos que descrevem as atualizações de cada manual são formados por blocos de informações, cada um dos quais contêm as alterações liberadas numa mesma

data. E dividido em duas partes: a primeira, com informações gerais e um sumário das alterações aplicáveis ao manual, e a segunda, com as alterações propriamente ditas, precedidas de informações que possibilitam sua localização no manual.
Já o arquivo **índice**, que deve ser o primeiro a ser consultado, contém as seguintes informações:

- No cabeçalho: Número seqüencial e data de emissão das atualizações. Indica também qual o sistema e linha de produtos a que o SAD se aplica.

- Títulos dos manuais disponíveis para essa linha de produto. Observar, entretanto, que alguns dos manuais listados referem-se a produtos de software opcionais, não fornecidos com o pacote básico do SOX.

- Códigos dos manuais correntemente em vigor, e para os quais se aplicam as atualizações emitidas. Através desse código, o usuário pode conferir se as edições de seus manuais são as mais atualizadas. Qualquer problema nesse sentido poderá ser esclarecido junto ao representante ou filial COBRA mais próximo.

- Nomes dos arquivos, tal como constam no diretório **/man/altera**, que contêm as atualizações aplicáveis a cada manual. Estes nomes são os que deverão ser informados ao sistema, para acesso a cada um dos mesmos.

- Número seqüencial e data de emissão de cada atualização já liberada para os manuais indicados. Observar que estas atualizações são sempre complementares para uma mesma edição do manual, ou seja, uma atualização mais recente não invalida ou inclui uma

atualização anterior.

O SAD é um mecanismo de atualização ágil e dinâmico, mas de caráter transitório. À medida que as atualizações do SAD vão sendo incorporadas às novas revisões e edições dos respectivos manuais, ou novas alterações se façam necessárias a uma documentação já liberada, os arquivos de atualização dos discos **altera-doc** vão sendo alterados correspondentemente, dando origem a novas versões do SAD.

A implantação e uso deste sistema se faz através de um comando especial do SOX, denominado **am**, que é descrito no "Manual de Referência Comandos SOX" (SX01).

O propósito deste manual é apresentar as facilidades oferecidas pelo pacote CURSES/TERMINFO no desenvolvimento de programas, implementando uma interface amigável que manipula janelas em um terminal, de forma otimizada e independente do tipo de terminal em uso.

Com o intuito de facilitar a consulta às informações, este manual foi dividido em três módulos:

MÓDULO 1 - PROGRAMANDO COM CURSES...

Neste primeiro módulo, os assuntos abordados apresentam-se da seguinte forma:

Capítulo 1

Apresenta os recursos, facilidades e características do CURSES.

Capítulo 2

Introduz o primeiro programa edit1. Trata-se de um editor básico de texto baseado em tela ("full-screen"), com poucos recursos, mas extremamente fácil de se usar, servindo perfeitamente o propósito de introduzir os conceitos básicos do CURSES (o programa usa uma única janela).

Capítulo 3

Adiciona funcionalidades ao editor edit1 para gerar um novo editor, edit2. A funcionalidade adicional foi escolhida com o objetivo de apresentar alguns conceitos mais avançados do CURSES (remoção de texto, inserção e deleção de linhas, rolamento da janela, atributos de vídeo, acentuação de caracteres).

Capítulo 4

Apresenta o editor edit3, com o objetivo de introduzir os recursos multijanelas do CURSES.

Capítulo 5

Apresenta conceitos finais sobre o CURSES.

Capítulo 6

Apresenta as rotinas do TERMINFO - uma biblioteca de funções usada pelo CURSES - para atingir sua independência de terminal. Tais rotinas também podem ser usadas diretamente nos seus programas.

Capítulo 7

Informa como compilar seus programas, discute como executar o programa final e fornece uma listagem completa dos editores edit1, edit2 e edit3, prontos para uso.

Apêndice A

Apresenta as principais diferenças existentes entre o padrão ANS e o Kernighan e Ritchie (K&R) para a linguagem C.

Glossário

Além de explicar as expressões que possam oferecer dificuldade ao usuário, referencia as mesmas quanto à sua localização no texto.

MÓDULO 2 - CONHECENDO O USO DO TERMINFO...

Estrutura-se conforme descrito a seguir:

Capítulo 1

Detalha o formato-fonte das descrições de terminais, a sintaxe dos nomes do terminal e as características do TERMINFO.

Capítulo 2

Apresenta as tabelas contendo as características do TERMINFO.

Capítulo 3

Classifica as características do TERMINFO.

Capítulo 4

Descreve o conjunto das características booleanas, numéricas e do tipo cadeia.

MÓDULO 3 - AS FUNÇÕES DAS BIBLIOTECAS CURSES/TERMINFO/TERMCAP

Compõe-se de:

Capítulo 1

Apresenta o uso das rotinas e descreve o conceito de janelas.

Capítulo 2

Classifica as rotinas quanto à sua utilização.

Capítulo 3

Descreve as rotinas das bibliotecas CURSES/TERMINFO/TERMCAP.

Compõe ainda o manual, dos Apêndices A e B, que apresentam as tabelas ASCII e ABICOMP e os Arquivos de Inclusão, respectivamente..

Pág.

Módulo 1 - Programando com CURSES...

CAPÍTULO 1 - APRESENTANDO O CURSES...

1.1	Panorama do CURSES	1-3

CAPÍTULO 2 - ELABORANDO UM PROGRAMA...

2.1	O Editor de Tela Simplificado edit1	2-1
2.2	A Estrutura Geral de edit1. . .	2-5
2.3	A Inicialização e o Encerramento do Programa	2-10
2.4	O Trecho Intermediário do Programa	2-21

CAPÍTULO 3 - SOFISTICANDO O PROGRAMA...

3.1	O Editor de Tela Aperfeiçoado - edit2	3-1
3.2	A Implementação de edit2 . . .	3-6
3.2.1	Implementação dos Comandos . .	3-6
3.2.2	Acentuação	3-20
3.2.3	Atributos de Vídeo	3-24

CAPÍTULO 4 - RECURSOS MULTIJANELAS

4.1	Novos Recursos do Editor edit3 .	4-1
4.1.1	Chamada do Editor Sem Nome de Arquivo	4-2
4.1.2	Inclusão de um Novo Arquivo de Texto durante a Edição	4-4
4.1.3	Execução de um Shell durante a Edição	4-4

Pág.

4.1.4	Operações com Blocos	4-4
4.2	Implementação do Editor - Modo Normal	4-10
4.2.1	A Região de Mensagens.	4-10
4.2.2	Chamando o Editor Sem Nome de Arquivo	4-16
4.2.3	Incluindo um Novo Arquivo no Texto	4-21
4.2.4	Executando um Shell durante a Edição	4-22
4.3	Implementação do Editor - Modo Bloco	4-26
4.3.1	Destaque do Bloco - As Alternativas	4-26
4.3.2	A Estrutura de Controle do Bloco	4-30
4.3.3	Definição e Deleção de um Bloco.	4-32
4.3.4	Operações com Blocos	4-38

CAPÍTULO 5 - RECURSOS ADICIONAIS

5.1	Controle da Entrada	5-1
5.2	Atributos de Vídeo	5-5
5.3	Janelas Dinâmicas (Pads)	5-6

CAPÍTULO 6 - BIBLIOTECA TERMINFO

CAPÍTULO 7 - COMPILANDO E EXECUTANDO UM PROGRAMA

7.1	Compilação	7-1
7.2	Execução	7-2
7.3	Descrição do Programa edit1 . .	7-3
7.4	Descrição do Programa edit2 . .	7-14
7.5	Descrição do Programa edit3 . .	7-27

Pág.

Apêndice A - Diferenças do Padrão ANS para K&R

Glossário

Módulo 2 - Conhecendo o TERMINFO...

CAPÍTULO 1 - APRESENTANDO O TERMINFO...

1.1 Sintaxe Geral 1-2
1.2 Sintaxe dos Nomes do Terminal . 1-2
1.3 Resumo das Características . . 1-3
1.4 Exemplo 1-5

CAPÍTULO 2 - CARACTERIZANDO O TERMINFO...

2.1 Características em Ordem de Nome de Variável 2-1
2.2 Características em Ordem de Nome TERMINFO 2-7
2.3 Características em Ordem de Nome TERMCAP 2-12

CAPÍTULO 3 - CLASSIFICANDO AS CARACTERÍSTICAS...

CAPÍTULO 4 - DESCREVENDO AS CARACTERÍSTICAS...

4.1 Características Booleanas . . . 4-1
4.2 Características Numéricas . . . 4-8
4.3 Características do Tipo Cadeia . 4-10
4.3.1 Tratamento de Parâmetros 4-11

Pág.

Módulo 3 - Funções das Bibliotecas CURSES/ /TERMINFO/TERMCAP

CAPÍTULO 1 - APRESENTANDO AS ROTINAS...

CAPÍTULO 2 - CLASSIFICANDO AS ROTINAS...

CAPÍTULO 3 - DESCREVENDO AS ROTINAS...

3.1 Rotinas da Biblioteca CURSES . . 3-1
3.2 Rotinas da Biblioteca TERMINFO . 3-160
3.3 Rotinas da Biblioteca TERMCAP. . 3-181

Apêndice A - Tabelas ASCII e ABICOMP

Apêndice B - Arquivos de Inclusão

Computadores e Sistemas Brasileiros S/A.

MÓDULO 1

Programando com CURSES ...

Página intencionalmente em branco.

CAPÍTULO 1

APRESENTANDO O CURSES...

O CURSES é uma biblioteca de sub-rotinas, ligada com seu programa. Este pode usar quaisquer das mais de cem sub-rotinas oferecidas pela biblioteca. Não se assuste, porém, com o número de rotinas disponíveis: seu uso é bastante simples, e este manual traz explicações bem objetivas.

As características principais do CURSES serão descritas no final deste capítulo. Por enquanto, basta dizer que as rotinas oferecem um meio de definir várias janelas (áreas da tela) de tamanhos variáveis, sobrepostas umas às outras, caso se queira. Estas janelas podem ser manipuladas de várias formas (adição de texto, movimentação na tela, definição de atributos de vídeo, etc.). Surge, pois, a questão: por que usar o CURSES? O que há de tão maravilhoso com este pacote? São três as razões principais:

- Primeiro, a própria funcionalidade do pacote (a manipulação de janelas) diminui tremendamente seu esforço de programação.

- A segunda razão diz respeito à velocidade de atualização da tela. Em sistemas multiusuários, os diversos terminais conectam-se à UCP através de uma porta serial, funcionando normalmente a uma velocidade de 9.600 bauds. A esta velocidade, aproximadamente mil caracteres podem ser transferidos para o terminal, por segundo. A tela inteira (por exemplo, 24 linhas por oitenta colunas) é atualizada em dois segundos. Muito lento, não ? Aí entra o CURSES. Suas rotinas de manipulação de tela acessam o terminal de forma

otimizada. Por exemplo, você pode, no seu programa, dizer: "Mude tais e tais coisas de tal janela, e tais outras desta segunda janela." Por enquanto, a tela do terminal não mudou. Agora, seu programa diz: "Atualize a tela com todas as modificações que mandei fazer." As rotinas de otimização do CURSES entram em ação e atualizam a tela com um mínimo de comunicação para o terminal, e usando todos os artifícios possíveis (seu terminal pode, por exemplo, remover caracteres ou linhas por **hardware**, sem a necessidade de redesenhar toda uma linha ou uma parte da tela). O CURSES atua.

. A terceira razão pela qual o uso do CURSES simplifica sua vida é, talvez, a mais importante. Trata-se da **independência de terminal**. A manipulação dos terminais de cada fabricante é diferente. A forma de movimentar o cursor (para dar um exemplo simples) não é igual em cada um dos terminais, mesmo de um único fabricante. Será que você vai precisar ter uma versão do programa para cada fabricante e, pior, talvez mais de uma versão para um fabricante que tenha mais de um tipo de terminal? O CURSES resolve este problema. Não importa que terminal está sendo usado para rodar seu software. O CURSES descobre como manipulá-lo (utilizando a biblioteca TERMINFO) e o faz, sem que seu programa tenha que saber dos detalhes. O CURSES usa a biblioteca **TERMINFO** para obter informações sobre a manipulação do terminal.

1.1 PANORAMA DO CURSES

Trata-se de uma biblioteca de funções para a manipulação de janelas e telas, totalmente compatível com a biblioteca **libcurses** do sistema UNIX®, System V. As funções da biblioteca são ligadas com um programa aplicativo escrito na linguagem C.

O CURSES permite a manipulação de **janelas.** Uma janela é uma área retangular contendo texto (caracteres). Cada janela tem uma posição na tela do terminal. Quando o CURSES é inicializado, uma janela-padrão (chamada **stdscr**) já existe e pode ser usada imediatamente. Janelas adicionais, de qualquer tamanho (inclusive maiores que a tela do terminal) podem ser criadas, manipuladas e destruídas. A forma normal de manipular uma janela, uma vez criada, é ler informação do terminal através da janela, adicionar texto à mesma, usando rotinas-padrão, e, uma vez o texto da janela pronto, pedir seu desenho na tela do terminal.

O texto adicionado a uma janela pode ser destacado com **atributos de vídeo**, tais como vídeo reverso, piscante, intensificado, etc. Caracteres acentuados da língua portuguesa podem ser lidos do terminal e colocados numa janela, como qualquer outro caractere. A leitura de caracteres do teclado processa teclas especiais, tais como setas de posicionamento, teclas de função, etc.

As rotinas de manipulação geral de janelas incluem:

. Atualização da janela na tela;

. Criação de novas janelas;

. Movimentação de uma janela na tela;

- Criação de subjanelas (porções de outras janelas), que são manipuladas posteriormente como janelas;

- Destruição de janelas;

- Manipulação de janelas especiais, maiores que a tela do terminal.

A adição de informação nas janelas permite as seguintes operações, entre outras:

- Adição de um caractere;

- Adição de uma cadeia de caracteres;

- Manipulação dos atributos de vídeo, destacando qualquer parte da janela;

- Apagamento de áreas da janela;

- Desenho de uma borda na janela;

- Remoção de caracteres da janela;

- Inserção e remoção de linhas inteiras da janela;

- Inserção de um caractere entre dois caracteres da janela, afastando o texto à direita;

- Movimentação do cursor da janela (cada janela tem seu próprio cursor);

- Sobreposição de uma janela a outra;

- Impressão formatada (como printf) em uma janela;

- Rolamento de uma parte da janela.

Uma informação pode ser lida através de uma janela, na posição corrente do cursor desta janela, das seguintes formas:

. Obtenção de um único caractere, lido do teclado, com ou sem eco na janela;

. Leitura de caracteres especiais (teclas de função, por exemplo) do teclado;

. Obtenção de uma cadeia de caracteres através da janela;

. Leitura do texto já presente na janela, em qualquer posição da mesma;

. Leitura formatada (como scanf) através de uma janela.

Várias outras rotinas existem, para manipular o terminal. Elas serão discutidas no Módulo 3.

Acreditamos que só há uma maneira de aprender a usar o CURSES: examinando programas reais que o usem. Apresentamos, neste manual, três programas completos que usam o CURSES. São programas pequenos (409, 465 e 961 linhas, incluindo comentários), mas reais, completos, funcionando, e até úteis.

Estamos supondo que você conhece a linguagem C. Nenhum comentário será feito para explicar construções da mesma. Nossos programas foram escritos segundo o padrão Kernighan e Ritchie, sendo usadas desta forma, construções que são aceitas por todo e qualquer compilador seguindo este padrão. Entretanto, nada impede que estes programas sejam compilados e executados com sucesso em compiladores que sigam o padrão ANS, como é o caso, por exemplo, do compilador C/SOX. Caso você deseje modificar os programas, a fim de utilizar construções mais específicas do padrão ANS, consulte o Anexo A, Diferenças do Padrão ANS para o K&R, e faça as alterações necessárias para a sua aplicação.

CAPÍTULO 2

ELABORANDO UM PROGRAMA...

Como já mencionamos no capítulo anterior, a melhor forma de aprender a usar o CURSES é programando. Começando com programas simples, e elaborando programas cada vez mais envolvidos, até cobrir todas as rotinas do CURSES. Comecemos, portanto, neste capítulo, com um programa introdutório, cuja finalidade é apresentar os conceitos do CURSES que são usados em qualquer programa.

2.1 O EDITOR DE TELA SIMPLIFICADO edit 1

Escolhemos como primeiro programa um editor de tela (**full-screen**) extremamente simples. Este editor será programado completamente neste capítulo, e o seu código completo aparece no item 7.3. Para apresentar a funcionalidade do editor, fornecemos o procedimento abaixo. Você deve estudá-lo com muito cuidado, pois constitui a especificação do programa que queremos escrever.

NOME

edit1 - Editor de tela simplificado

SINOPSE

edit1 arquivo

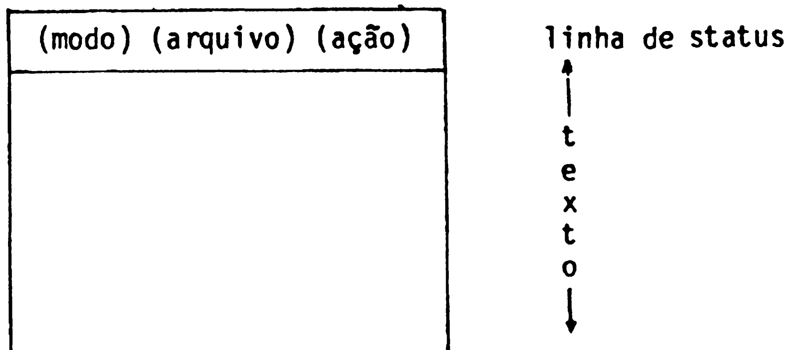
DESCRIÇÃO

Editl é um editor de tela permitindo a edição de textos compostos de exatamente uma tela de texto. Como editor de tela, ele permite que você posicione o cursor em qualquer lugar da tela e digite seus textos exatamente como aparecerão no arquivo. Quando é executado, o editl lê o arquivo fornecido como argumento e o apresenta na tela do terminal. O editor tem cinco comandos de movimentação do cursor:

CONTROLE S - Movimenta uma coluna à esquerda;
CONTROLE D - Movimenta uma coluna à direita;
CONTROLE E - Movimenta uma linha para cima;
CONTROLE X - Movimenta uma linha para baixo;
RETORNO DO CARRO - Movimenta para o início da próxima linha.

Observe que as teclas S, D, E e X formam uma cruz no seu teclado e indicam o sentido do movimento do cursor. Ao entrar no editor, o modo de operação é INSERÇÃO. Neste modo, caracteres digitados são inseridos antes do cursor, e os caracteres sob e à direita do cursor são afastados à direita. O último caractere da linha é perdido. O outro modo de operação chama-se "ADIÇÃO". Neste modo, os caracteres digitados são sobrepostos aos caracteres presentes na tela. O modo de operação é mudado com o comando CONTROLE V. Caracteres não-imprimíveis não são aceitos.

A organização da tela do editor edit1 é a seguinte:



A linha de status contém três itens. O modo pode ser INSERÇÃO ou ADIÇÃO. O nome do arquivo aparece no meio da linha de status. O lugar chamado "ação" é reservado para o comando CONTROLE A. Ao digitar este comando, a seguinte pergunta é feita na linha de status:

Sai(s) ou Grava(g) ?

Se você responder "s", o edit1 abandona a edição. Se digitar "g", o edit1 grava no arquivo o texto mostrado na tela e volta à edição. Para gravar e sair, digite:

CONTROLE A CONTROLE G CONTROLE A CONTROLE S

Na gravação, os espaços em branco finais de cada linha são removidos.

Para que você entenda perfeitamente a funcionalidade do editor, sugerimos que digite o programa que está apresentado no item 7.3, num arquivo chamado **edit1.c**. Compile agora este programa, usando o seguinte comando:

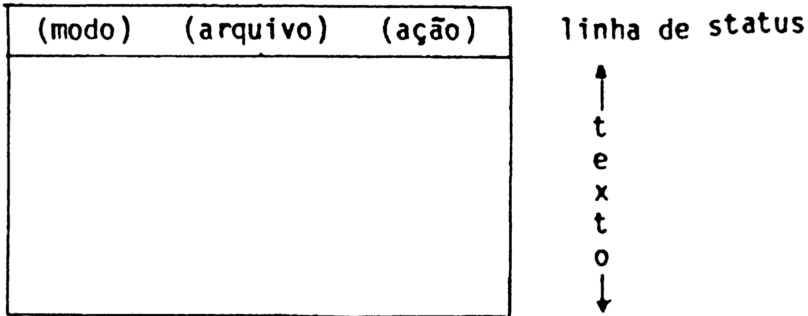
```
cc edit1.c -lcurses -o edit1
```

A opção "-lcurses" faz a ligação do programa com a biblioteca CURSES. O Capítulo 7 fornece mais detalhes sobre a compilação e execução de programas. Uma vez compilado, trabalhe bastante com o programa, procurando familiarizar-se com seus mínimos detalhes de operação. Tente digitar caracteres ilegais (CONTROLE B, por exemplo), tente mover o cursor fora dos limites da janela, etc., para ver o comportamento do edit1 em todos os casos. Quando você estiver bem familiarizado com o editor, volte à leitura deste manual. Antes de proceder ao treino, entretanto, tome cuidado com o seguinte ponto: **não** edite arquivos seus já existentes, pois o editor só consegue editar arquivos de até uma tela. Se você gravar em cima do arquivo, apenas as primeiras 23 linhas (para uma tela de 24 linhas) serão gravadas, destruindo assim o resto do arquivo.

2.2 A ESTRUTURA GERAL DE edit1

Já que você voltou à leitura do manual, estamos supondo que está familiarizado com o uso do editor. Vamos agora projetar sua estrutura e escrevê-lo.

A organização da tela implementada por edit1 é mostrada a seguir:



Precisamos escolher os objetos do CURSES que serão usados para representar esta organização de tela. Lembre que o CURSES oferece o conceito de "janelas", correspondendo a áreas retangulares da tela. Temos, na nossa tela, duas áreas retangulares principais: a linha de status e a área de texto. Poderíamos usar uma janela para representar cada uma destas áreas. Entretanto preferimos, neste item, usar uma única janela para representar a tela inteira. Isto não significa que esta escolha seja a melhor, mas simplesmente que queremos apresentar os recursos mais simples do CURSES, usando uma única janela. Quando seu programa utiliza uma única janela para representar a tela inteira, não há necessidade de criá-la explicitamente. O CURSES já fornece esta janela, e ela se chama **stdscr** (**standard screen** - tela-padrão).

Pronto! A primeira decisão sobre a estrutura do programa já foi tomada: usaremos **stdscr**. Precisamos pensar, agora, sobre a estrutura de dados que representará o texto digitado. A escolha de uma estrutura de dados sempre deve seguir os mesmos três passos: primeiro, deve-se identificar os **objetos** que devem ser representados pela estrutura de dados. Segundo, as **operações** que serão aplicadas aos objetos devem ser identificadas. Finalmente, uma **estrutura** para representar os objetos deve ser escolhida, de forma que as operações identificadas no segundo passo possam ser implementadas fácil e eficientemente.

Neste caso, o objeto sob consideração é uma tela de texto, consistindo em várias linhas, cada linha composta de vários caracteres. Se você examinar a especificação do editor formada no início deste capítulo poderá identificar as seguintes operações aplicadas ao texto:

- Leitura do texto do arquivo do usuário, para colocá-lo na estrutura representando o texto.

- Movimentação pelo texto (horizontal e verticalmente) de uma coluna ou linha de cada vez.

- Adição de um novo caractere digitado na posição onde estamos atualmente no texto.

- Inserção de um novo caractere digitado na posição onde estamos atualmente no texto, afastando os caracteres à direita de uma posição.

- Recuperação de todo o texto de volta e sua gravação no arquivo do usuário, eliminando primeiro os espaços em branco finais de cada linha.

Simples, não é? Agora, qual é a estrutura de dados que você escolheria para representar o texto, de forma a permitir estas cinco operações? Temos, então, a parte mais criativa , a de elaborar a estrutura, para suportar as operações. Neste caso, não precisamos de muita criatividade para ver que podemos representar o texto, usando uma matriz retangular de caracteres e duas variáveis para manter a posição atual horizontal e vertical no texto. Com esta estrutura, as cinco operações descritas anteriormente seriam implementadas da seguinte forma:

. Repita até o fim do arquivo, ou até encher a matriz;
  - Leia uma linha do arquivo do usuário;
  - Coloque o texto lido na linha correspondente da matriz.

. Mude as duas variáveis de posicionamento.

c) Coloque o caractere lido na posição da matriz indicada pelas duas variáveis de posicionamento. Avance de uma coluna a variável de posicionamento horizontal.

. Afaste uma posição para a direita os caracteres sob e à direita (na mesma linha) da posição indicada pelas variáveis de posicionamento, e introduza o novo caractere na brecha aberta. Avance a variável de posicionamento horizontal de uma coluna.

e) Repita da primeira à última linha da matriz;
   - Localize a última posição não-branca da linha;
   - Varra a linha do início à posição localizada acima e grave os caracteres no arquivo do usuário;
   - Coloque um caractere de fim de linha no arquivo.

Nossa estrutura deve estar adequada porque conseguimos implementar as cinco operações facilmente. Não precisamos montar a estrutura de dados que acabamos de descrever. A janela **stdscr** já não é uma matriz retangular contendo texto? Ela já não tem variáveis de posicionamento (cada janela do CURSES tem um cursor indicando a posição corrente)? Então, se **stdscr** já tiver tudo isso, podemos dispensar uma estrutura adicional, desde que tenhamos uma forma (com rotinas do CURSES) de implementar as cinco operações necessárias, lembrando que, agora, nossa matriz é a própria janela **stdscr**. Como veremos em mais detalhes adiante, as seguintes rotinas do CURSES serão usadas para implementar as operações:

. mvaddstr - Adiciona uma cadeia de caracteres à janela;

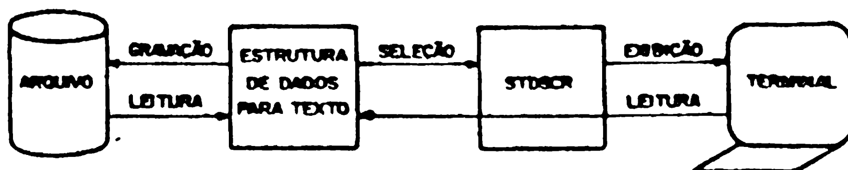
. move - Move o cursor da janela;

. addch - Adiciona um caractere na janela;

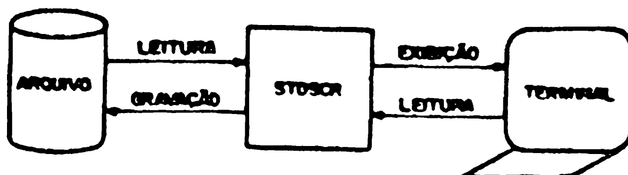
. insch - Insere um caractere na janela;

. mvinch - Obtém um caractere já presente na janela.

Na realidade, criamos o editor edit1 justamente para que uma estrutura de dados adicional não fosse necessária. Afinal, queremos estudar o CURSES, e não como escrever um editor de textos. Um editor de textos profissional precisaria, necessariamente, de uma estrutura de dados adicional para armazenar o texto, pois a janela só seria usada para apresentar, na tela, **parte** do texto de cada vez. Resumindo, um editor de textos profissional seria organizado conforme ilustra a figura abaixo:



O editor edit1, entretanto, usa o seguinte fluxo de informação:



2.3 A INICIALIZAÇÃO E O ENCERRAMENTO DO PROGRAMA

Podemos dividir o código do editor edit1 em três partes importantes: a fase de inicialização, a fase de interação com o usuário e a fase de encerramento do programa. A primeira e a última fases, discutidas neste item, somam aproximadamente a metade do código do editor. A fase de inicialização deve cuidar de quatro aspectos importantes:

1) Inicialização do CURSES;

2) Inicialização do terminal;

3) Leitura do arquivo do usuário na janela **stdscr**;

4) Inicialização da linha de status.

Vamos agora criar a estrutura da rotina principal **main**, que controla o resto do programa.

Dividimos os quatro passos de inicialização em mais de uma rotina, pois eles não estão inter-relacionados. Os passos de inicialização do CURSES e do terminal foram agrupados na mesma rotina **inicializa**. Esta rotina também verifica a sintaxe do comando. O terceiro passo de inicializaçao - a leitura do arquivo do usuário - ocorre na rotina **le_arq**. A inicialização da linha de status é feita na rotina **comandos**. Esta rotina também lê os comandos do usuário e contém o laço principal do programa. O resultado é a rotina **main**, mostrada a seguir (os números à esquerda indicam a linha do programa, no item 7.3):

```
7     #include <curses.h>
55    /*
56     * Rotina principal
57     * Inicializa o curses
58     * Le o arquivo do usuario
59     * Interpreta os comandos do usuario
60     * sai, fornecendo o status do comando
61     */
62
63    main( argc, argv )
64    int    argc;
65    char   *argv[];
66    {
67
68          if( inicializa( argc, argv ) == ERR ) { /* nao inicializou */
69              exit( 1 );
70          } else if( le_arq( argv[0], argv[1] ) == ERR /* nao leu */
71              || comandos( argv[1] ) == ERR ) {  /* erro nos comandos */
72
73              tchau( 1 );     /* nao retorna */
74          } else {
75              tchau( 0 );     /* nao retorna */
76          }
77    }
```

Observe que quando a rotina **inicializa** acha algum erro (na sintaxe do comando, como veremos logo em seguida), ela retorna o valor ERR, que significa "erro". Quase todas as rotinas do CURSES retornam o valor ERR, se há problema. Caso contrário, retornam o valor OK. A rotina **inicializa** é nossa, não do CURSES, mas estamos aproveitando a mesma convenção para a indicação de erro. As rotinas **le_arq** e **comandos** fazem a mesma coisa. Os valores ERR e OK estão definidos no arquivo **curses.h**.

Este arquivo deve ser incluído em qualquer programa que use o CURSES. Além de definir os valores de ERR e OK, o **curses.h** contém detalhes importantes para o uso do CURSES, que serão introduzidos à medida que aparecerem.

Uma segunda observação sobre a rotina **main** diz respeito ao encerramento do programa, que se dá na linha 69, com uma chamada a **exit** ou à rotina **tchau** (linhas 73 e 75). A diferença entre estas duas formas de encerramento consiste em que, no primeiro caso, o CURSES não terá sido inicializado, enquanto que, no outro, ele precisa ser desativado, antes do término do programa com **exit**.

A rotina **inicializa** é mostrada a seguir:

```
  8    #include <signal.h>

...

 94    /*
 95     * inicializa
 96     * verifica a sintaxe do comando
 97     * inicializa curses e o terminal
 98     */
 99
100    inicializa( argc, argv )
101    int  argc;
102    char *argv[];
103    {
104
```

```
105         if( argc != 2 ) {
106             fprintf(stderr, "Sintaxe: %s arquivo\n", argv[0] );
107             return( ERR );
108         } else {  /* tudo OK */
109             signal( SIGINT, SIG_IGN );
110             initscr();
111             noecho();
112             cbreak();
113             nl();
114             noxon();
115         }
116         return( OK );
117     }
```

Se a sintaxe do comando estiver correta, a rotina ignora o sinal SIGINT (uma outra alternativa seria de capturá-lo e chamar **tchau**). Em seguida, o CURSES é inicializado com a rotina **initscr**. Esta rotina sempre deve ser chamada antes de qualquer outra rotina do CURSES. Ela desempenha basicamente três funções:

- Descobre qual é o terminal que está sendo usado e as suas características (maiores detalhes sobre este passo serão apresentados posteriormente no Módulo 2);

- Inicializa o terminal;

- Cria a janela **stdscr**.

Initscr inicializou **stdscr**, atribuindo a esta janela algumas características importantes, uma das quais é a de ecoar na janela qualquer caractere lido com a rotina **getch**. No editor editl, porém, não queremos que todo e qualquer caractere digitado pelo usuário seja ecoado na janela. Não faria sentido, por exemplo, ecoar os caracteres CONTROLE A e CONTROLE V. Para desligar esta característica, chamamos **noecho**.

Vamos passar agora à inicialização do terminal. Já dissemos que **initscr** inicializa o terminal. Por que inicializar de novo? Na realidade, o terminal foi inicializado com algumas características (eco de caracteres desligado, por exemplo), mas não com todas as que queremos. Tal como foi inicializado, o sistema operacional SOX só enviará os caracteres digitados no terminal quando <RETURN> for digitada. Os terminais no SOX funcionam assim, por default. No editor editl, queremos receber os caracteres assim que forem digitados. Isto significa "colocar o terminal no modo CBREAK". Daí, a rotina **cbreak** do CURSES, chamada na linha 112. De forma semelhante, os caracteres CONTROLE S e CONTROLE Q são tratados de forma especial pelo SOX (de acordo com o protocolo XON-XOFF). Como queremos receber o caractere CONTROLE S no editl, temos que avisar o SOX para não tratá-lo de forma especial. Usamos a rotina **noxon** do CURSES, na linha 114, para fazer isso. Finalmente, a rotina **nl** termina a inicialização do terminal. Esta rotina diz ao SOX para transformar ' r' (retorno de carro) em ' n' (nova linha) na entrada, e transformar ' r' e ' n' na seqüência ' r' ' n', na saída. Fizemos isso, porque não sabemos se sua tecla RETURN gera ' r' ou ' n'. Se você quisesse desligar este mapeamento e tratar ' r' e ' n' de forma diferente no seu programa, teria que chamar a rotina **nonl**.

A rotina **tchau**, mostrada abaixo, encerra o CURSES com a rotina **endwin** e encerra o programa. **Endwin** tem o objetivo principal de voltar com as características normais do terminal. Esta rotina **deve** ser chamada, antes de encerrar o programa.

```
80   /*
81    * tchau
82    * sai, fornecendo o status do sistema
83    */
84
85   tchau( status )
86   int      status; /* status do programa */
87   {
88
89        endwin();      /* encerra o curses*/
90        exit( status );
91   }
```

Pronto! O CURSES está inicializado, e a janela **stdscr** existe e está vazia (isto é, cheia de brancos). Note bem que a tela do terminal ainda não mudou, apenas a janela **stdscr**. Vamos completar agora a janela com o texto contido no arquivo do usuário através da rotina **le_arq**, mostrada a seguir.

```
13   /* definicao de constantes e macros */
14
15   #define MAXCOLUNA   132       /* maior tela aceitavel */
16
17   /* posicao inicial do texto */
18   #define INI_Y  1    /* linha inicial do texto */
19   #define INI_X  0    /* coluna inicial do texto */
...
120  /*
121   * le_arq
122   * abre e le o arquivo do usuario
```

```
123      * colocando o texto na janela stdscr
124       */
125
126     le_arq( prog, arquivo )
127     char        *prog;          /* nome do programa */
128     char        *arquivo;       /* nome do arquivo do usuario */
129     {
130
131         char            *fgets();
132         register int    lin;        /* para varrer as linhas */
133         FILE            *fp;        /* arquivo aberto para leitura */
134         int             tam;        /* tamanho da leitura */
135         char            buf[MAXCOLUNA + 2]; /* buffer de leitura */
136
137         tam = min( MAXCOLUNA, COLS + 2 );  /* +2 por causa do \n
138                                             * e do \0 finais
139                                             */
140         if( (fp = fopen(arquivo,access(arquivo,0) >= 0 ? "r" : "w+")) ==
141                                 (FILE *)NULL ) {
142             fprintf(stderr, "%s: nao pode abrir %s\n", prog, arquivo );
143             return( ERR );
144         }
145
146         for( lin = INI_Y; lin < LINES; lin++ ) {
147             if( fgets( buf, tam, fp ) == (char *)NULL ) {
148                 if( feof( fp ) ) {       /* fim do arquivo */
149                     break;
150                 } else {        /* erro de leitura */
151                     fprintf(stderr, "%s: erro de leitura em %s\n",
152                                             prog, arquivo );
153                     return( ERR );
154                 }
155             }
156             mvaddstr( lin, 0, buf );   /* coloca o texto na janela */
157         }
158
159         fclose( fp );
160         return( OK );
161     }
```

Esta rotina é muito simples. Ela abre o arquivo do usuário para leitura, se ele existe, ou cria-o, se não existe. Em seguida, para cada linha da janela onde queremos colocar texto (da linha INI_Y até a linha LINES-1), uma linha é lida do arquivo com **fgets** (que faz parte da biblioteca padrão da linguagem C) e adicionada à janela com **mvaddstr**, uma rotina do CURSES. Passemos às explicações mais detalhadas.

As linhas e as colunas de uma janela iniciam-se com 0. Portanto, o canto superior esquerdo de **stdscr** é "linha 0, coluna 0" (ou (0,0)). Como não queremos colocar texto na linha de status, o texto começa na posição (INI_Y, INI_X) = (1,0). Quando **initscr** inicializou o CURSES, ela descobriu qual era o tamanho do seu terminal (e, portanto, o tamanho de **stdscr**) e inicializou as variáveis LINES e COLS, para indicar o número de linhas e colunas, respectivamente. Ambas as variáveis estão definidas em **curses.h**.

Para colocar uma cadeia de caracteres presente no buffer **buf** num lugar da janela, por exemplo (lin, col), podemos fazer o seguinte:

```
move( lin, col );
addstr( buf );
```

A rotina **move** move o cursor da janela (não do terminal!), e a rotina **addstr** adiciona a cadeia fornecida pelo argumento à janela, a partir da posição atual do cursor. As duas rotinas acima podem ser combinadas pela seguinte macro, definida em **curses.h**:

```
mvaddstr( lin, col, buf );
```

Falta então inicializar a linha de status, o que fazemos logo no início da rotina **comandos**:

```
 21     /* modos de operacao */
 22     #define INSERCAO    0
 23     #define ADICAO      1

...

 41     /* regioes da linha de status */
 42     #define COL_ACAO    (COLS - 30)     /* regiao "acao" */
 43     #define COL_ARQUIVO (COLS/2 - 7)    /* regiao "arquivo" */
 44     #define COL_MODO    1               /* regiao "modo" */

...

164     /*
165      * comandos
166      * inicializa o cursor, a linha de status e o modo de operacao
167      * interpreta os comandos do usuario
168      */
169
170     comandos( arquivo )
171     char      *arquivo;      /* nome do arquivo do usuario */
172     {
173
174          int       modo;          /* modo de operacao */

...

180          modo = INSERCAO;         /* modo inicial de operacao */
181          sta_arq( arquivo );
182          sta_modo( modo );

...
```

```
226     }
227
228
229     /*
230      * sta_arq
231      * exibe o nome do arquivo na linha de status
232      * (somente o ultimo componente, porque o nome completo
233      * pode ser grande)
234      */
235
236     sta_arq( arquivo )
237     char      *arquivo;      /* nome do arquivo do usuario */
238     {
239
240         char      *ult_nome();
241         int       lemb_y, lemb_x; /* para voltar o cursor */
242
243         getyx( stdscr, lemb_y, lemb_x );
244         mvprintw( 0, COL_ARQUIVO, "%14s", ult_nome( arquivo ) );
245         move( lemb_y, lemb_x );
246     }
247
248
249     /*
250      * sta_modo
251      * exibe o modo de operacao na linha de status
252      */
253
254     sta_modo( modo )
255     int       modo;      /* o modo de operacao */
256     {
257
258         int       lemb_y, lemb_x;     /* para voltar o cursor */
259
260         getyx( stdscr, lemb_y, lemb_x );
261         mvprintw( 0, COL_MODO, "%10s", modo == INSERCAO ? "INSERCAO" :
262                                                             " ADICAO" );
263         move( lemb_y, lemb_x );
264     }
```

E agora, algumas explicações de fecho para o assunto inicialização. As variáveis **lemb_y** e **lemb_x** das rotinas **sta_arq** e **sta_modo** são usadas para lembrar a posição do cursor antes de imprimir alguma informação na linha de status. A macro **getyx** coloca nos seus argumentos a posição atual do cursor da janela. A rotina **printw** funciona exatamente como a rotina **printf**. Finalmente, tal como no caso de **mvaddstr**, a rotina **mvprintw** é equivalente à rotina **move** seguida da rotina **printw**.

2.4 O TRECHO INTERMEDIÁRIO DO PROGRAMA

A seguir, a apresentação da rotina **comandos**, por inteiro:

```
  9    #include <ctype.h>

...

 25    #define CTRL(c)      ((c) - 'A' + '\001')
 26    #define min(x, y)    ((x) < (y) ? (x) : (y))
 27
 28    /* comandos do editor */
 29    #define ACAO         CTRL('A')
 30    #define MUDAMODO     CTRL('V')
 31    #define ESQUERDA     CTRL('S')
 32    #define DIREITA      CTRL('D')
 33    #define CIMA         CTRL('E')
 34    #define BAIXO        CTRL('X')
 35    #define PROXLINHA    ('\n')

...

 49    /* o que fazer apos uma acao */
 50    #define CONTINUA     0    /* continua a editar */
 51    #define FIM          1    /* termina o programa */

...

164    /*
165     * comandos
166     * inicializa o cursor, a linha de status e o modo de operacao
167     * interpreta os comandos do usuario
168     */
169
```

```
170     comandos( arquivo )
171     char       *arquivo;      /* nome do arquivo do usuario */
172     {
173
174         int        modo;          /* modo de operacao */
175         int        pos_y, pos_x;  /* posicao atual do cursor */
176         caractere cmd;            /* caractere digitado pelo usuario */
177         int        resultado;     /* acao deu erro ? */
178
179         move( INI_Y, INI_X );
180         modo = INSERCAO;          /* modo inicial de operacao */
181         sta_arq( arquivo );
182         sta_modo( modo );
183
184         while( TRUE ) {
185             getyx( stdscr, pos_y, pos_x );
186             refresh();
187             switch( cmd = getch() ) {
188             case ACAO:
189                 if( acao( arquivo, &resultado ) == FIM ) {
190                     return( resultado );
191                 }
```

```
192                 break;
193             case MUDAMODO:
194                 modo = modo == INSERCAO ? ADICAO : INSERCAO;
195                 sta_modo( modo );   /* mostra novo modo */
196                 break;
197             case ESQUERDA:
198                 movimenta( pos_y, pos_x - 1 );
199                 break;
200             case DIREITA:
201                 movimenta( pos_y, pos_x + 1 );
202                 break;
203             case CIMA:
204                 movimenta( pos_y - 1, pos_x );
205                 break;
206             case BAIXO:
207                 movimenta( pos_y + 1, pos_x );
208                 break;
209             case PROXLINHA:
210                 movimenta( pos_y + 1, 0 );
211                 break;
212             default:
213                 if( isprint( cmd ) ) {  /* da para imprimir ? */
214                     if( modo == ADICAO ) {
215                         addch( cmd );
216                     } else {
217                         insch( cmd );
218                         movimenta( pos_y, pos_x + 1 );
219                     }
220                 } else {
221                     beep();
222                 }
223                 break;
224             }
225         }
226     }
```

Examine a linha 179. A primeira coisa a fazer, antes de entrar no laço principal de processamento, é inicializar o cursor da janela na posição (1,0). As próximas três linhas inicializam a linha de status, conforme mostrado no item 2.3. O resto da rotina consiste em um laço (TRUE está definido em "curses.h") do qual só podemos sair na linha 190, como resultado de alguma ação (CONTROLE A). A linha 185 obtém a posição atual do cursor, para podermos saber onde localizá-lo após os comandos. A linha 187 obtém um caractere do usuário (rotina **getch**) através da janela **stdscr.** Por que obter um caractere **através** de uma janela? E que alguns atributos da janela são importantes para a leitura. Já vimos um destes atributos: um caractere lido através de uma janela pode ser ecoado ou não na janela. Outros atributos serão vistos no decorrer do texto. Vale então a observação: um caractere sempre é lido do terminal, através de alguma janela. No caso da rotina **getch**, a janela usada é **stdscr** (não podemos explicar ainda por que o resultado de **getch** é do tipo "caractere" (linha 176)).

A rotina **refresh** da linha 186 é muito importante. Quando um texto é adicionado à janela (como fizemos com as rotinas **mvaddstr** e **mvprintw**), o conteúdo da janela não é imediatamente mostrado na tela. Para fazer com que a tela do terminal fique igual à janela **stdscr**, chamamos a rotina **refresh.** Observe que não fizemos uma atualização da tela com **refresh** após cada chamada a **mvaddstr** e **mvprintw.** O CURSES consegue otimizar melhor a atualização da tela, quando todo o texto que queremos mostrar na mesma estiver pronto na janela. Se você refletir um pouco, descobrirá que, normalmente, devemos chamar **refresh** apenas quando queremos aceitar alguma entrada do usuário. Neste caso, deve-se olhar para a tela do terminal e detectar o texto mais atualizado possível, para escolher o que se vai digitar.

Por isto, chamamos **refresh** apenas duas vezes no programa (linhas 186 e 335), imediatamente antes da rotina **getch**, que espera até o usuário digitar um caractere. Esta é a forma padrão de usar **refresh** no CURSES.

Na rotina **comandos**, só falta discutirmos o que deve ser feito quando o usuário digita algo no terminal. A mudança de modo de operação através do comando CONTROLE V (linhas 193 a 196) simplesmente muda a variável **modo** e atualiza a linha de status. Os comandos de posicionamento (linhas 197 a 211) calculam a nova posição do cursor e o movimentam. A rotina **movimenta** é usada ao invés de **move**, para centralizar a verificação de movimentações ilegais:

```
267    /*
268     * movimenta
269     * movimenta o cursor da janela, verificando casos ilegais
270     */
271
272    movimenta( y, x )
273    int       y;
274    int       x;   /* posicao desejada */
275    {
276
277         if( y < INI_Y ||     /* tentando entrar na linha de status ? */
278             move( y, x ) == ERR ) {   /* caindo fora da janela ? */
279               beep();
280               return( ERR );
281         }
282         return( OK );
283    }
```

Beep é uma rotina do CURSES que faz a campainha do terminal soar, avisando o usuário de que o último comando dado não funcionou.

Quando o usuário digita caracteres normais, eles devem ser adicionados à janela. Primeiro, o programa verifica se o caractere é visível (linha 213) e faz a campainha soar, se não for (linha 221). Se o caractere puder ser impresso, ele é adicionado à janela. A forma de adicioná-lo à janela depende do modo de operação. No modo ADIÇÃO, a rotina **addch** do CURSES é usada. Esta rotina adiciona o caractere à janela, na posição atual do cursor da janela, e, automaticamente, avança o cursor uma posição à direita. No modo INSERÇÃO, a rotina **insch** é usada. **Insch** insere o caractere na posição atual do cursor da janela, afastando de uma posição os caracteres à direita. O último caractere da linha é perdido. Como a rotina **insch** não movimenta o cursor da janela, nosso programa tem que fazê-lo (linha 218).

Finalmente, se o usuário digitar o comando de ação (CONTROLE A), a rotina **acao** é chamada. Esta rotina deve retornar dois valores. Primeiro, queremos saber se a ação deve encerrar o programa, ou se a edição deve continuar após a ação. Para indicar isso, a rotina **acao** retorna o valor FIM (encerra) ou CONTINUA (não encerra). Se o valor for FIM, a rotina **comandos** deve retornar para **main**, devolvendo o valor OK, se tudo estiver OK, ou ERR, em caso de erro na ação. **Resultado** sempre será igual a OK, quando **acao** retornar FIM. Entretanto escolhemos esta interface mais flexível, para facilitar mudanças posteriores ao programa.

```
286     /*
287      * acao
288      * pergunta ao usuario o que ele quer fazer (sair ou gravar)
289      * e desempenha a acao
290      */
291
292     acao( arquivo, ap_res )
293     char *arquivo;      /* nome do arquivo do usuario */
294     int  *ap_res;       /* onde colocar o resultado: OK ou ERR */
295     {
296
297         caractere     resp;     /* resposta do usuario */
298         int           ret;      /* valor de retorno da funco acao() */
299
300         *ap_res = OK;       /* supoe que nao havera erro */
301         ret = CONTINUA;
302         resp = pergunta( "Sai(s) ou Grava(g) ? ");
303         switch( PADRONIZA( resp ) ) {
304         case ACAO_SAI:
305             ret = FIM;
306             break;
307         case ACAO_GRAVA:
308             if( (*ap_res = grava( arquivo )) == ERR ) {
309                 pergunta( "Erro de grav. Tecle algo. ");
310             }
311             break;
312         default:
313             beep();         /* usuario se arrependeu ou errou */
314             break;
315         }
316
317         return( ret );
318     }
```

Examine o código com cuidado. A rotina primeiro pergunta o que o usuário quer fazer e obtém a resposta (linha 302). A resposta é mudada ligeiramente (linha 303) pelo programa, de forma a permitir a digitação de uma letra minúscula, maiúscula ou de controle, para dizer a mesma coisa. Se a ação for 's' (sai), precisamos apenas retornar o código FIM. Se for 'g' (grava), a rotina **grava** é chamada e é esta rotina que diz se houve erro ou não. Havendo erro, o usuário é informado e deve digitar alguma coisa, para continuar. No que diz respeito ao CURSES, nada de novo na rotina **acao.**

Examine agora a rotina **pergunta**:

```
321     /*
322      * pergunta
323      * faz uma pergunta e retorna a resposta (um caractere)
324      */
325
326     pergunta( perg )
327     char        *perg;
328     {
329
330             caractere       resp;
331             int             lemb_y, lemb_x;       /* para voltar o cursor */
332
333             getyx( stdscr, lemb_y, lemb_x );
334             mvaddstr( 0, COL_ACAO, perg );
335             refresh();
336             resp = getch();
337             move( 0, COL_ACAO );
338             clrtoeol();
339             move( lemb_y, lemb_x );
340             return( resp );
341     }
```

Duas observações são importantes: mais uma vez, usamos um **refresh** imediatamente antes do **getch**. Segundo, ao sair da rotina, devemos apagar a mensagem que foi exibida ao usuário. Isto é feito com a rotina **clrtoeol** do CURSES, que preenche com brancos a linha atual da janela, a partir da posição do cursor.

A rotina **grava**, finalmente, encarrega-se de gravar, no arquivo do usuário, a informação presente na janela.

```
344    /*
345     * grava
346     * grava o texto no arquivo do usuario
347     */
348
349    grava( arquivo )
350    char      *arquivo;      /* nome do arquivo do usuario */
351    {
352
353         int            ret; /* valor de retorno da funcao grava() */
354         register  int  lin; /* para varrer as linhas da janela */
355         register int   col; /* para varrer as colunas da janela */
356         int            col_final;  /* coluna final a gravar na linha */
357         FILE           *fp;        /* arquivo aberto para gravacao */
358         int            lemb_y, lemb_x; /* para voltar o cursor */
359
360         if( (fp = fopen( arquivo, "w")) == (FILE *)NULL ) {
361              fprintf(stderr, "Nao pode gravar em %s\n", arquivo );
362              return( ERR );
363         }
364
365         getyx( stdscr, lemb_y, lemb_x );
366         ret = OK;
367         for( lin = INI_Y; ret == OK && lin < LINES; lin++ ) {
368              /* procura a ultima coluna que interessa nesta linha */
369              for( col_final = COLS -1;
370                   col_final >= 0 && mvinch( lin, col_final ) == ' ';
371                   col_final-- )
372                   ;
```

```
373
374             for( col = 0; col <= col_final; col++ ) {
375                 if( fputc( mvinch( lin, col ), fp ) == EOF ) {
376                     ret = ERR;
377                     break;
378                 }
379             }
380
381             if( fputc( '\n', fp ) == EOF ) {
382                 ret = ERR;
383             }
384         }
385
386         if( fclose( fp ) == EOF ) {
387             ret = ERR;
388         }
389         move( lemb_y, lemb_x );
390         return( ret );
391     }
```

Para fazer isto, precisamos de uma rotina que nos indique o que está na janela. A rotina **inch** retorna o caractere presente na janela, na posição do cursor (desta janela). A macro

```
mvinch( lin, col );
```

é equivalente a:

```
move( lin, col );
inch();
```

Concluímos nossa descrição do editor edit1. O próximo capítulo vai acrescentar muita funcionalidade ao editor, com o intuito de introduzir novas rotinas do CURSES.

CAPÍTULO 3

SOFISTICANDO O PROGRAMA...

Neste capítulo vamos adicionar funcionalidade ao editor edit1, para gerar um novo editor, que chamamos de edit2. Você deve se lembrar que a nova funcionalidade adicionada ao editor foi escolhida com o intuito de apresentar novas rotinas do CURSES, e não com o objetivo de mostrar o que um editor de tela deve fazer. Seguiremos os mesmos passos usados no capítulo anterior: apresentação da nova funcionalidade do editor, discussão geral sobre a implementação desta funcionalidade e, finalmente, uma discussão detalhada do código resultante, destacando sempre as rotinas do CURSES. O código completo do editor edit2 aparece no item 7.4.

3.1 O EDITOR DE TELA APERFEIÇOADO - edit2

O que mais falta ao editor edit2, como editor, é a facilidade de editar textos de mais de uma tela. Esta adição, entretanto, não envolveria novas rotinas do CURSES e foi descartada por esta razão. Edit2 permanece um editor de uma única tela. O que vamos adicionar é funcionalidade, para melhor manipular o texto desta tela. Para tanto, vamos acrescentar os seguintes comandos ao editor:

1. Zerar até o fim do arquivo.

   Este comando zera todo o texto presente na tela, a partir da posição corrente do cursor, até o fim da tela. A posição do cursor não é alterada. O comando é ativado com as teclas CONTROLE F ("F" significa "zera até o Fim").

2. Zerar todo o texto.

   Este comando reinicializa o texto sendo editado, exatamente como se o arquivo estivesse sendo editado do zero. O cursor é então posicionado no início da área de textos da tela, isto é, na posição (1,0). O comando é ativado com as teclas CONTROLE T ("T" significa "zera Tudo").

3. Zerar até o fim da linha.

   Este comando zera todo o texto presente na linha, a partir da posição corrente do cursor, até o fim da linha corrente. A posição do cursor não é alterada. O comando é ativado com as teclas CONTROLE B ("B" significa "Branqueia até o fim da linha").

4. Remover um caractere.

   Este comando remove o caractere presente na posição atual do cursor. Todos os caracteres que estavam à direita do cursor são afastados uma posição à esquerda. Um caractere branco (espaço) é introduzido na última coluna da linha atual. A posição do cursor não é alterada. O comando é ativado com as teclas CONTROLE C ("C" significa "remove Caractere").

5. Remover uma linha.

   Este comando remove a linha atual (isto é, a linha do cursor). Todas as linhas presentes abaixo do cursor são movidas uma linha para cima. Uma linha contendo espaços em branco é introduzida na última posição da tela. A posição do cursor na tela não é alterada. O comando é ativado com as teclas CONTROLE L ("L" significa "remove uma Linha").

6. Inserir uma linha.

   Este comando afasta todas as linhas da tela uma linha para baixo, a partir da linha do cursor, até a penúltima linha da tela. A última linha da tela é perdida. Uma linha contendo espaços em branco é introduzida na posição do cursor. A posição do cursor na tela não é alterada. O comando é ativado com as teclas CONTROLE I ("I" significa "Insira uma linha").

7. Rolar o texto.

   Dois comandos efetuam o rolamento do texto na tela. Nenhum deles afeta a posição do cursor. O comando CONTROLE W efetua um rolamento do texto uma linha para baixo, na área de textos da tela. A última linha da tela é perdida. Uma linha em branco é introduzida no início do texto. O comando CONTROLE Z efetua o rolamento para cima. Ele opera de forma simétrica ao comando CONTROLE W.

8. Remover o caractere à esquerda do cursor.

   Este comando movimenta o cursor uma posição à esquerda e atua como o comando CONTROLE C. Observe que nada acontecerá com este comando, se o cursor estiver posicionado originalmente na primeira coluna de uma linha. O comando deve ser ativado com a tecla de apagamento de caractere que o usuário utiliza normalmente no SOX. Se esta tecla entrar em conflito com outro comando de edit2, o comando de remoção do caractere à esquerda do cursor não estará disponível.

9. Movimentar o cursor com as teclas de seta.

   Se o terminal dispuser de teclas de setas, tais teclas e mais as teclas CONTROLE E, D, E, e X podem ser usadas para movimentar o cursor.

10. Redesenhar a tela.

    Este comando apaga e redesenha a tela inteira, sem mudar o conteúdo do texto ou a posição do cursor. O comando pode ser útil quando a tela fica "suja", devido, por exemplo, a uma mensagem recebida de um usuário em outro terminal. O comando é ativado com as teclas CONTROLE R.

Aí estão nossos novos comandos. Você terá algumas dúvidas a respeito dos mesmos, estas serão certamente esclarecidas, na descrição da implementação de cada comando.

Nosso editor edit2 apresenta duas características adicionais que não são comandos propriamente ditos. Primeiro, caracteres acentuados da língua portuguesa devem ser aceitos na entrada e exibidos na tela, se o terminal que você está usando possibilitar a geração e exibição de tais caracteres. Segundo, o modo de

operação do editor (INSERÇÃO ou ADIÇÃO) deve aparecer na linha de status com algum destaque, por exemplo, vídeo reverso, se o terminal possuir tais atributos de vídeo.

Sugerimos agora, mais uma vez, que você se familiarize com este novo editor de tela. Digite o programa contido no item 7.4, compile-o e exercite todos os seus comandos. Volte à leitura do manual, quando você estiver completamente familiarizado com o uso do programa.

3.2 A IMPLEMENTAÇÃO DE edit2

A estrutura geral do editor edit2 é idêntica à do editor edit1. As mudanças necessárias são, na realidade, **novos** trechos de código acrescidos ao editor edit1. Nenhuma porção de edit1 deve ser refeita ou desfeita, sinal de que temos uma implementação limpa. Vamos, pois, à implementação.

3.2.1 IMPLEMENTAÇÃO DOS COMANDOS

Os comandos CONTROLE F (zera até o **F**im), CONTROLE B (**B**ranqueia até o fim da linha), CONTROLE C (remove o **C**aractere sob o cursor), CONTROLE L (remova a **L**inha do cursor) e CONTROLE I (**I**nsira uma linha na posição do cursor) são implementados com uma única rotina do CURSES. Veja a seguir:

```
36 #define ZERAATEFIM     CTRL('F')
37 #define ZERALINHA      CTRL('B')
38 #define REM_CARAC      CTRL('C')
39 #define REM_LINHA      CTRL('L')
40 #define INS_LINHA      CTRL('I')

...

184 comandos( arquivo )
185 char       *arquivo;     /* nome do arquivo do usuario */
186 {
```

```
...

230            case ZERAATEFIM:
231                clrtobot();
232                break;
233            case ZERALINHA:
234                clrtoeol();
235                break;
236            case REM_CARAC:
237                delch();
238                break;
239            case REM_LINHA:
240                deleteln();
241                break;
242            case INS_LINHA:
243                insertln();
244                break;

...

280 }
```

As novas rotinas do CURSES que estamos introduzindo são, portanto:

clrtobot: "clear to bottom"
clrtoeol: "clear to end of line"
delch: "delete character"
deleteln: "delete line"
insertln: "insert line"

A descrição do efeito destas rotinas pode ser tomada da descrição dos comandos correspondentes do edit2, que já fornecemos acima. Observe que estas rotinas atuam sobre a janela **stdscr.** Como antes, as modificações só serão refletidas na tela, quando a rotina **refresh** for chamada. As rotinas **wclrtobot**,**wclrtoeol**, **wdelch**, **wdeleteln** e **winsertln** também estão disponíveis e atuam sobre qualquer janela, como veremos no próximo capítulo.

Passemos ao comando CONTROLE T (zera Tudo). Este comando poderia ser implementado da seguinte forma:

```
move( INI_Y, INI_X );
clrtobot();
```

Entretanto decidimos apresentar uma forma alternativa de implementação, que nos fornece um meio de introduzir uma nova rotina do CURSES. Eis a segunda alternativa:

```
41  #define ZERATUDO      CTRL('T')
...
184 comandos( arquivo )
185 char      *arquivo;     /* nome do arquivo do usuario */
186 {
...
245           case ZERATUDO:
246               clear();
247               sta_arq( arquivo );
248               sta_modo( modo );
249               move( INI_Y, INI_X );
250               break;
...

280 }
```

A rotina do CURSES **clear** apaga a janela **stdscr** por inteiro, isto é, a janela é preenchida completamente com brancos. Já que nossa linha de status faz parte da janela **stdscr**, ela será também apagada. Devemos, portanto, redesenhá-la (linhas 247 e 248). Acreditamos que esta implementação alternativa seja inferior à primeira alternativa proposta (por questão de clareza e efeitos resultantes). Ela tem a vantagem, porém, de ser mais rápida, como teremos ocasião de explicar no Capítulo 4.

Passemos, então, aos comandos de rolamento de tela. As rotinas do CURSES usadas para o rolamento de janelas atuam sobre o que é chamado de **Região de Rolamento.** Esta região inclui algumas ou todas as linhas de uma janela, conforme configurado pela rotina **setscrreg.** Por exemplo, após a chamada da rotina:

```
setscrreg( 5, 14 )
```

a região de rolamento da janela **stdscr** incluiria as linhas 5 a 14, inclusive. A figura abaixo ilustra a situação:

Observe que a rotina **setscrreg** apenas **define** a região de rolamento da janela **stdscr.** O rolamento do texto é feito com a rotina **nscroll.** Esta rotina recebe como parâmetro o número de linhas a rolar. Por exemplo, a chamada:

```
nscroll( 1 );
```

rola o texto da região de rolamento **uma** linha para cima. De forma mais clara, as linhas 0 a 4 e de 15 ao fim da tela não são afetadas. O texto da linha 5 é substituído pela linha 6, e assim sucessivamente, na região de rolamento. A linha 14 fica em branco. Para fazer rolamento no sentido contrário (movendo o texto para baixo), basta chamar a rotina **nscroll**, passando como parâmetro o negativo do número de linhas a rolar. Como observação final, a rotina **scroll** é equivalente a **nscroll(1)**. Uma última observação com respeito às rotinas de rolamento: se o cursor estiver dentro da região de rolamento quando a rotina é chamada, ele também é rolado com o texto, desde que a posição final do mesmo ainda esteja na região de rolamento. Caso contrário, a posição do cursor não é afetada.

De posse destas rotinas do CURSES, a implementação dos comandos CONTROLE W e CONTROLE Z é simples. Os trechos correspondentes de código aparecem a seguir:

```
42 #define ROLABAIXO     CTRL('W')
43 #define ROLACIMA      CTRL('Z')

...


109 inicializa( argc, argv )
110 int       argc;
111 char      *argv[];
112 {
...

124             setscrreg( INI_Y, LINES - 1 );
125             idlok( stdscr, TRUE );

...


131 }

...


184 comandos( arquivo )
185 char      *arquivo;     /* nome do arquivo do usuario */
186 {

...


251             case ROLABAIXO:
252                  nscroll( -1 );
253                  movimenta( pos_y, pos_x );
254                  break;
255             case ROLACIMA:
256                  nscroll( 1 );
257                  movimenta( pos_y, pos_x );
258                  break;
...
280 }
```

Você terá observado que, na inicialização, chamamos a rotina **idlok** ("Inserção e Deleção de Linhas OK"). Expliquemos: suponha que você esteja usando o editor edit2, com uma tela cheia de texto. O que você espera ver na tela, ao digitar o comando CONTROLE W (rola para baixo)? Evidentemente, o edit2 terá que redesenhar a tela inteira, com exceção da primeira linha. Ora, se a velocidade da porta de comunicação for 9.600 bauds, a atualização da tela deve demorar quase dois segundos, um tempo apreciável. Você pode, entretanto, estar na situação feliz de possuir um terminal capaz de deletar e inserir linhas por **hardware**. Isto é, certos terminais podem receber comandos especiais para inserir ou deletar linhas internamente no terminal e de forma muito rápida (alguns milissegundos) **sem** a necessidade do programa mandar todo o texto da tela (ou quase) de novo para o terminal. Caso o terminal que você esteja usando tenha esta capacidade, o CURSES otimiza a atualização da tela, valendo-se deste recurso. Esta técnica, por outro lado, pode gerar uma atualização da tela visualmente não muito agradável. A fim de permitir mais controle sobre as otimizações feitas pelo CURSES para atualizar a tela, você pode usar a rotina **idlok**. Com

```
idlok( stdscr, TRUE );
```

o CURSES usa a inserção e deleção de linhas por hardware, se estiverem disponíveis. Com

```
idlok( stdscr, FALSE );
```

os recursos de inserção e deleção por hardware não são usados. Neste caso, a tela inteira seria redesenhada, para fazer o rolamento. Observe que a inserção e deleção por hardware não são usadas pelo CURSES apenas no caso de rolamento de texto.

As rotinas **deleteln** e **insertln**, discutidas anteriormente, também podem levar ao uso destes recursos, sempre que o terminal sendo usado dispuser dos mesmos.

E para concluir o assunto rolamento, uma última observação: vamos supor que você esteja usando o edit2 com uma tela cheia de texto e que você dê o comando CONTROLE Z (rolamento do texto para cima de uma linha). Isto é, sua tela teria a seguinte aparência antes e depois do comando:

```
 INSERÇÃO abc
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

 ...                       ...

ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

TELA ANTES DO COMANDO CONTROLE Z

```
 INSERÇÃO abc
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

 ...                      ...

ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
 (linha em branco)
```

TELA DEPOIS DO COMANDO CONTROLE Z

Vamos supor ainda que seu terminal não dispõe de inserção e deleção de linhas por hardware. Qual é a forma mais rápida de atualizar a tela, passando da primeira ilustração à segunda? Acontece que a grande maioria dos terminais, mesmo aqueles menos sofisticados, tem uma forma de rolar a **tela inteira** por hardware. Basta posicionar o cursor na primeira coluna da última linha e enviar o caractere de alimentação de linha ao terminal. Se fizéssemos isso, a primeira tela mostrada acima apareceria como se segue:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 ...                         ...


ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
 (linha em branco)
```

TELA APÓS ROLAMENTO POR HARDWARE DO TERMINAL

Mas isso é **quase** o que queremos! Bastaria redesenhar a primeira linha (a linha de status), para obter o resultado final. O CURSES usa o rolamento da tela inteira, sempre que julga esta solução mais eficiente (como é o caso aqui). Para pedir que seja considerada esta otimização, você deve chamar a rotina

```
idlok( stdscr, TRUE );
```

logo após a chamada à rotina **initscr**, tal como fizemos aqui (linha 125).

Vamos lá. Próximo comando: remover o caractere à esquerda do cursor. O detalhe particular deste comando é que ele é ativado com uma tecla diferente para usuários diferentes. Por que isso? O SOX permite que cada usuário escolha sua própria tecla de apagamento de caractere. Observe que esta é a tecla usada quando você conversa com o SOX, e nada tem a ver com edit2. Entretanto seria interessante que a mesma tecla fosse utilizada no edit2, para desempenhar esta função, já que o usuário acostumou-se a ela. A função **erasechar** do CURSES retorna o valor desta tecla. Podemos então implementar o comando desejado como se segue:

```
184 comandos( arquivo )
185 char      *arquivo;     /* nome do arquivo do usuario */
186 {

...

262           default:
263                if( cmd == erasechar() ) {
264                     if( movimenta( pos_y, pos_x - 1 ) == OK ) {
265                          delch();
266                     }
267                } else if( isprint( cmd ) ) {  /* da para imprimir ? */
268                     if( modo == ADICAO ) {
269                          addch( cmd );
270                     } else {
271                          insch( cmd );
272                          movimenta( pos_y, pos_x + 1 );
273                     }
274                } else {
275                     beep();
276                }
277                break;
278           }
279      }
280 }
```

Infelizmente, é possível que o valor retornado por **erasechar** entre em conflito com o valor de outro comando do edit2. Neste caso, o comando para apagar o caractere à esquerda do cursor simplesmente não está disponível no editor edit2.

Passemos agora à movimentação do cursor com as teclas de setas. É mais conveniente usar as teclas de setas para movimentar o cursor, do que usar os comandos CONTROLE E, D, S e X. Estes últimos comandos, entretanto, devem permanecer, porque vários terminais simplesmente não possuem teclas de setas. O que queremos, então, é que pressionar tanto CONTROLE D como a tecla -- deva movimentar o cursor para a direita. O problema básico a superar, para implementar este recurso, é que o sistema SOX usa o conjunto de caracteres ASCII, e, neste conjunto, não há valor atribuído para representar as setas. Para resolver este problema, a maioria dos terminais gera uma **seqüência** de caracteres ASCII, quando uma tecla de seta é pressionada. O quadro abaixo indica as seqüências mais comumente geradas pelos terminais.

TECLA	SEQÜENCIA GERADA
↑	ESC A
↓	ESC B
→	ESC C
←	ESC D

A fim de facilitar o tratamento de teclas gerando seqüências, o CURSES transforma as seqüências em valores únicos, muito mais facilmente testados nos nossos programas. Visto que os valores usados para representar estas teclas não devem entrar em conflito com o conjunto ASCII, o CURSES escolhe valores maiores que 255 para as mesmas. No caso particular das quatro teclas de interesse aqui, o CURSES usa os seguintes valores, definidos no arquivo "infoscr.h":

```
#define KEY_DOWN  0402
#define KEY_UP    0403
#define KEY_LEFT  0404
#define KEY_RIGHT 0405
```

Entende-se agora porque a rotina **getch** retorna o tipo "caractere". Este tipo é definido no arquivo "infoscr.h" e permite retornar valores maiores que 255 pela rotina **getch**. O valor retornado por **getch** sempre deve ser armazenado num objeto do tipo "caractere", nunca num "char".

Só falta um detalhe a esclarecer, antes de mostrarmos a implementação final. Lembre que o CURSES deve mapear seqüências de caracteres representando teclas com valores únicos (por exemplo, KEY_RIGHT). Este mapeamento é feito na rotina **getch**, que, normalmente, não faz este mapeamento. Você deve pedir explicitamente que ela o faça, através da seguinte chamada:

```
keypad( stdscr, TRUE );
```

Esta chamada a **keypad** significa: "Sempre que um caractere for lido com **getch** pela janela **stdscr**, cuide do mapeamento de teclas". A chamada

```
keypad( stdscr, FALSE );
```

seria usada para pedir que o mapeamento não fosse feito. O código final é mostrado a seguir.

```
109  inicializa( argc, argv )
110  int        argc;
111  char       *argv[];
112  {
...
126             keypad( stdscr, TRUE );
...
131  }
...

184  comandos( arquivo )
185  char       *arquivo;     /* nome do arquivo do usuario */
186  {
...

211             case ESQUERDA:
212             case KEY_LEFT:
213                  movimenta( pos_y, pos_x - 1 );
214                  break;
215             case DIREITA:
216             case KEY_RIGHT:
217                  movimenta( pos_y, pos_x + 1 );
218                  break;
219             case CIMA:
220             case KEY_UP:
221                  movimenta( pos_y - 1, pos_x );
222                  break;
223             case BAIXO:
224             case KEY_DOWN:
225                  movimenta( pos_y + 1, pos_x );
226                  break;
...

280  }
```

Nosso último comando (redesenhar a tela) tem uma utilidade particular. O SOX é um sistema multiusuário e, como tal, permite que um usuário envie mensagens para o terminal de outro usuário. Agora imagine o que ocorre se, enquanto você estiver usando o editor edit2, algum usuário enviar uma mensagem para seu terminal. Este é o problema que queremos resolver. Queremos uma maneira de pedir ao edit2 que ele redesenhe a tela por inteiro, para remover os caracteres não-desejados. O trecho do programa que implementa o comando é o seguinte:

```
44 #define REDESENHA    CTRL('R')
...

184 comandos( arquivo )
185 char     *arquivo;     /* nome do arquivo do usuario */
186 {
...


259          case REDESENHA:
260               clearok( stdscr, TRUE );
261               break;

...


280 }
```

A rotina **clearok**, tal como chamada acima, não altera o texto presente na janela **stdscr**, e tampouco altera o que está na tela do terminal. Apenas diz ao CURSES: "Olhe, a próxima vez que você atualizar a tela por causa da rotina **refresh**, apague primeiro a tela do terminal e redesenhe-a por inteiro". Terminando de descrever a implementação dos novos comandos do editor edit2, passaremos agora à implementação da acentuação dos caracteres da língua portuguesa.

3.2.2 ACENTUAÇÃO

O conjunto de caracteres usados pelo SOX chama-se ASCII e inclui 128 caracteres diferentes. Este conjunto, entretanto, não inclui os caracteres "acentuados" da língua portuguesa (ã, é, etc.). Para resolver este problema, o CURSES usa, internamente, um conjunto estendido de caracteres, num total de 256, incluindo todos os caracteres da língua portuguesa. Este conjunto chama-se ABICOMP, porque foi padronizado por aquela entidade ("Associação Brasileira da Indústria de Computadores"). Por outro lado, é possível que você não possa usufruir desta característica do CURSES (por exemplo, se o terminal que você está usando não tiver recursos de acentuação). Antes de nos estender sobre os recursos de acentuação do CURSES, falemos um pouco sobre como vários terminais resolvem o problema de acentuação.

Três abordagens devem ser feitas: qual é o **conjunto** de caracteres usado pelo terminal, que **formato** é usado pelo terminal para **entregar** caracteres acentuados digitados no teclado, que **formato** deve ser usado para **enviar** caracteres acentuados para o terminal. Estamos supondo, evidentemente, que o terminal trata caracteres acentuados. Se o terminal não os tratar, a discussão aqui não se aplica.

Existem vários conjuntos de caracteres que incluem caracteres acentuados. Os mais conhecidos são o padrão ABICOMP, o padrão IBM-PC e o padrão ANSI X3.64. Infelizmente, os vários padrões são incompatíveis entre si. De qualquer forma, se seu terminal aceitar o manuseio de caracteres acentuados, ele deve seguir um dos padrões mencionados. Todos estes padrões usam 8 bits para representar um caractere. O conjunto ASCII usa apenas 7 bits.

Observe bem o teclado do seu terminal. Se ele aceitar o manuseio de caracteres acentuados, você deve ver uma tecla para o c-cedilha, e várias teclas de acentuação (~, ', ^, `). Se você digitar, por exemplo, a tecla "~" seguida da tecla "a", estará informando o terminal de que você quer gerar a tecla "ã". Já mencionamos acima que os conjuntos de caracteres com letras acentuadas usam 8 bits para representar cada caractere. A forma mais óbvia de representar o caractere digitado no terminal seria enviando os 8 bits correspondendo ao caractere, dentro do padrão usado pelo terminal. Alguns terminais funcionam justamente assim. Outros geram apenas 7 bits por caractere. Para representar um caractere com o oitavo bit ligado (justamente os caracteres acentuados), estes terminais geram uma **seqüência** de caracteres de 7 bits.

O mesmo acontece quando o computador quer enviar uma letra acentuada para o terminal. Para alguns terminais, deve-se enviar um caractere de 8 bits. Para outros, deve-se enviar uma seqüência de caracteres de 7 bits.

Voltemos ao CURSES. Uma característica positiva do pacote é que ele trata os diferentes terminais de forma transparente para você. De que forma faz isso? Usando o conjunto de caracteres ABICOMP, internamente, e encarregando-se de todo o mapeamento necessário para conversar com terminais que usem outros conjuntos.

Vejamos como o CURSES trabalha na leitura de caracteres acentuados. Por default, ele **não** transforma seqüências representando letras acentuadas em caracteres de 8 bits. Para pedir que faça isso, você deve chamar:

```
modoabc( TRUE );
```

Neste caso, a rotina **getch** retornará caracteres acentuados como valor único (e não como uma seqüência), sempre usando o padrão ABICOMP. Na realidade, para receber caracteres acentuados, você ainda deve chamar a rotina:

```
acent( TRUE );
```

Por quê? Por default, o CURSES **desacentua** as letras acentuadas, tira a cedilha do "cê cedilha" tanto na entrada como na saída. Então, um "ç" seria transformado em "c", "ã" em "a", etc. Se você não quiser que a "desacentuação" seja feita na janela **stdscr**, deve chamar a rotina **acent** como mostrado acima.

Na saída (rotinas **addch, insch**, etc.), as coisas se passam como na entrada: **modoabc** e **acent** devem ser chamadas com argumento TRUE, para que a acentuação funcione corretamente. Porém ainda é possível que o CURSES desacentue caracteres, antes de mandá-los para o terminal: isso ocorre sempre que o terminal em uso não possuir recursos de acentuação.

Para aceitar acentuação no editor edit2, basta fazer o seguinte:

```
109 inicializa( argc, argv )
110 int     argc;
111 char    *argv[];
112 {
...

127         modoabc( TRUE );
128         acent( TRUE );

...

131 }
```

Uma rotina do CURSES que pode ser útil nos seus programas (mas que não usamos no edit2) é

```
desacent( c );
```

onde "c" representa um caractere do conjunto ABICOMP, e a rotina retorna a letra equivalente, desacentuada.

Você deve ter obervado que incluímos o arquivo "isctype.h" no editor edit2, ao invés do arquivo "ctype.h", normalmente usado. A razão é que as macros **isprint,isalpha**, etc., do SOX não funcionariam corretamente para caracteres acentuados do conjunto ABICOMP. Assim, fornecemos uma nova implementação destas macros no arquivo "isctype.h", incluído no item 7.4.

3.2.3 ATRIBUTOS DE VÍDEO

No tocante a este assunto, a situação assemelha-se àquela referente a acentuação: alguns terminais podem representar caracteres na tela com algum destaque (vídeo reverso, piscante, etc.), e outros não têm este recurso. Nos seus programas, entretanto, você pode ligar e desligar atributos de vídeo. O CURSES encarrega-se de representá-los no terminal, quando possível.

Cada caractere presente numa janela do CURSES é representado por 16 bits. Os 8 bits menos significativos são o caractere propriamente dito (usando o conjunto de caracteres ABICOMP), e os 8 bits mais significativos representam os atributos. Os possíveis atributos de vídeo de um caractere são:

Destaque	(A_STANDOUT)
Sublinhado	(A_UNDERLINE)
Vídeo Reverso	(A_REVERSE)
Piscante	(A_BLINK)
Atenuado	(A_DIM)
Intensificado	(A_BOLD)

Os valores entre parênteses estão definidos no arquivo "infoscr.h". Observe que é possível que um determinado terminal tenha condição de representar alguns, mas não todos os atributos. Se você quiser, então, destacar algum texto numa janela, terá que descobrir quais dos atributos estão disponíveis no terminal que está sendo usado. Para remover este obstáculo, o CURSES define um atributo, chamado **A_STANDOUT** (destaque), que representa a "melhor" forma de destacar algo no terminal em questão.

Este atributo muda de um terminal para outro, evidentemente, e é freqüentemente associado ao vídeo reverso.

O CURSES oferece várias rotinas para manipular os atributos dos caracteres de uma janela. Apenas duas delas nos interessam, por enquanto. Após uma chamada à rotina

standout()

todos os caracteres adicionados à janela **stdscr** receberão o atributo **A_STANDOUT**, até que você chame a rotina

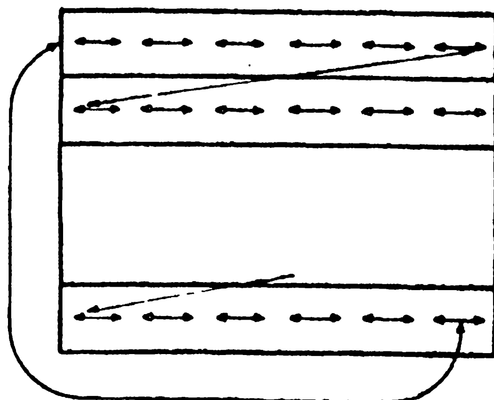
standend()

que significa "fim do destaque". Portanto, no editor edit2, podemos destacar o modo de operação do editor, da seguinte forma:

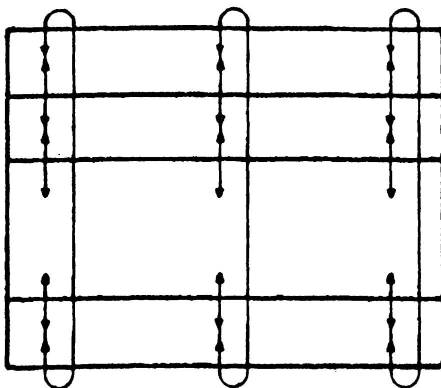
```
308  sta_modo( modo )
309  int      modo;           /* o modo de operacao */
310  {
311
312      int      lemb_y, lemb_x;          /* para voltar o cursor */
313
314      getyx( stdscr, lemb_y, lemb_x );
315      standout();
316      mvprintw( 0, COL_MODO, "%10s", modo == INSERCAO ? "INSERCAO" :
317                                                        "  ADICAO" );
318      standend();
319      move( lemb_y, lemb_x );
320  }
```

Encerrando este capítulo, propomos a você um pequeno exercício.

Você deve ter observado que, nos editores edit1 e edit2, a movimentação do cursor não permite passar, por exemplo, do início de uma linha ao fim da linha anterior, digitando CONTROLE S. Em outras palavras, nossa tela não é **circular.** Seu problema, agora, é tornar a tela do edit2 circular. Nossa definição de "circular" pode ser melhor entendida, se você estudar as duas figuras a seguir.



MOVIMENTAÇÃO HORIZONTAL (← E →)



MOVIMENTAÇÃO VERTICAL (↑ E ↓)

CAPÍTULO 4

RECURSOS MULTIJANELAS

Neste capítulo, nosso editor adquire várias características adicionais. A maioria dos novos recursos é implementada usando janelas adicionais do CURSES, além da janela padrão **stdscr**. Este editor (edit3) tem toda a funcionalidade do editor do capítulo anterior (edit2) e, no próximo capítulo, descrevemos apenas os novos recursos desejados.

4.1 NOVOS RECURSOS DO EDITOR edit3

Podemos dividir a nova funcionalidade do editor em quatro grupos:

- Chamada do editor sem nome de arquivo;
- Inclusão de um novo arquivo no texto, durante a edição;
- Operações com blocos;
- Execução de um shell durante a edição.

Os primeiros três grupos utilizam janelas adicionais do CURSES. O último recurso (execução de um shell) não usa janelas, mas lança mão de recursos interessantes do CURSES, que preferimos discutir neste capítulo. A seguir, uma descrição detalhada de cada item.

4.1.1 CHAMADA DO EDITOR SEM NOME DE ARQUIVO

Os editores edit1 e edit2 são chamados com a mesma sintaxe: o nome do arquivo a editar deve ser fornecido como argumento do programa. O editor edit3, por sua vez, pode ser chamado com ou sem nome de arquivo. Isto não causa problemas durante a edição de texto: o texto simplesmente começa "vazio", caso nenhum nome de arquivo tenha sido especificado. O problema aparece no momento em que queremos gravar o texto num arquivo (ação G). Ao pedir a gravação, e se nenhum nome de arquivo tiver sido dado na chamada do editor, este pergunta o nome do arquivo que deve receber o texto. O usuário fornece então o nome do arquivo desejado.

Estamos, então, pela primeira vez, introduzindo uma interação maior entre o usuário e o editor. Para desempenhar uma determinada ação, o usuário deve fornecer informação adicional ao editor. Em termos de implementação, o ponto importante é que o nome do arquivo digitado pelo usuário **não** faz parte do texto normal sendo editado. A pergunta que você deve estar se fazendo é "onde, na tela, o editor pede a informação e onde será apresentada a resposta do usuário?". Julgamos que a "região de ação" usada nos editores edit1 e edit2 seja pequena demais para conter toda esta informação (a pergunta é grande, e a resposta também pode sê-lo). Neste editor, portanto, decidimos eliminar a região de ação e introduzir uma **região de mensagens**, maior. Examine a figura a seguir.

INSERÇÃO

Nome do arquivo para gravacao:_

Observe que usamos três linhas para a linha de mensagens. Apenas a linha do meio é usada para comunicação com o usuário. Qualquer mensagem para o usuário ou pergunta a ser feita ao mesmo é apresentada como na figura. O caso acima ilustra a pergunta feita ao usuário, quando ele pede a gravação do texto num arquivo ainda desconhecido. Ele digita então o nome do arquivo desejado, encerrando com a tecla <RETURN>. Obviamente, enquanto digita sua resposta, o usuário pode usar a tecla <BACKSPACE> ou a tecla de seta (←) para corrigir erros de digitação.

Antes de passarmos a outros tópicos, duas observações adicionais: primeiro, a região de mensagens é usada para efetuar qualquer comunicação com o usuário, não apenas para a mensagem mostrada na figura anterior. Segundo, as três linhas da região de mensagens não são usadas apenas para mensagens e perguntas. Durante a digitação normal de texto, elas ficam disponíveis para texto, exatamente como nos editores anteriores. Na apresentação de uma mensagem, esta é sobreposta ao texto **apenas na tela do terminal**. Quando a mensagem não é mais necessária, o texto "escondido" deve reaparecer.

4.1.2 INCLUSÃO DE UM NOVO ARQUIVO DE TEXTO DURANTE A EDIÇÃO

Estando posicionado com o cursor em qualquer lugar da região de texto, o comando de inclusão de arquivo (CONTROLE Q) pode ser usado para incluir um arquivo qualquer nesta posição do cursor. O nome do arquivo a ser incluído é pedido na região de mensagens. O editor inclui tantas linhas do arquivo quantas forem necessárias, para chegar à última linha da tela.

4.1.3 EXECUÇÃO DE UM SHELL DURANTE A EDIÇÃO

O shell é criado com uma ação (CONTROLE A). Ao invés de ter duas ações possíveis, como nos editores anteriores (Sai ou Grava), uma terceira ação pode ser pedida pelo usuário, digitando o caractere "!". O editor executa um shell e espera que este termine antes de voltar à edição, no ponto exato original de edição.

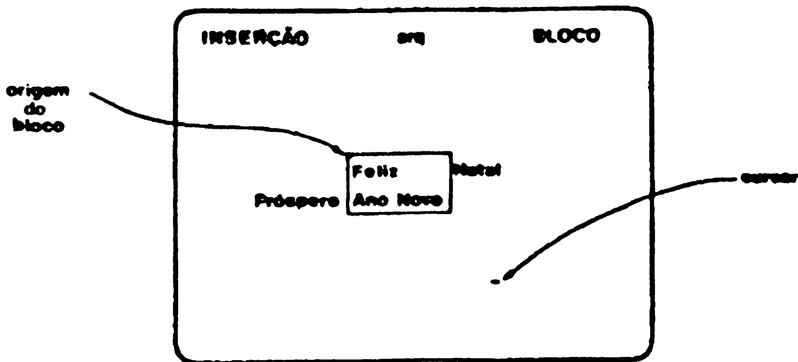
4.1.4 OPERAÇÕES COM BLOCOS

Neste grupo de operações, queremos manusear áreas retangulares de texto que chamamos **blocos**. O editor manipula um único bloco de cada vez. Uma vez definida a localização do bloco (por exemplo, uma área de cinco linhas por dez caracteres com origem na posição (8,21)), as operações que desejamos fazer com ele são:

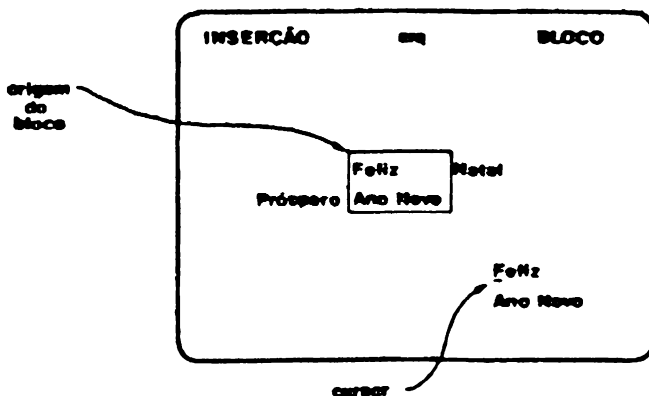
- Copiar o texto incluso no bloco, para uma outra posição qualquer da tela (sempre na região de texto, obviamente) indicada pela posição do cursor.

- Zerar ou branquear o bloco, isto é, colocar um caractere branco (espaço) em cada posição do bloco.
- Gravar o texto presente no bloco num arquivo especificado pelo usuário. O editor deve pedir o nome do arquivo na região de mensagens.

Como exemplo da cópia de um bloco, se começássemos com a tela mostrada na figura abaixo:



e executássemos o comando para copiar o bloco, o resultado seria o seguinte:



Para poder incluir tais operações no editor, precisamos incluir outras operações de "gerência de blocos", a saber:

. Criação de um bloco;
. Eliminação de um bloco (**não** do texto contido no bloco);
. Mudança das dimensões e da localização de um bloco.

A seqüência normal do uso destas operações seria a seguinte:

i. Posicione o cursor no lugar desejado de origem do bloco
ii. Crie um bloco com o comando CONTROLE K (operação 4)
iii. Mude as dimensões do bloco, para cobrir a área desejada (operação 6).
iv. Para zerar o bloco, forneça o comando CONTROLE T (operação 2). Este comando elimina o bloco automaticamente (operação 5).
v. Para gravar o texto do bloco num arquivo, digite o comando de ação (CONTROLE A) e peça a gravação (G). O editor perguntará o nome do arquivo desejado na região de mensagens (operação 3).
vi. Para copiar o bloco, posicione o cursor no lugar desejado (onde a cópia será feita) e acione o comando de cópias de bloco - CONTROLE K (operação 1).
vii. Elimine o bloco com a tecla <ESC> (tecla de escape).

Você deve ter notado que, quando um bloco existe, o sentido de outros comandos muda. Por exemplo, a ação de gravação grava o texto do bloco, se o bloco existe, e grava o texto inteiro, caso contrário. Este segundo caso é idêntico nos editores anteriores. De forma geral, o editor edit3 funciona em um de dois modos: o **modo normal** e o **modo bloco**. Diz-se que o editor está no modo bloco sempre que um bloco existe, e este fato é indicado na linha de status. Ao entrar no modo bloco (comando CONTROLE K), um bloco é criado a partir da posição atual do cursor e com o mesmo tamanho do último bloco criado. A primeira vez que um bloco é criado, ele se estende até o fim da tela. Este bloco é destacado na tela de alguma forma, para você visualizar sua posição. Este detalhe será visto em discussão futura, pois não é tão simples quanto parece à primeira vista.

Uma vez que estamos no modo bloco, as seguintes operações são possíveis:

. Movimentar o cursor

  Os comandos são CONTROLE S, D, E, X, como nos editores anteriores. Chamemos esta operação de "movimentação com a cruz" (lembrar que as letras S, D, E, X formam uma cruz no teclado).

. Redesenhar a tela

  Funciona como no modo normal, usando o comando CONTROLE R.

. Alterar a posição e as dimensões do bloco

  Na realidade, o que alteramos é a posição das **bordas** do bloco.

Estas bordas acompanham o cursor, se este for movimentado com as **teclas de setas**. Isto é, podemos movimentá-lo, sem afetar o bloco (com a cruz), ou afetando o bloco (com as setas). Dois exemplos esclarecerão esta operação. Se o cursor estiver no meio de um bloco, tanto faz movimentá-lo com a cruz ou com as setas: as bordas do bloco permanecem inalteradas. Porém, se, por exemplo, o cursor estiver posicionado na borda esquerda do bloco, tanto CONTROLE S como a tecla ← movimentam o cursor uma posição à esquerda. Porém a borda esquerda do bloco só acompanha o cursor, se este for movimentado com a tecla ← .

. Eliminar o bloco

Este comando é acionado com a tecla <ESC>. Ele elimina o bloco e o editor volta ao modo normal.

. Zerar o conteúdo do bloco

Este comando (CONTROLE T) coloca brancos em todas as posições delimitadas pelo bloco e elimina o bloco, como na operação 4.

. Copiar o bloco para a posição do cursor

Uma vez criado o bloco, o cursor pode ser movimentado para outro lugar da região de texto (operação 1). O conteúdo do bloco pode então ser copiado para a posição do cursor com o comando CONTROLE K. O bloco original permanece inalterado, e o editor fica no modo bloco.

. Disparar uma ação com o comando CONTROLE A

Três ações são possíveis, tal como no modo normal (G = grava, S = Sai, ! = Shell). Entretanto as ações G e S são ligeiramente diferentes no modo bloco. "Gravar" significa gravar o conteúdo do bloco num arquivo. "Sair" significa sair do modo bloco, e não do editor. No modo bloco, portanto, a ação "sair" é equivalente ao comando <ESC> (operação 4). Com as ações G e !, o editor permanece no modo bloco, uma vez terminada a ação.

Observe que nenhum texto pode ser digitado no modo bloco. No modo normal, três novas operações são possíveis:

- Criar um bloco e entrar no modo bloco (CONTROLE K);
- Incluir um arquivo (CONTROLE Q);
- Executar um shell (CONTROLE A !). Quando o shell é encerrado, o editor permanece no modo normal.

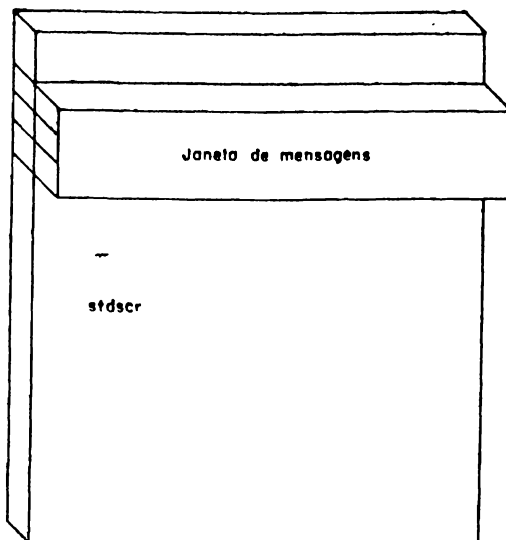
Aí está a descrição do novo editor que queremos construir. Faça como nos capítulos anteriores: digite o programa (item 7.5), compile-o e use-o um pouco, para conhecer as operações do editor.

4.2 IMPLEMENTAÇÃO DO EDITOR - MODO NORMAL

Vamos falar de três assuntos: a gravação de um arquivo quando nenhum nome foi dado, a inclusão de um arquivo durante a edição e a execução de um shell durante a edição. Como estes três recursos usam a região de mensagens, vamos cuidar dela primeiro.

4.2.1 A REGIÃO DE MENSAGENS

Como você se lembra, a região de mensagens consiste em três linhas sobrepostas às primeiras três linhas da região de texto. Se colocássemos as mensagens na janela **stdscr**, perderíamos o texto aí presente. O que precisamos, portanto, é de um meio de colocar informação na tela, sem afetar o que está na janela **stdscr**. No CURSES, isto é feito criando-se uma **nova** janela, com seu próprio tamanho, texto, cursor, localização, etc., e independente de **stdscr**. A figura abaixo ilustra a situação.



Você observará que a informação presente nas linhas 1 a 3 da tela pode corresponder à informação de **stdscr** ou da nova janela. Como decidir quem aparece? Já discutimos a função **refresh**. Na realidade, **refresh** é uma macro que tem a seguinte expansão:

```
wrefresh( stdscr )
```

A rotina **wrefresh** atualiza (na tela) a informação da janela recebida como parâmetro. Portanto, no caso da figura acima, as linhas 1 a 3 da tela conterão a informação associada à **última** janela sujeita à função **wrefresh**.

Uma janela é caracterizada por vários atributos, dentre os quais destacamos:

. Sua origem na tela;

. Sua largura (em colunas);

. Sua altura (em linhas).

Por exemplo, a janela **stdscr** tem origem (0,0), largura 80 e altura 24 (os valores 80 e 24 são, na realidade, COLS e LINES). A janela de mensagens deve ser criada com origem (1,0), largura igual a COLS e altura 3. Isto é feito como se segue:

```
janmsg = newwin( 3, COLS, 1, 0 )
```

A rotina **newwin** cria uma nova janela e retorna um identificador de janela do tipo

```
JANELA *
```

Portanto, a definição da variável **janmsg** seria:

```
JANELA *janmsg;
```

Podemos aplicar a esta nova janela quase todas as funções do CURSES que já encontramos. Por exemplo:

```
wmove( janmsg, 2, 1 );
waddch( janmsg, 'c' );
wrefresh( janmsg );
```

Expliquemos estes três exemplos. O primeiro move o cursor da janela **janmsg** (não da tela!) para a posição (2,1), relativamente à origem da janela (não da tela). A segunda chamada acrescenta o caractere 'c' na janela **janmsg**, na posição atual do cursor desta janela (posição (2,1)). O cursor da janela passaria então para a posição (2,2), relativamente à origem da janela. Após a chamada a **wrefresh**, a informação mudada nesta janela desde a última chamada a wrefresh (janmsg) seria colocada na tela. O caractere 'c' apareceria portanto na posição (3,1) da **tela**, e o cursor da tela estaria sob a posição (3,2).

A janela **janmsg** é criada no editor edit3, como mostrado abaixo.

```
117  JANELA  *janmsg; /*janela usada para imprimir mensagens e ler respostas*/

...
174  inicializa()
175  {

...
189      /* cria e inicializa a janela de mensagens */
190      if( (janmsg = newwin( 3, COLS, INI_Y, INI_X )) == NULL ) {
191          return( ERR );
192      }
193      idlok( janmsg, TRUE );
194      keypad( janmsg, TRUE );
195      wmodoabc( janmsg, TRUE );
196      wacent( janmsg, TRUE );
197      return( OK );
198  }
```

Observe que a função **newwin** retorna NULL, em caso de erro.

O manuseio da região de mensagens é simples. O resto do programa pode escrever algo nesta região e receber de volta uma resposta do usuário, através da função **mens** (que não faz parte do CURSES). Esta função é semelhante a **printf**, mas só aceita um parâmetro, além do formato. Isto é,

```
mens( "Nao pode abrir arquivo %s. Tecle algo. ", nome_arq )
```

funciona, mas

```
mens( "Estou na posi|-o %d, %d. Tecle algo. ", pos_y, pos_x)
```

não funciona (tem dois parâmetros, além do formato).

A função encarrega-se de exibir a mensagem na área de mensagens e de receber a resposta do usuário. Esta resposta é encerrada com RETURN , e **mens** deve então remover a mensagem da tela, reexibindo o texto que foi sobreposto. Sua implementação é mostrada a seguir:

```
495  /*
496   * mens
497   * faz uma pergunta e retorna a resposta (uma cadeia)
498   */
499
500  char *
501  mens( msg, param )
502  char     *msg;
503  char     *param;
504  {
505
506       static char resposta[MAXCOLUNA];
507
508       mvwprintw( janmsg, 1, 1, msg, param );
509       wclrtoeol( janmsg );
510       box( janmsg, '|', '-' );
511       wrefresh( janmsg );
512       echo();
513       wgetstr( janmsg, resposta );
514       noecho();
515       touchwin( stdscr );
516       return( resposta );
517  }
```

A rotina **mvwprintw** (linha 508) é equivalente a **wmove** seguida de **wprintw.** Esta última função é equivalente a **printw,** já vista, mas coloca o texto na janela especificada (**janmsg**). A função **box** (linha 510) desenha uma moldura na janela especificada. As bordas verticais são desenhadas com o caractere '|' e as bordas horizontais com '-'. A janela é então exibida na tela (linha 511).

A resposta é obtida com a rotina **wgetstr** (linha 513). Esta rotina aceita a edição da informação digitada (com as teclas ← ou CONTROLE H) e armazena a cadeia obtida no vetor **resposta**, encerrando quando o usuário digita <RETURN>. Observe que o CURSES deve estar ecoando os caracteres durante a digitação (linhas 512 e 514). O efeito da chamada à rotina do CURSES **touchwin** é sutil (linha 515). Ao retornar da função **mens**, a próxima atualização de tela será feita com a rotina **refresh** (equivalente a **wrefresh(stdscr)**). Se não chamássemos **touchwin**, o texto de **stdscr** nas linhas 1 a 3 **não** seria exibido na tela!

Como? Por quê? Não já falamos que a informação exibida corresponde ao último **wrefresh** dado? E, neste caso, a tela **stdscr** não sobreporia **janmsg**? O pequeno detalhe que não deve ser esquecido é que **wrefresh** atualiza apenas as informações que **mudaram** na **janela** apropriada, desde o último **wrefresh** dado para esta janela. No nosso caso, nenhuma informação de **stdscr** terá mudado (a tela mudou, mas não a informação em **stdscr**). O resultado seria que **wrefresh(stdscr)** decidiria que nada mudou na janela, e nada deve ser feito para exibir esta janela na tela. Em outras palavras, a rotina **wrefresh** não é suficientemente inteligente para descobrir o que é a sobreposição de janelas. O programador deve ajudar. A chamada a **touchwin(stdscr)**, em efeito, diz: "A partir de agora, pode supor que **tudo** mudou na janela **stdscr**". Isto significa que a próxima chamada a **refresh** atualizará corretamente a tela. A rotina **touchwin** é necessária, sempre que um programa usa janelas sobrepostas.

4.2.2 CHAMANDO O EDITOR SEM NOME DE ARQUIVO

Para poder chamar o editor sem nome de arquivo, duas mudanças devem ser feitas no programa. Primeiro, a rotina **main** deve chamar **le_arq** apenas se um nome de arquivo tiver sido fornecido. Examine a nova versão desta rotina (o teste é feito na linha 145).

```
130  main( argc, argv )
131  int argc;
132  char   *argv[];
133  {
134
135      if( argc > 2 ) {
136          fprintf(stderr, "Sintaxe: %s [arquivo]\n", argv[0] );
137          exit( 1 );
138      }
139
140      if( argv[1] != NULL ) {
141          strcpy( arq_usuario, argv[1] );
142      }
143
144      if( inicializa() == ERR          /* nao inicializou */
145       || argv[1] && le_arq(argv[1], INI_Y, INI_X) == ERR /*nao leu arq*/
146       || comandos() == ERR ) {        /* erro nos comandos */
147
148          tchau( 1 ); /* nao retorna */
149      } else {
150          tchau( 0 ); /* nao retorna */
151      }
152  }
```

Você observará também que a chamada a **le_arq** recebe, além do nome do arquivo, dois parâmetros indicando **onde** colocar o texto do arquivo na janela **stdscr.** Isto é necessário porque queremos implementar o comando de inclusão de arquivos (CONTROLE Q), que lê um arquivo em qualquer posição da tela.

Este comando também usará a rotina **le_arq** (mais detalhes podem ser obtidos no item 4.2.3). A nova rotina **le_arq** é mostrada a seguir e é muito semelhante à versão original. Observe a chamada a **mens** nas linhas 224 e 229.

```
201  /*
202   * le_arq
203   * abre e le o arquivo do usuario
204   * colocando o texto na janela stdscr
205   */
206
207  le_arq( arquivo, y, x )
208  char    *arquivo;   /* nome do arquivo do usuario */
209  int      y, x;      /* posicao onde colocar o texto do arquivo */
210  {
211
212      char           *fgets();
213      register int   lin;    /* para varrer as linhas */
214      FILE           *fp;    /* arquivo aberto para leitura */
215      int            tam;    /* tamanho da leitura */
216      char           buf[MAXCOLUNA + 2]; /* buffer de leitura */
217      int            ret;
218
219      ret = OK;
220      tam = min( MAXCOLUNA, COLS + 2 );   /* +2 por causa do \n
221                                           * e do \0 finais
222                                           */
223      if( (fp = fopen( arquivo, "r")) == (FILE *)NULL ) {
224          mens( "Nao pode abrir %s. Tecle <RETURN>. ", arquivo );
225      } else {
226          for( lin = y; lin < LINES; lin++ ) {
227              if( fgets( buf, tam, fp ) == (char *)NULL ) {
228                  if( ! feof( fp ) ) {    /* erro de leitura */
229                      mens( "Erro de leitura em %s. Tecle <RETURN>. ",
```

```
230                             arquivo );
231                         ret = ERR;
232                     }
233                     break;
234                 }
235                 mvaddstr( lin, x, buf ); /* coloca o texto na janela */
236             }
237             fclose( fp );
238         }
239         move( y, x );
240         return( ret );
241     }
```

A segunda mudança a ser feita no programa, para aceitar a chamada do editor sem nome de arquivo, diz respeito à gravação. Se um nome de arquivo não tiver sido especificado (o vetor **arq_usuario** contém uma cadeia nula - veja as linhas 118, 140 a 142), a gravação só pode prosseguir quando o usuário tiver dado um nome de arquivo, a pedido do editor. Examine o trecho de código abaixo.

```
422  acao( ap_res )
423  int *ap_res;    /* onde colocar o resultado: OK ou ERR */
424  {

...
443      case ACAO_GRAVA:
444          if( bloco.existe ) {    /* gravacao do bloco */
...

455          } else {    /* gravacao de stdscr inteira */
456              tam_vert = LINES;
457              tam_hor = COLS;
458              ini_y = INI_Y;
459              ini_x = INI_X;
460              jan = stdscr;
```

```
461            if( *arq_usuario == '\0' ) {
462                strcpy( arq_usuario,
463                    mens( "Nome do arquivo para gravacao: " ) );
464                sta_arq( arq_usuario );
465            }
466            arq = arq_usuario;
467        }
468        if( (*ap_res = grava( arq,jan,tam_vert,tam_hor,ini_y,ini_x ))
469                    == ERR ) {
470            mens( "Erro de gravacao. Tecle <RETURN>. ");
471        }

...
475        break;

...
491      return( ret );
492  }
```

Lembre que a ação de gravação pode ser usada para gravar um bloco ou o texto inteiro. Apenas o segundo caso nos interessa por enquanto. A linha 444 testa se o editor está no modo bloco e será explicada no item 4.3. O editor obtém o nome do arquivo através da região de mensagens, na linha 463, caso o usuário não tenha especificado um (o teste está na linha 461). A rotina

grava(arq, jan, tam_vert, tam_hor, ini_y, ini_x)

grava no arquivo **arq** a informação da janela **jan** contida numa região retangular com origem **(ini_y, ini_x)**, contendo **tam_vert** linhas de **tam_hor** colunas. Esta rotina é mais geral do que nos editores anteriores, porque será usada para gravar um bloco num arquivo (item 4.3). Estude a função **grava**, a seguir (ela mudou pouco com relação às versões anteriores).

A rotina **mvwinch** é semelhante a **mvinch**, já empregada, mas funciona com uma janela qualquer.

```
520  /*
521   * grava
522   * grava o texto de uma janela no arquivo do usuario
523   */
524
525  grava( arquivo, janela, tam_vert, tam_hor, ini_y, ini_x )
526  char    *arquivo;   /* nome do arquivo do usuario */
527  JANELA  *janela;    /* janela a gravar */
528  int     tam_vert, tam_hor;  /* tamanho da janela a gravar */
529  int     ini_y, ini_x;   /* posicao inicial da janela a gravar */
530  {
531
532      int             ret;     /* valor de retorno da funcao grava() */
533      register int    lin;     /* para varrer as linhas da janela */
534      register int    col;     /* para varrer as colunas da janela */
535      int             col_final;  /* coluna final a gravar na linha */
536      FILE            *fp;     /* arquivo aberto para gravacao */
537      int             lemb_y, lemb_x; /* para voltar o cursor */
538
539      if( (fp = fopen( arquivo, "w")) == (FILE *)NULL ) {
540          mens( "Nao pode criar arquivo %s. Tecle algo. ", arquivo );
541          return( ERR );
542      }
543
544      getyx( janela, lemb_y, lemb_x );
545      ret = OK;
546      for( lin = ini_y; ret == OK && lin < tam_vert; lin++ ) {
547          /* procura a ultima coluna que interessa nesta linha */
548          for( col_final = tam_hor - 1;
549               col_final >= 0 && mvwinch( janela, lin, col_final ) == ' ';
550               col_final-- )
551              ;
552
553          for( col = 0; col <= col_final; col++ ) {
554              if( fputc( mvwinch( janela, lin, col ), fp ) == EOF ) {
555                  ret = ERR;
```

```
556                 break;
557             }
558         }
559
560         if( fputc( '\n', fp ) == EOF ) {
561             ret = ERR;
562         }
563     }
564
565     if( fclose( fp ) == EOF ) {
566         ret = ERR;
567     }
568     wmove( janela, lemb_y, lemb_x );
569     return( ret );
570 }
```

4.2.3 INCLUINDO UM NOVO ARQUIVO NO TEXTO

Com a rotina **le_arq** pronta (item 4.2.2), a implementação do comando CONTROLE Q fica trivial. Vejamos:

```
48  #define LEARQ       CTRL('Q')
...
250 comandos()
251 {
...
264     while( TRUE ) {
265         getyx( stdscr, pos_y, pos_x );
266         refresh();
267         switch( cmd = getch() ) {
...
331         case LEARQ:
332             arq = mens( "Nome do arquivo a ler: " );
333             le_arq( arq, pos_y, pos_x );
334             break;


...
351         }
352     }
353 }
```

4.2.4 EXECUTANDO UM SHELL DURANTE A EDIÇÃO

Este recurso é muito simples de implementar e, aparentemente, nada tem a ver com janelas. Decidimos incluí-lo no editor, para ilustrar como "fugir" temporariamente do controle da tela feito pelo CURSES e como reintegrar ao CURSES.

A forma padrão de executar, no SOX, um shell a partir de outro programa é ilustrada a seguir e não será explicada.

```
480     istat = signal( SIGINT, SIG_DFL );
481     system( "exec sh" );
482     signal( SIGINT, istat );
```

O que nos interessa é o que fazer **antes** e **depois** deste trecho, para integrar o código ao resto do editor.

Uma das tarefas é a de colocar o terminal no modo correto, já que o CURSES inicializa o terminal com características não-padrão (eco desligado, etc.). A rotina **reset_shell_mode** é usada para deixar o terminal no estado em que estava antes da chamada a **initscr**. Para colocá-lo no estado em que o colocamos através do CURSES (isto é, o inverso de **reset_shell_mode**), usamos a rotina **reset_prog_mode**. O trecho de código para executar o shell cresceu, pois, para o seguinte:

```
479     reset_shell_mode();
480     istat = signal( SIGINT, SIG_DFL );
481     system( "exec sh" );
482     signal( SIGINT, istat );
483     reset_prog_mode();
```

Continuando, você há de convir que, após a execução do shell, o CURSES não terá a mínima idéia do que deixamos na tela, com a execução de comandos pelo shell. Isto prejudica, porque o CURSES **tem** que saber o que está na tela, para que seus algoritmos de otimização funcionem. A solução é dizer a eles: "Esqueça do que você acha que está na tela. Desenhe tudo de novo". Isto é feito com a chamada

```
clearok( stdscr, TRUE )
```

Observe que a rotina **clearok** não redesenha a tela. Ela apenas diz que, quando o **próximo wrefresh** for feito com a janela **stdscr**, a tela deve ser apagada e redesenhada por inteiro. A chamada a **clearok** seria feita, então, após o término da execução do shell.

Finalmente, **antes** de executar o shell, seria interessante apagar a tela do terminal. No CURSES, temos uma rotina para apagar **janelas**, mas não temos uma rotina para apagar a **tela**. Visto que não queremos mexer com o texto de nossas janelas, teremos que apagar a tela "por fora" (isto é, sem usar o CURSES). Usaremos a biblioteca TERMINFO para fazer isso. A solução não será discutida em detalhes aqui, já que a TERMINFO será objeto de discussão no Capítulo 6.

A seguir, a solução completa para a execução de um shell. Observe que, para usar a biblioteca TERMINFO, devemos incluir o arquivo **infoterm.h** (linha 7). Já que este arquivo inclui **curses.h**, não temos que fazê-lo novamente.

```
  7  #include   <infoterm.h>

...
 53  #define ACAO_SHELL  '!'

...
422  acao( ap_res )
423  int *ap_res;    /* onde colocar o resultado: OK ou ERR */
424  {
425
426      char    *resp;  /* resposta do usuario */
427      int     ret;    /* valor de retorno da funco acao() */
428      int     tam_vert;   /* tamanho vertical da janela a gravar */
429      int     tam_hor;    /* tamanho horizontal da janela a gravar */
430      char    *arq;       /* nome do arquivo a gravar */
431      JANELA  *jan;       /* janela a gravar */
432      int     ini_y, ini_x;   /* posicao inicial da janela a gravar */
433      int     (*istat)(); /* acao para SIGINT deve ser guardada */
434                          /* antes de disparar um shell */
435
436      *ap_res = OK;   /* supoe que nao havera erro */
437      ret = CONTINUA;
438      resp = mens( "Sai(s), Grava(g) ou Shell(!) ? ");
439      switch( PADRONIZA( *resp ) ) {

...

476      case ACAO_SHELL:
477          /* dispara um shell */
478          apag_tela();
479          reset_shell_mode();
480          istat = signal( SIGINT, SIG_DFL );
481          system( "exec sh" );
482          signal( SIGINT, istat );
483          reset_prog_mode();
484          clearok( stdscr, TRUE );    /* forca redesenho da tela */
485          break;
486      default:
```

```
487            beep();      /* usuario se arrependeu ou errou */
488            break;
489        }
490
491        return( ret );
492    }

...

949    /*
950     * apag_tela
951     * Apaga a tela do terminal sem usar curses
952     */
953
954    apag_tela()
955    {
956
957        int _sai_carac(); /* rotina interna do curses */
958
959        tputs( clear_screen, 1, _sai_carac );
960        fflush( stdout );
961    }
```

4.3 IMPLEMENTAÇÃO DO EDITOR - MODO BLOCO

4.3.1 DESTAQUE DO BLOCO - AS ALTERNATIVAS

Antes de partirmos para a discussão dos vários comandos disponíveis no modo bloco do editor, precisamos cuidar de alguns detalhes operacionais. O primeiro deles é como destacar o bloco na tela, de forma a que o usuário o visualise facilmente. Nossa primeira decisão é destacar apenas as **bordas** (ou margens) do bloco. A forma mais simples e visualmente atraente de fazer isto é usando atributos de vídeo. Infelizmente, nem todo terminal possui todos os atributos de vídeo manipulados pelo CURSES. É justamente por esta razão que o atributo **A_STANDOUT (standout** significa "destaque") foi inventado. Ele representa o "melhor" atributo que o terminal possui para destacar algo na tela. Este atributo não é o mesmo para todos os terminais. Na maioria, representa vídeo reverso. Se tivéssemos, então, um bloco definido entre as linhas Y_INF a Y_SUP e as colunas X_INF a X_SUP, o seguinte trecho de código desenharia as quatro bordas do bloco:

```
787         for( x = X_INF; x <= X_SUP; x++ ) {
788             c = mvinch( Y_INF, x );
789             addch( c | A_STANDOUT );     /* linha inicial */
790             c = mvinch( Y_SUP, x );
791             addch( c | A_STANDOUT );     /* linha final */
792         }
793         for( y = Y_INF; y <= Y_SUP; y++ ) {
794             c = mvinch( y, X_INF );
795             addch( c | A_STANDOUT );     /* coluna inicial */
796             c = mvinch( y, X_SUP );
797             addch( c | A_STANDOUT );     /* coluna final */
798         }
```

Observe que, para ligar o atributo **A_STANDOUT** de um único caractere, temos que mover o cursor de **stdscr** para o lugar correto, obter o caractere aí presente (linhas 788, 790, 794, 796) e colocar o caractere de volta, mas com atributo ligado (linhas 789, 791, 795, 797). O efeito seria o mesmo se fizéssemos:

```
standout();
for( x = X_INF; x  = X_SUP; x++ )
    c = mvinch( Y_INF, x );
    addch( c );
    c = mvinch( Y_SUP, x );
    addch( c );

for( y = Y_INF; y  = Y_SUP; y++ )
    c = mvinch( y, X_INF );
    addch( c );
    c = mvinch( y, X_SUP );
    addch( c );

standend();
```

No editor, usamos o primeiro método, para mostrar que os atributos de um caractere podem ser ligados diretamente com a rotina **addch**. Neste caso, entretanto, o atributo dado ao caractere não afeta os atributos dos próximos caracteres colocados na janela.

Nossa solução, infelizmente, não funciona para qualquer terminal. Primeiro, há alguns terminais que não possuem atributos de vídeo. Segundo, há alguns terminais que, apesar de possuírem atributos de vídeo, têm comportamento "estranho" na presença de atributos de vídeo. Este tipo de terminal tem vários apelidos:

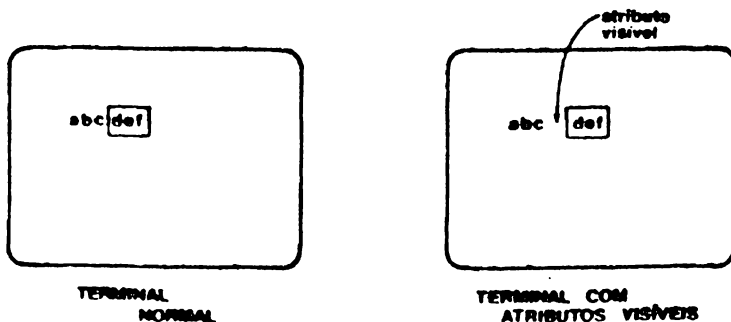
. Terminal com atributos visíveis;

. Terminal com atributos que ocupam espaço;

. Terminal com **magic cookie.**

Qualquer que seja seu nome (usaremos o primeiro), este tipo de terminal nos atrapalha. Por quê? Terminais modernos usam **dois** octetos para representar cada posição na tela, um octeto para o caractere, e um octeto para os atributos deste caractere. Terminais mais antigos, entretanto, usam um único octeto para cada posição da tela. Quando um atributo é ligado, uma posição da tela é "queimada", para representar o atributo. Na tela, esta posição (o **magic cookie**) fica em branco, mas internamente, no terminal, e contém informações dizendo que todos os caracteres **à direita** possuem certos atributos. Daí, os três nomes dados a este tipo de terminal que mencionamos acima.

Vamos mostrar com um exemplo o que ocorre neste tipo de terminal. Enviamos os caracteres "abc" para o terminal, depois ligamos um atributo qualquer, e então enviamos os caracteres "def". Veja o resultado na figura abaixo, para dois tipos de terminais.



Observe que, no primeiro caso, seis posições da tela foram ocupadas. No segundo exemplo, sete posições foram usadas.

A fim de manter o texto da tela nas posições desejadas pelo programador, o CURSES trata atributos de vídeo de forma um pouco especial, para terminais com atributos visíveis. Os atributos são ligados (e desligados) apenas se houver um espaço em branco **já presente**, que será aproveitado para colocar o **magic cookie**. Desta forma, o usuário não sente que o terminal tem comportamento estranho com respeito a atributos, mas cabe ao programador organizar suas telas deixando um espaço em branco no texto, sempre que houver mudança de atributos. Você pode observar que fizemos isso nos editores edit2 e edit3. O modo do editor (INSERÇÃO ou ADIÇÃO) é exibido em destaque na coluna 1 da linha de status, deixando um branco na coluna 0.

Muito bem. Vamos voltar ao assunto inicial: como destacar o bloco no editor edit3. Atributos de vídeo não podem ser usados no caso de terminais com atributos visíveis, já que não podemos assegurar que haverá um espaço em branco, em cada linha, de cada lado da borda esquerda e da borda direita. Neste caso, decidimos colocar um '*' nos quatro cantos do bloco. Isto significa que devemos guardar os quatro caracteres cujas posições serão tomadas pelos asteriscos, e devolvê-los à janela, quando o bloco é removido.

Nosso programa, portanto, deve decidir a forma usada para destacar o bloco. Atributos de vídeo podem ser usados, desde que o terminal tenha o atributo **A_STANDOUT** e não tenha atributos visíveis.

Podemos testar estas duas condições usando variáveis do TERMINFO. No editor, o teste é feito com a seguinte macro:

```
108  /*
109   * O desenho das bordas do bloco pode usar atributos
110   * de video se o terminal for "razoavel"
111   */
112  #define PODE_USAR_STANDOUT  (magic_cookie_glitch < 1 && \
113                               enter_standout_mode != NULL)
```

Se a macro **PODE_USAR_STANDOUT** tiver valor verdadeiro, as bordas são desenhadas com o atributo **A_STANDOUT**, caso contrário, os quatro cantos do bloco são marcados com o caractere '*'.

4.3.2 A ESTRUTURA DE CONTROLE DO BLOCO

O editor precisa manter algumas informações sobre o bloco, a saber:

- Se ele existe ou não;
- Sua localização;
- Os quatro caracteres que foram sobrepostos pelos asteriscos e que deverão ser recolocados na janela, quando o bloco desaparecer.

A localização do bloco (segundo item) deve ser guardada, mesmo quando o bloco não existe, porque a criação de um bloco deve manter o tamanho do último bloco criado. No início da execução do programa, como nunca houve bloco antes, usamos uma localização impossível (bloco na linha de status). Ao criar um bloco, e encontrando uma localização absurda, o programa cria o primeiro bloco, estendendo-se até o fim da tela.
A estrutura que controla o bloco é mostrada a seguir. Observe a localização inicial do bloco.

```
68  /*
69   * estrutura que controla o bloco
70   */
71
72  struct bloco {
73      bool    existe;      /* o bloco existe ? */
74      short   y_inf;       /* linha inicial do bloco */
75      short   x_inf;       /* coluna inicial do bloco */
76      short   y_sup;       /* linha final do bloco */
77      short   x_sup;       /* coluna final do bloco */
78      caractere   c_inf_inf;  /* o caracteres dos quatro cantos
79                               * do bloco (se precisarmos desenhar
80                               * a borda sem atributos de video)
81                               * ...
82                               */
83      caractere   c_inf_sup;
84      caractere   c_sup_inf;
85      caractere   c_sup_sup;  /* ... ate aqui */
86  } bloco = {
87      FALSE,       /* no inicio do programa, o bloco nao existe */
88      0, 0, 0, 0, /* devem ser zero para indicar que nunca
89                   * existiu bloco
90                   */
91      ' ', ' ', ' ', ' '  /* estes valores nao importam */
92  };
93
94
95  /*
96   * macros usadas pelas rotinas que nao devem
97   * conhecer a estrutura do bloco mas que
98   * precisam informacao sobre a localizacao
99   * do bloco
100  */
101
102 #define Y_INF   (bloco.y_inf)
103 #define X_INF   (bloco.x_inf)
104 #define Y_SUP   (bloco.y_sup)
105 #define X_SUP   (bloco.x_sup)
```

4.3.3 DEFINIÇÃO E DELEÇÃO DE UM BLOCO

Já estamos prontos para escrever algumas rotinas de apoio, para definir um bloco na janela **stdscr** e removê-lo. Não sabemos ainda quem vai chamar tais rotinas. Fecharemos este assunto no item 4.3.4.

A primeira sub-rotina que queremos escrever é chamada da seguinte forma:

```
def_bloco( y_inf, x_inf, y_sup, x_sup )
```

onde o bloco sendo definido estende-se da linha **y_inf** à linha **y_sup**, e da coluna **x_inf** à coluna **x_sup**. Esta rotina deve desempenhar as seguintes funções:

- Atualizar as informações de controle na estrutura **bloco**;
- Atualizar as bordas do bloco na janela **stdscr** (removendo as bordas anteriores, se for o caso);
- Indicar que o editor está no modo bloco, na linha de status.

Veja como a rotina foi implementada:

```
717  /*
718   * def_bloco
719   * define um bloco delineado pelos parametros recebidos
720   *
721   * preenche a estrutura "bloco" com a informacao sobre
722   * a localizacao do bloco, desenha a borda para identificar
723   * o bloco e marca o bloco como existente
724   */
725
726  def_bloco( y_inf, x_inf, y_sup, x_sup )
727  register int    y_inf, x_inf, y_sup, x_sup; /* localizacao */
728  {
729
730      if( bloco.existe ) {
731          del_bloco();     /* deleta o bloco existente para
732                            * definir outro
733                            */
734      }
735
736      Y_INF = y_inf;
737      X_INF = x_inf;
738      Y_SUP = y_sup;
739      X_SUP = x_sup:
740
741      desenha_borda();
742
743      bloco.existe = TRUE;
744      sta_bloco( TRUE );
745  }
```

As bordas anteriores são removidas na linha 731, redefinidas nas linhas 736 a 739, redesenhadas na linha 743. Observe que poderíamos criar uma forma mais rápida de atualizar as bordas, já que, normalmente, três das bordas anteriores permanecem no novo bloco. Optamos, entretanto, pela simplicidade do código.

A linha de status é atualizada pela rotina **sta_bloco** (linha 744), que mostramos abaixo.

```
 58  #define COL_BLOCO   (COLS - 30) /* regiao "bloco" */

...

747  /*
748   * sta_bloco
749   * exibe se estamos no modo bloco ou nao na linha de status
750   */
751
752  sta_bloco( modo )
753  int modo;   /* TRUE: modo bloco, FALSE: modo normal */
754  {
755
756      int lemb_y, lemb_x;      /* para voltar o cursor */
757
758      getyx( stdscr, lemb_y, lemb_x );
759      if( modo ) {
760          standout();
761      }
762      mvprintw( 0, COL_BLOCO, "%10s", modo ? "BLOCO" : "" );
763      standend();
764      move( lemb_y, lemb_x );
765  }
```

Nada de novo na rotina **sta_bloco**. Passemos ao desenho das bordas. Como já mencionamos no item 4.3.1, devemos desenhá-la com atributos de vídeo ou com asteriscos nos quatro cantos do bloco, dependendo do tipo de terminal que estamos manuseando. Veja o resultado:

```
767  /*
768   * desenha_borda
769   *
770   * representacao visual do bloco na janela
771   * Se for possivel usar atributos de video, usa-os
772   * senao, coloca um asterisco nos quatro cantos
773   * do bloco, salvando os caracteres ai presentes para
774   * restauracao posterior
775   */
776
777  desenha_borda()
778  {
779
780      int             lemb_y, lemb_x; /* para voltar o cursor */
781      register int    y, x;        /* para varrer as bordas */
782      caractere       c;       /* para pegar o que esta na janela */
783
784      getyx( stdscr, lemb_y, lemb_x );
785
786      if( PODE_USAR_STANDOUT ) {
787          for( x = X_INF; x <= X_SUP; x++ ) {
788              c = mvinch( Y_INF, x );
789              addch( c | A_STANDOUT );    /* linha inicial */
790              c = mvinch( Y_SUP, x );
791              addch( c | A_STANDOUT );    /* linha final */
792          }
793          for( y = Y_INF; y <= Y_SUP; y++ ) {
794              c = mvinch( y, X_INF );
795              addch( c | A_STANDOUT );    /* coluna inicial */
796              c = mvinch( y, X_SUP );
797              addch( c | A_STANDOUT );    /* coluna final */
798          }
799      } else {    /* tem que colocar '*' nos quatro cantos */
800          bloco.c_inf_inf = mvinch( Y_INF, X_INF );
801          addch( '*' );
802          bloco.c_inf_sup = mvinch( Y_INF, X_SUP );
803          addch( '*' );
804          bloco.c_sup_inf = mvinch( Y_SUP, X_INF );
```

```
005         addch( '*' );
006         bloco.c_sup_sup = mvinch( Y_SUP, X_SUP );
007         addch( '*' );
008     }
009     move( lemb_y, lemb_x );
010 }
```

Passemos à deleção do bloco, feita com a rotina **del_bloco**. Esta rotina deve:

- Eliminar as bordas;
- Atualizar as informações da estrutura de controle **bloco**;
- Atualizar a linha de status.

O resultado é o que se segue.

```
013  /*
014   * del_bloco
015   * deleta o bloco e remove as bordas
016   */
017
018  del_bloco()
019  {
020
021      elim_borda();
022      bloco.existe = FALSE;
023      sta_bloco( FALSE );
024  }
025
026
027  /*
```

```
828   * elim_borda
829   *
830   * elimina as bordas, removendo os atributos de video
831   * ou eliminando os '*' dos quatro cantos
832   */
833
834  elim_borda()
835  {
836
837      int             lemb_y, lemb_x; /* para voltar o cursor */
838      register int    y, x;       /* para varrer as bordas */
839      caractere       c;      /* para pegar o que esta na janela */
840
841      getyx( stdscr, lemb_y, lemb_x );
842
843      if( PODE_USAR_STANDOUT ) {
844          for( x = X_INF; x <= X_SUP; x++ ) {
845              c = mvinch( Y_INF, x );
846              addch( c & A_TEXTO );   /* linha inicial */
847              c = mvinch( Y_SUP, x );
848              addch( c & A_TEXTO );   /* linha final */
849          }
850          for( y = Y_INF; y <= Y_SUP; y++ ) {
851              c = mvinch( y, X_INF );
852              addch( c & A_TEXTO );   /* coluna inicial */
853              c = mvinch( y, X_SUP );
854              addch( c & A_TEXTO );   /* coluna final */
855          }
856      } else {    /* tem que remover os '*' nos quatro cantos */
857          mvaddch( Y_INF, X_INF, bloco.c_inf_inf );
858          mvaddch( Y_INF, X_SUP, bloco.c_inf_sup );
859          mvaddch( Y_SUP, X_INF, bloco.c_sup_inf );
860          mvaddch( Y_SUP, X_SUP, bloco.c_sup_sup );
861      }
862      move( lemb_y, lemb_x );
863  }
```

Observe como os quatro caracteres guardados na rotina **desenha_borda** são devolvidos à janela nas linhas 857 a 860. Se estivermos usando atributos de vídeo, a rotina **mvinch** retornará o caractere juntamente com seu atributo (nos 8 bits mais significativos). A devolução do caractere com **addch** deve remover o atributo, e isto é feito com a máscara **A_TEXTO**, definida no arquivo **curses.h**.

4.3.4 OPERAÇÕES COM BLOCOS

Lembre que, no modo bloco, devemos implementar as seguintes operações:

- Entrar no modo bloco, criando um bloco (na realidade, esta operação é feita no modo normal do editor);
- Eliminar o bloco, saindo do modo bloco;
- Mudar as dimensões do bloco;
- Copiar o bloco para a posição do cursor;
- Zerar ou branquear o bloco;
- Gravar o bloco num arquivo;
- Movimentar o cursor;
- Redesenhar a tela;
- Executar um shell.

Discutiremos a implementação destas operações neste capítulo, encerrando a discussão do editor edit3. Para entrar no modo bloco, basta reconhecer mais um "case" na rotina **comandos**:

```
 45  #define COPIABLOCO  CTRL('K')

...
 250  comandos()
 251  {

...
 264      while( TRUE ) {
 265          getyx( stdscr, pos_y, pos_x );
 266          refresh();
 267          switch( cmd = getch() ) {

...
 328          case COPIABLOCO:
 329              trata_bloco();
 330              break;

...
 351          }
 352      }
 353  }
```

A rotina **trata_bloco**, por sua vez, deve definir um bloco com origem na posição do cursor, e cujo tamanho é igual ao tamanho do bloco anterior, se possível. A rotina então obtém um comando do usuário e o interpreta de forma semelhante à rotina **comandos**.

```
591  /*
592   * trata_bloco
593   * enquanto um bloco esta definido, o laco
594   * de comandos eh este, e nao o presente na rotina
595   * "comandos"
596   */
597
598  trata_bloco()
599  {
600
601      int         pos_y, pos_x;   /* posicao do cursor antes
602                                   * da digitacao de um comando
603                                   */
604      int         fim_y, fim_x;   /* localizacao o fim do bloco */
605      int         resultado;      /* resultado da acao */
606      bool        em_bloco;       /* usado para sair do modo bloco */
607      caractere   cmd;            /* comando digitado pelo usuario */
608
609      getyx( stdscr, pos_y, pos_x );
610
611      /* definicao do bloco */
612      if( Y_INF == 0 ) {
613          /* nunca houve bloco antes */
614          fim_y = LINES - 1;
615          fim_x = COLS - 1;
616      } else {
617          /* tenta manter o tamanho do bloco anterior */
618          fim_y = min( pos_y + Y_SUP - Y_INF, LINES-1 );
619          fim_x = min( pos_x + X_SUP - X_INF, COLS-1 );
620      }
621      def_bloco( pos_y, pos_x, fim_y, fim_x );
622      em_bloco = TRUE;
```

```
623      while( em_bloco ) {
624          getyx( stdscr, pos_y, pos_x );
625          refresh();
626          switch( cmd = getch() ) {
627          case ESQUERDA:
628              movimenta( pos_y, pos_x - 1 );
629              break;
630          case DIREITA:
631              movimenta( pos_y, pos_x + 1 );
632              break;
633          case CIMA:
634              movimenta( pos_y - 1, pos_x );
635              break;
636          case BAIXO:
637              movimenta( pos_y + 1, pos_x );
638              break;

...
706          case REDESENHA:
707              clearok( stdscr, TRUE );
708              break;
709          default:
710              beep();
711              break;
712          }
713      }
714  }
```

No trecho acima, já incluímos algumas operações (movimentação do cursor, redesenho da tela, erros do usuário), que são idênticas às operações equivalentes já vistas nos editores anteriores.

Vamos discutir agora o redimensionamento do bloco com as teclas de setas. Discutiremos apenas o caso da tecla → (**KEY_RIGHT**). O código para as outras três teclas é semelhante e pode ser examinado no item 7.5. A tecla **KEY_RIGHT** pode ser usada para mover a borda esquerda (se o cursor estiver posicionado nela), ou para mover a borda direita. O único caso especial importante a considerar ocorre quando o bloco tem largura de uma única coluna (as bordas esquerda e direita estão na mesma coluna). Evidentemente, como a borda esquerda não pode estar à direita da borda direita, devemos mover a borda **direita**, neste caso. É o que o código faz. Veja:

```
652         case KEY_RIGHT:
653             if( movimenta( pos_y, pos_x + 1 ) == OK ) {
654                 /* X_SUP tem prioridade sobre X_INF */
655                 if( pos_x == X_SUP ) {
656                     def_bloco( Y_INF, X_INF, Y_SUP, pos_x + 1 );
657                     pos_x++;
658                 }
659                 if( pos_x == X_INF ) {
660                     def_bloco( Y_INF, pos_x + 1, Y_SUP, X_SUP );
661                 }
662             }
663             break;
```

A linha 657 (**pos_x++**) evita que as duas bordas sejam movidas ao mesmo tempo.

Os comandos CONTROLE T (para zerar ou branquear o bloco) e <ESC> (elimina bloco) são implementados como se segue e dispensam discussão:

```
 46  #define ZERABLOCO   CTRL('T')
 47  #define DELBLOCO    '\033'    /* ESC */
...
598  trata_bloco()
599  {
...
622      em_bloco = TRUE;
623      while( em_bloco ) {
624          getyx( stdscr, pos_y, pos_x );
625          refresh();
626          switch( cmd = getch() ) {
...
688          case ZERABLOCO:
689              branq_bloco();
690              del_bloco();
691              em_bloco = FALSE;
692              break;
693          case DELBLOCO:
694              del_bloco();
695              em_bloco = FALSE;
696              break;
...
712          }
713      }
714  }
...
866  /*
867   * branq_bloco
868   *
869   * preenche a area do bloco com brancos
870   */
871  branq_bloco()
872  {
873
874      int             lemb_y, lemb_x;
875      register int    y, x;
876
877      getyx( stdscr, lemb_y, lemb_x );
878      for( y = Y_INF; y <= Y_SUP; y++ ) {
879          for( x = X_INF; x <= X_SUP; x++ ) {
880              mvaddch( y, x, ' ' );
881          }
882      }
883      move( lemb_y, lemb_x );
884  }
```

Passemos às ações (comando CONTROLE A). No modo bloco, são as seguintes:

. S - Sai do modo bloco, removendo o bloco;

. G - Grava o bloco num arquivo;

. ! - Executa um shell.

A chamada à rotina **acao** é feita na rotina **trata_bloco&.**

```
598  trata_bloco()
599  {

...
697          case ACAO:
698              if( acao( &resultado ) == FIM ) {
699                  del_bloco();
700                  em_bloco = FALSE;
701              }
702              break;

...
714  }
```

Observe que a rotina **acao** deve retornar **FIM** apenas para a ação "Sai" (S), e é justamente o que ela já faz (já fazia isso nos editores anteriores). Já examinamos o caso da ação "!" (shell) no item 4.2.4. Nada de novo no modo bloco. Falta então a ação de gravação (G), que certamente é diferente do modo normal, pois, no modo bloco, queremos gravar apenas o conteúdo do bloco num arquivo, e não o texto inteiro.

A gravação propriamente dita é feita pela rotina **grava**, que já examinamos no item 4.2.2. Você precisa se lembrar de como esta rotina é chamada, para acompanhar a discussão e prosseguir na leitura deste manual.

Para gravar o bloco, poderíamos chamar a rotina **grava**, como se segue (supondo que **arq** seja o nome do arquivo desejado):

```
grava( arq, stdscr, Y_SUP-Y_INF+1, X_SUP-X_INF+1, Y_INF, X_INF )
```

ou seja, grave no arquivo **arq** a informação da janela **stdscr**, com **Y_SUP-Y_INF+1** linhas, **X_SUP-X_INF+1** colunas, a partir da posição (**Y_INF**, **X_INF**)". Poderíamos fazer desta forma. Entretanto, escolhemos uma implementação alternativa que tem como única vantagem a possibilidade de apresentar um novo recurso do CURSES.

No CURSES, podemos definir uma **subjanela** de outra janela. Uma subjanela é manipulada exatamente como uma janela normal, mas **compartilha** sua informação com a janela-mãe. Isto é, se definirmos uma subjanela de **stdscr** na posição (5,5) e com tamanho 10x10, e se fizermos

```
mvwaddstr( subjan, 0, 0, "alo" )
```

os caracteres "alo" estarão presentes nas posições (0,0) a (0,2) da janela **subjan** e também nas posições (5,5) a (5,7) da janela-mãe (**stdscr**, neste caso). Uma subjanela, portanto, oferece uma visão de parte de outra janela, e esta nova "visão" pode ser manipulada como janela. A alteração do texto de qualquer uma das duas janelas afeta o texto presente na outra. Uma subjanela é criada com uma chamada à rotina **subwin**:

```
subjan = subwin( janela-m-e, tamanho vertical,
                            tamanho horizontal,
                            y_inicial, x_inicial )
```

onde **y_inicial** e **x_inicial** são relativos ao início da janela-mãe.

Nossa implementação da gravação de um bloco usa uma subjanela. Veja a seguir.

```
422  acao( ap_res )
423  int *ap_res;     /* onde colocar o resultado: OK ou ERR */
424  {

...
431       JANELA  *jan;        /* janela a gravar */

...
443       case ACAO_GRAVA:
444           if( bloco.existe ) {     /* gravacao do bloco */
445               tam_vert = Y_SUP - Y_INF + 1;
446               tam_hor  = X_SUP - X_INF + 1;
447               ini_y = 0;
448               ini_x = 0;
449               if( (jan = subwin( stdscr, tam_vert, tam_hor,
450                           Y_INF, X_INF )) == NULL ) {
451                   mens( "Gravacao do bloco falhou. Tecle <RETURN> " );
452                   break;
453               }
454               arq = mens( "Nome do arquivo para gravacao do bloco: " );
455           } else {     /* gravacao de stdscr inteira */
456               tam_vert = LINES;
457               tam_hor = COLS;
458               ini_y = INI_Y;
459               ini_x = INI_X;
460               jan = stdscr;
461               if( *arq_usuario == '\0' ) {
462                   strcpy( arq_usuario,
463                       mens( "Nome do arquivo para gravacao: " ) );
```

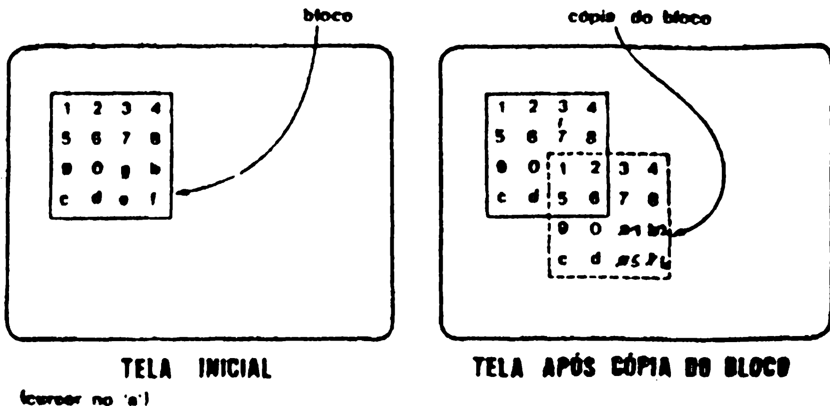
```
464                  sta_arq( arq_usuario );
465              }
466              arq = arq_usuario;
467          }
468          if( (*ap_res = grava( arq,jan,tam_vert,tam_hor,ini_y,ini_x ))
469                          == ERR ) {
470              mens( "Erro de gravacao. Tecle <RETURN>. ");
471          }
472          if( bloco.existe ) {
473              delwin( jan );  /* deleta janela temporaria */
474          }
475          break;

...
492  }
```

Falta apenas discutirmos a operação de cópia de bloco no lugar do cursor. Nossa implementação também usa subjanelas. Examinemos por quê. Poderíamos copiar o bloco para outro lugar com dois laços, como mostrado abaixo (observe a remoção dos atributos das bordas com a máscara **A_TEXTO**:

```
for( y = Y_INF; y <= Y_SUP; y++ ) {
    for( x = X_INF; x <= X_SUP; x++ ) {
        c = mvinch( y, x );
        mvaddch( y-Y_INF+pos_y, x-X_INF+pos_x, c & A_TEXTO )
    }
}
```

No trecho acima, o cursor está na posição (**pos_y**, **pos_x**). Infelizmente, esta implementação tem dois efeitos indesejáveis. Primeiro, se o cursor estiver perto do fim da tela, apenas parte do bloco será copiado, já que a rotina **mvaddch** não adiciona o caractere à janela, se a posição fornecida cair fora da janela. Preferimos uma implementação que copie o bloco inteiro, ou nada, se o bloco não couber na posição-destino.

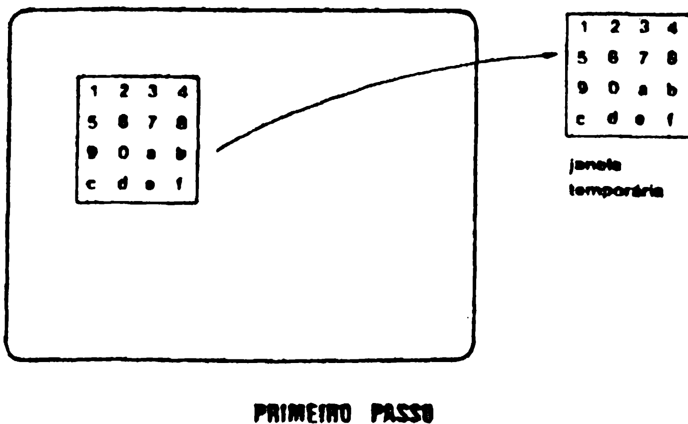
O segundo problema desta tentativa de solução é mostrado na figura abaixo.



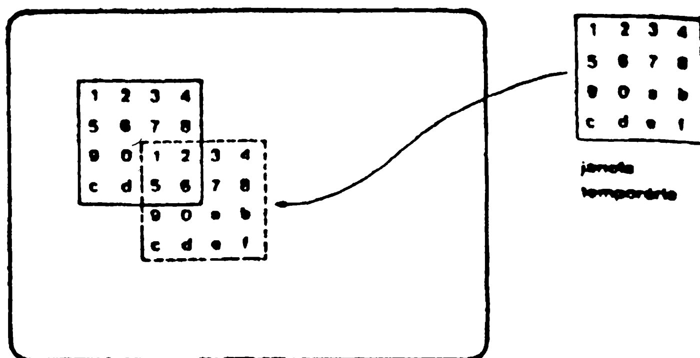
TELA INICIAL

TELA APÓS CÓPIA DO BLOCO

Observe que o próprio ato de copiar pode afetar a informação sendo copiada, se o cursor estiver dentro do bloco. O que precisamos é ilustrado a seguir:



PRIMEIRO PASSO



SEGUNDO PASSO

Em outras palavras, queremos primeiro copiar o bloco para um lugar temporário, antes de copiá-lo para a posição final. Tudo isso pode ser feito usando janelas, já que o CURSES tem a função **overwrite** para copiar uma janela inteira para outra. No nosso caso, precisamos de três janelas para efetuar o trabalho:

- Uma subjanela que se ajusta exatamente em cima do bloco;

- Uma janela (não subjanela) temporária para receber a cópia do bloco;

- Uma subjanela colocada no lugar onde queremos copiar o bloco.

Observe que, ao criar uma janela, devemos especificar sua posição na tela. No caso da segunda janela acima, esta posição não importa, já que não faremos **wrefresh** para esta janela (ela nunca será exibida na tela). O código final que efetua a cópia do bloco é mostrado a seguir.

```
598  trata_bloco()
599  {

...
703          case COPIABLOCO:
704              copia_bloco();
705              break;

...
714  }

...
886  /*
887   * copia_bloco
888   *
889   * copia o bloco marcado na posicao do cursor
890   * Isto eh feito criando uma subjanela do tamanho do bloco
891   * e copiando-a para uma nova janela do mesmo tamanho
892   * com "overwrite". A mesma coisa eh feita para copiar
893   * a nova janela no lugar de destino
894   */
895
896  copia_bloco()
897  {
898
899      int     pos_y, pos_x;   /* onde queremos copiar */
900      int     tam_vert;   /* tamanho vertical do bloco */
901      int     tam_hor;    /* tamanho horizontal do bloco */
902      JANELA      *sjanorig;  /* subjanela de origem */
903      JANELA      *sjandest;  /* subjanela de destino */
904      JANELA      *guardajan; /* janela temporaria para
905                       * copiar o bloco
906                       */
```

```
907
908
909        getyx( stdscr, pos_y, pos_x );
910        tam_vert = Y_SUP - Y_INF + 1;
911        tam_hor  = X_SUP - X_INF + 1;
912        sjanorig = sjandest = guardajan = NULL;
913
914        /* elimina a borda do bloco na janela stdscr
915         * para nao copiar atributos de video ou os '*'
916         */
917        elim_borda();
918
919        if( (sjanorig = subwin( stdscr, tam_vert, tam_hor,
920                       Y_INF, X_INF )) == NULL
921         || (sjandest = subwin( stdscr, tam_vert, tam_hor,
922                       pos_y, pos_x )) == NULL
923         || (guardajan = newwin( tam_vert, tam_hor, 0, 0 )) == NULL
924         || overwrite( sjanorig, guardajan ) == ERR
925         || overwrite( guardajan, sjandest ) == ERR ) {
926
927            beep();
928            mens( "Copia falhou. Tecle <RETURN>. " );
929        } else {    /* copia deu certo */
930            wnoutrefresh( sjandest );
931        }
932
933        /* elimine as janelas temporarias */
934        if( sjanorig != NULL ) {
935            delwin( sjanorig );
936        }
937        if( sjandest != NULL ) {
938            delwin( sjandest );
939        }
940        if( guardajan != NULL ) {
941            delwin( guardajan );
942        }
943
944
945        desenha_borda();
946    }
```

Cabem duas observações a respeito da rotina **copia_bloco.** Primeiro, as bordas do bloco não devem ser copiadas para o destino. Daí as linhas 917 e 945. Segundo, quando tudo dá certo, chamamos a rotina **wnoutrefresh.** O que é isso? Você sabe agora que o CURSES pode manipular várias janelas de uma vez. Para descobrir exatamente como desenhar a tela da forma mais rápida possível, o CURSES mantém uma janela interna (chamada **curscr**), que contém o que o programador deseja que seja colocado na tela. A operação **wefresh**, então, é realizada em duas etapas. Primeiro a informação é passada da janela apropriada para **curscr** e então a informação que foi mudada em **curscr** é desenhada na tela do terminal. Estas duas etapas são mostradas abaixo, onde fornecemos a implementação interna da rotina **wrefresh.**

```
wrefresh( jan )
JANELA *jan;


wnoutrefresh( jan ); /* atualiza curscr */
doupdate(); /* atualiza a tela */
```

Se quiséssemos atualizar várias janelas na tela, poderíamos usar o código:

```
wnoutrefresh( jan1 );
wnoutrefresh( jan2 );
wnoutrefresh( jan3 );
doupdate();
```

Isto seria mais rápido do que chamar **wrefresh** três vezes, já que uma única chamada a **update** seria feita.

Voltemos à rotina **copia_bloco.** Uma vez o bloco copiado, como fazer com que a cópia apareça na tela? Bastaria uma chamada a **refresh** (linha 625)? Não! Já falamos do problema de janelas sobrepostas (item 4.2.1). Poderíamos chamar

```
touchwin( stdscr )
```

na linha 930 (em vez de chamar **wnoutrefresh**), mas isto seria mais lento, já que a janela **stdscr** inteira seria copiada para **curscr.** Nossa solução é forçar a atualização apenas da área que mudou como resultado da cópia do bloco, o que é feito na linha 930:

```
930        wnoutrefresh( sjandest );
```

Página intencionalmente em branco.

CAPÍTULO 5
RECURSOS ADICIONAIS

Este capítulo é diferente dos anteriores. Não escreveremos um programa completo para apresentar novos recursos do CURSES, pois quase todos os mais importantes já foram apresentados.

Nem pretendemos aqui apresentar todas as demais rotinas, pois julgamos que o Módulo 3 as esclarece de forma adequada. Discutiremos brevemente três assuntos que merecem destaque.

5.1 CONTROLE DA ENTRADA

Algumas rotinas do CURSES que controlam a leitura de caracteres do teclado foram discutidas nos Capítulos 2 a 4. São elas:

- **cbreak**;
- **def_prog_mode**;
- **def_shell_mode**;
- **echo**; e
- **keypad.**

Discutiremos a seguir três outras rotinas deste conjunto.

A rotina principal de entrada do CURSES (**wgetch**) é do tipo **bloqueado**, isto é, **wgetch** espera, até que o usuário digite algo no teclado, e retorna então o valor do caractere digitado. Em algumas aplicações especiais, uma entrada bloqueada não é adequada. Para tornar a leitura de caracteres através de uma janela **jan** não-bloqueada, usa-se:

```
nodelay( jan, TRUE )
```

Neste caso, **wgetch(jan)** retorna imediatamente com o valor do caractere já digitado ou com o valor **ERR**, se não houver caracteres digitados pendentes na entrada. Uma rotina semelhante é

```
tem_pendencia()
```

que retorna **TRUE**, se houver caracteres pendentes (digitados, mas não lidos) na entrada, e **FALSE**, caso contrário.

A terceira rotina de interesse, **typeahead**, também diz respeito aos caracteres pendentes digitados pelo usuário. A atualização da tela do terminal pode ser demorada, principalmente quando a tela inteira deve ser redesenhada, uma operação que leva cerca de dois segundos a uma velocidade de 9.600 bauds. Durante esta atualização, o usuário freqüentemente continuará digitando algo no teclado. Se ele estiver digitando mais rapidamente do que as atualizações feitas na tela, podemos chegar à conclusão de que não está tão interessado na informação presente na tela. Se ele realmente quiser ver o que está na tela, interromperá a digitação. Com base neste fato, a rotina **doupdate**, que atualiza a tela (ver o item 4.3.4) retorna imediatamente, **sem** atualizar a tela, se houver caracteres pendentes na entrada. Você pode ver este efeito com os editores apresentados nos capítulos anteriores. Execute edit3 e digite rapidamente. Observará

que os caracteres que você digita só aparecem quando você finalmente faz uma pausa, ou diminui o ritmo de digitação. Se você não gostar deste efeito, pode avisar o CURSES para sempre atualizar a tela, apesar da existência de caracteres pendentes, com a seguinte chamada:

```
typeahead( -1 )
```

Não apresentamos um programa que use as rotinas descritas neste capítulo, mas julgamos que um exemplo seja útil. Considere um editor de textos baseado em tela, tal como os que projetamos, mas que possibilite a edição de textos grandes. Quando o usuário deseja "andar" no texto, poderá usar um comando para "avançar uma tela" no texto. Vamos supor agora que o usuário deseje avançar rapidamente no texto, digitando este comando cinco vezes. Não adianta atualizar a tela cinco vezes, pois sabemos que o usuário está interessado apenas na quinta tela apresentada. Até aqui, tudo bem: o CURSES se encarrega de verificar a existência de caracteres pendentes na entrada e só apresentará a última tela (quando o usuário parar de digitar). Mas será que você quer que o CURSES **sempre** faça isso? Vamos supor que você queira este comportamento apenas para o comando "avança uma tela", ou seja, você quer que o CURSES normalmente **não** faça a verificação de caracteres pendentes (por exemplo, enquanto o usuário digita texto). Como proceder? A solução é usar **typeahead**, para desligar o tratamento de pendências, e implementar este tratamento no próprio editor, usando **tem_pendencia**. O seguinte esqueleto de código esclarece a situação:

```
comandos()
{

    ...

    while( TRUE ) {
        if( n-o houve grandes mudanlas na janela ||
            ! tem_pendencia() ) {
            refresh();
        }
        switch( cmd = getch() ) {
        case ESQUERDA:       /* comando rTpido */
            ...
        case AVANCA_TELA:   /* comando lento */
            avanca_tela();
            anotar que houve grandes mudancas na janela
            ...
        }
    }
}
```

Observe que a rotina **refresh** nem sempre é chamada antes de **getch**. Se o comando **AVANCA_TELA** for digitado rapidamente, repetidas vezes, o **if** será falso, e **refresh** não será chamado, até que **getch** seja chamado um número suficiente de vezes, para obter todos os caracteres digitados pelo usuário. Finalmente, não havendo mais caracteres pendentes na entrada, **refresh** é chamado.

5.2 ATRIBUTOS DE VÍDEO

Atributos de vídeo foram discutidos nos Capítulos 3 e 4. Nos editores aí apresentados, usamos apenas o atributo padrão de destaque, chamado **A_STANDOUT**. Outros atributos de vídeo são reconhecidos e manipulados pelo CURSES, quais sejam:

A_REVERSE	reverso
A_BOLD	intensificado
A_DIM	atenuado
A_BLINK	piscante
A_UNDERLINE	sublinhado

Tais atributos podem ser associados a um único caractere, com as rotinas **addch** e **insch**. Por exemplo:

```
addch( c | A_BOLD );
insch( c | A_BOLD );
```

Por outro lado, é possível ligar atributos para uma janela, de forma que qualquer texto adicionado à janela adquira tais atributos. Isto é feito com a rotina **attron**. Por exemplo:

```
attron( A_BOLD );
wprintw( "BLOCO" );
```

Atributos da janela podem ser desligados, da seguinte forma:

```
attroff( A_BOLD )
```

Observe que mais de um atributo pode ser usado de uma vez, sem afetar os demais atributos da janela:

```
attron( A_BOLD | A_BLINK )
```

Como observação final, vale lembrar que certos atributos (ou todos) podem não ser representáveis em determinados terminais.

5.3 JANELAS DINÂMICAS (PADS)

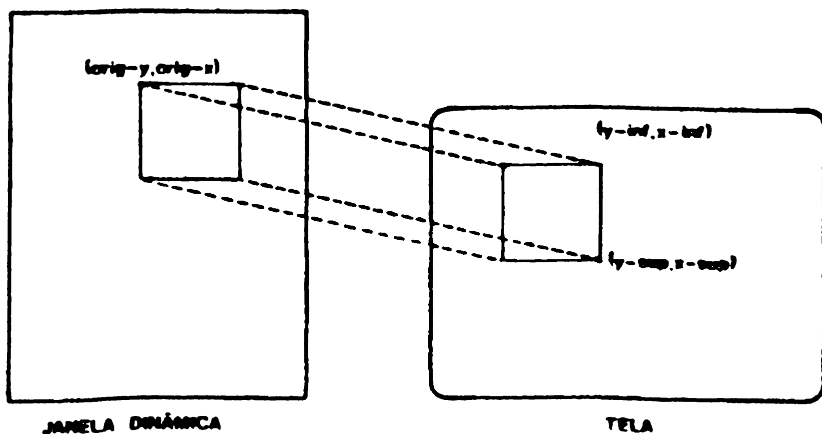
Apesar do CURSES permitir a criação de janelas maiores que a tela do terminal, deve estar claro que tais janelas não podem ser exibidas na tela, isto é, não podem ser sujeitas à função **wrefresh.** Entretanto, algumas raras aplicações precisam manipular janelas grandes. Apenas porções de tais janelas seriam exibidas na tela de cada vez. É possível implementar este esquema de controle, usando uma combinação das rotinas **newwin**, **subwin** e **mvwin.** Por outro lado, o CURSES oferece janelas especiais, que chamamos **janelas dinâmicas** (**pads**, em inglês), usadas justamente para este propósito. Uma janela dinâmica tem um tamanho, mas não possui posição na tela do terminal. É criada com a seguinte chamada

```
jan = newpad( tamanho vertical, tamanho horizontal )
```

Todas as rotinas de manipulação que descrevemos neste módulo podem ser aplicadas a janelas dinâmicas, mas não se pode aplicar-lhes a rotina **wrefresh**. Para exibir uma parte de uma janela dinâmica na tela, usa-se a rotina **prefresh**, que, além de especificar a janela desejada, deve informar que parte da janela deve ser exibida, e onde na tela. A sintaxe é a seguinte:

```
prefresh( jandin, orig_y, orig_x, y_inf, x_inf, y_sup, x_sup )
```

O efeito desta chamada ficará claro, examinando-se a figura abaixo:



Página intencionalmente em branco.

CAPÍTULO 6
BIBLIOTECA TERMINFO

O capítulo introdutório deste manual expôs os objetivos do pacote CURSES, um dos quais é escrever programas com independência de terminal. O CURSES atinge esta independência através de uma outra biblioteca de funções, chamada **TERMINFO.** Neste capítulo, descrevemos as rotinas que compõem a biblioteca. Tais rotinas são usadas diretamente por programas aplicativos que precisam manipular o terminal diretamente, sem usar as rotinas do CURSES. Usamos o TERMINFO no editor edit3 (item 4.2.4, linhas 949 a 961 do edit3.c e item 4.3.1, linhas 108 a 113).

A implementação da independência de terminais é conceitualmente simples. Cada tipo de terminal que pode ser usado numa instalação é cadastrado. O cadastro descreve todas as características do terminal (como apagar a tela, posicionar o cursor, ligar atributos de vídeo, etc.). Qualquer programa pode consultar o cadastro do terminal em uso, para descobrir como manipulá-lo. Não pretendemos discutir todos os itens que fazem parte do cadastro, pois este é o objetivo do Módulo 2 deste manual.

As características de um terminal são agrupadas em três categorias:

. Características booleanas

  Por exemplo, "Quando um caractere é colocado na última posição de uma linha, o cursor se move para o início da próxima linha?".

- Características numéricas

  Por exemplo, "O terminal possui quantas linhas?".

- Características do tipo cadeia

  Por exemplo, "Que cadeia de caracteres deve ser enviada ao terminal, para apagar a tela?".

Um programa carrega as características do terminal apropriado numa estrutura de dados (definida no arquivo **term.h**), usando a chamada

```
setupterm( nome, 1, NULL )
```

onde **nome** é o nome pelo qual o terminal é conhecido no cadastro. Normalmente, este nome é obtido da variável do ambiente **TERM**, usando-se a rotina **getenv** da biblioteca da linguagem C (ver o Capítulo 2). É importante observar que nossos três editores não chamaram **setupterm** diretamente, pois esta rotina é chamada automaticamente por **initscr**.

Uma vez carregada a estrutura, as características do terminal são obtidas através dos seus nomes simbólicos. O Módulo 2 descreve cada um destes nomes. Como exemplo, a cadeia que liga o atributo **A_STANDOUT** do terminal chama-se **enter_standout_mode**, o número de espaços ocupados na tela por um atributo de vídeo chama-se **magic_cookie_glitch**. Se um terminal **não** possui uma determinada característica, os valores assumidos são:

Característica booleana: FALSE;
Característica numérica: -1;
Característica do tipo cadeia: NULL.

O seguinte trecho de código foi usado no item 4.3.1 e deve estar mais claro agora:

```
108  /*
109   * O desenho das bordas do bloco pode usar atributos
110   * de video se o terminal for "razoavel"
111   */
112  #define PODE_USAR_STANDOUT (magic_cookie_glitch < 1 && \
113                  enter_standout_mode != NULL)
```

Características do tipo cadeia são emitidas para o terminal com a rotina **tputs**. Veja como foi usada no item 4.2.4:

```
959 tputs( clear_screen, 1, _sai_carac );
```

O segundo parâmetro de **tputs** é explicado em detalhes no Módulo 2. O terceiro parâmetro é a função usada por **tputs** para enviar caracteres para o terminal. **_sai_carac** é uma rotina interna do TERMINFO e é funcionalmente equivalente a **putchar** da biblioteca da linguagem C.

Certas cadeias que descrevem as características de um terminal devem receber parâmetros. Por exemplo, a cadeia que descreve a movimentação do cursor depende da posição (y, x) desejada. A função **tparm** é usada para aplicar parâmetros atuais à cadeia que contém parâmetros formais, para formar a cadeia final a ser enviada ao terminal. Por exemplo, a movimentação do cursor do terminal é feita da seguinte forma:

```
tputs( tparm(cursor_address, y, x), 1, _sai_carac)
```

onde **cursor_address** é a cadeia para movimentação do cursor, e (y,x) é a posição desejada.

O TERMINFO possui uma rotina que movimenta o cursor de forma otimizada (não use **cursor_address**, se houver uma melhor maneira), mas, para usá-la, você deve conhecer a localização **atual** do cursor. Para posicioná-lo em (dy, dx), estando em (y, x), chame:

```
mvcur( y, x, dy, dx )
```

CAPÍTULO 7
COMPILANDO E EXECUTANDO UM PROGRAMA ...

7.1 COMPILAÇÃO

A fase de compilação para um programa que use CURSES e/ou TERMINFO é idêntica à seqüência normalmente usada no seu sistema. Usualmente, você usaria

```
cc -c prog.c
```

para produzir o módulo-objeto prog.o.

Para ligar o(s) módulo(s)-objeto com a bilblioteca CURSES/TERMINFO, você deve ligar o arquivo /usr/lib/libcurses.a com seu programa. Normalmente, isto é feito como se segue:

```
cc prog.o -lcurses
```

Naturalmente, a compilação e ligação podem ser feitas no mesmo passo:

```
cc prog.c -lcurses
```

deixando o resultado no arquivo a.out, ou usando

```
cc prog.c -lcurses -o prog
```

deixando o resultado no arquivo prog.

7.2 EXECUÇÃO

Antes de executar um programa que use as bibliotecas CURSES/TERMINFO, você deve tomar o cuidado de informá-las que tipo de terminal está usando. Na sua instalação, cada tipo de terminal deve receber um nome. Fale com o responsável por sua instalação, para descobrir os nomes dados aos vários tipos de terminais. Este nome deve ser colocado na variável do ambiente **TERM**, como mostrado abaixo, no caso de um terminal do tipo TI 200 (este comando é dado para o shell):

```
TERM=TI200
```

Esta variável deve então ser colocada no ambiente:

```
export TERM
```

Agora, você pode executar o programa desejado, e as bibliotecas CURSES/TERMINFO saberão que o terminal que elas manipularão é do tipo TI 200.

7.3 DESCRIÇÃO DO PROGRAMA edit1

```
1   /*
2    * edit1 - um editor de tela simplificado
    *
4    * Sintaxe: edit1 arquivo
5    */
6
7   #include    <curses.h>
8   #include    <signal.h>
9   #include    <ctype.h>
10  #include    <string.h>
11
12
13   /* definicao de constantes e macros */
14
15   #define MAXCOLUNA   132 /* maior tela aceitavel */
16
17   /* posicao inicial do texto */
18   #define INI_Y   1   /* linha inicial do texto */
19   #define INI_X   0   /* coluna inicial do texto */
20
21   /* modos de operacao */
22   #define INSERCAO    0
23   #define ADICAO      1
24
25   #define CTRL(c)       ((c) - 'A' + '\001')
26   #define min(x, y)     ((x) < (y) ? (x) : (y))
27
28   /* comandos do editor */
29   #define ACAO          CTRL('A')
30   #define MUDAMODO      CTRL('V')
31   #define ESQUERDA      CTRL('S')
32   #define DIREITA       CTRL('D')
33   #define CIMA          CTRL('E')
34   #define BAIXO         CTRL('X')
35   #define PROXLINHA     ('\n')
36
37   /* acoes possiveis */
38   #define ACAO_SAI      'S'
39   #define ACAO_GRAVA    'G'
40
41   /* regioes da linha de status */
42   #define COL_ACAO      (COLS - 30) /* regiao "acao" */
43   #define COL_ARQUIVO (COLS/2 - 7)     /* regiao "arquivo" */
44   #define COL_MODO      1         /* regiao "modo" */
45
46   /* digitar s, S ou CONTROLE S deve ser a mesma coisa */
```

```
47  #define PADRONIZA(c)    (toupper((c) | 0x40))
48
49  /* o que fazer apos uma acao */
50  #define CONTINUA    0   /* continua a editar */
51  #define FIM     1   /* termina o programa */
52
53
54
55  /*
56   * Rotina principal
57   * Inicializa o curses
58   * Le o arquivo do usuario
59   * Interpreta os comandos do usuario
60   * sai, fornecendo o status do comando
61   */
62
63  main( argc, argv )
64  int argc;
65  char    *argv[];
66  {
67
68      if( inicializa( argc, argv ) == ERR ) { /* nao inicializou */
69          exit( 1 );
70      } else if( le_arq( argv[0], argv[1] ) == ERR    /* nao leu */
71          || comandos( argv[1] ) == ERR ) {   /* erro nos comandos */
72
73          tchau( 1 ); /* nao retorna */
74      } else {
75          tchau( 0 ); /* nao retorna */
76      }
77  }
78
79
80  /*
81   * tchau
82   * sai, fornecendo o status do sistema
83   */
84
85  tchau( status )
86  int status; /* status do programa */
```

```
87  {
88
89      endwin();   /* encerra o curses */
90      exit( status );
91  }
92
93
94  /*
95   * inicializa
96   * verifica a sintaxe do comando
97   * inicializa curses e o terminal
98   */
99
100  inicializa( argc, argv )
101  int argc;
102  char    *argv[];
103  {
104
105      if( argc != 2 ) {
106          fprintf(stderr, "Sintaxe: %s arquivo\n", argv[0] );
107          return( ERR );
108      } else {    /* tudo OK */
109          signal( SIGINT, SIG_IGN );
110          initscr();
111          noecho();
112          cbreak();
113          nl();
114          noxon();
115      }
116      return( OK );
117  }
118
119
120  /*
121   * le_arq
122   * abre e le o arquivo do usuario
123   * colocando o texto na janela stdscr
124   */
125
126  le_arq( prog, arquivo )
```

```
127  char    *prog;       /* nome do programa */
128  char    *arquivo;    /* nome do arquivo do usuario */
129  {
130
131      char        *fgets();
132      register int    lin;    /* para varrer as linhas */
133      FILE        *fp;    /* arquivo aberto para leitura */
134      int     tam;    /* tamanho da leitura */
135      char        buf[MAXCOLUNA + 2]; /* buffer de leitura */
136
137      tam = min( MAXCOLUNA, COLS + 2 );    /* +2 por causa do \n
138                           * e do \0 finais
139                           */
140      if( (fp = fopen( arquivo, access( arquivo, 0 ) >= 0 ? "r" : "w+" )) ==
141                  (FILE *)NULL ) {
142          fprintf(stderr, "%s: nao pode abrir %s\n", prog, arquivo );
143          return( ERR );
144      }
145
146      for( lin = INI_Y; lin < LINES; lin++ ) {
147          if( fgets( buf, tam, fp ) == (char *)NULL ) {
148              if( feof( fp ) ) {  /* fim do arquivo */
149                  break;
150              } else {        /* erro de leitura */
151                  fprintf(stderr, "%s: erro de leitura em %s\n",
152                      prog, arquivo );
153                  return( ERR );
154              }
155          }
156          mvaddstr( lin, 0, buf );     /* coloca o texto na janela */
157      }
158
159      fclose( fp );
160      return( OK );
161  }
162
163
164  /*
165   * comandos
166   * inicializa o cursor, a linha de status e o modo de operacao
167   * interpreta os comandos do usuario
```

```
168  */
169
170  comandos( arquivo )
171  char    *arquivo;   /* nome do arquivo do usuario */
172  {
173
174      int     modo;        /* modo de operacao */
175      int     pos_y, pos_x;   /* posicao atual do cursor */
176      caractere       cmd;         /* caractere digitado pelo usuario */
177      int     resultado;  /* acao deu erro ? */
178
179      move( INI_Y, INI_X );
180      modo = INSERCAO;     /* modo inicial de operacao */
181      sta_arq( arquivo );
182      sta_modo( modo );
183
184      while( TRUE ) {
185          getyx( stdscr, pos_y, pos_x );
186          refresh();
187          switch( cmd = getch() ) {
188          case ACAO:
189              if( acao( arquivo, &resultado ) == FIM ) {
190                  return( resultado );
191              }
192              break;
193          case MUDAMODO:
194              modo = modo == INSERCAO ? ADICAO : INSERCAO;
195              sta_modo( modo );   /* mostra novo modo */
196              break;
197          case ESQUERDA:
198              movimenta( pos_y, pos_x - 1 );
199              break;
200          case DIREITA:
201              movimenta( pos_y, pos_x + 1 );
202              break;
203          case CIMA:
204              movimenta( pos_y - 1, pos_x );
205              break;
206          case BAIXO:
207              movimenta( pos_y + 1, pos_x );
208              break;
```

```
209             case PROXLINHA:
210                 movimenta( pos_y + 1, 0 );
211                 break;
212             default:
213                 if( isprint( cmd ) ) {  /* da para imprimir ? */
214                     if( modo == ADICAO ) {
215                         addch( cmd );
216                     } else {
217                         insch( cmd );
218                         movimenta( pos_y, pos_x + 1 );
219                     }
220                 } else {
221                     beep();
222                 }
223                 break;
224         }
225     }
226 }
227
228
229 /*
230  * sta_arq
231  * exibe o nome do arquivo na linha de status
232  * (somente o ultimo componente, porque o nome completo
233  * pode ser grande)
234  */
235
236 sta_arq( arquivo )
237 char    *arquivo;   /* nome do arquivo do usuario */
238 {
239
240     char    *ult_nome();
241     int lemb_y, lemb_x; /* para voltar o cursor */
242
243     getyx( stdscr, lemb_y, lemb_x );
244     mvprintw( 0, COL_ARQUIVO, "%14s", ult_nome( arquivo ) );
245     move( lemb_y, lemb_x );
246 }
247
248
249 /*
```

```
250   * sta_modo
251   * exibe o modo de operacao na linha de status
252   */
253
254  sta_modo( modo )
255  int modo;       /* o modo de operacao */
256  {
257
258      int lemb_y, lemb_x;     /* para voltar o cursor */
259
260      getyx( stdscr, lemb_y, lemb_x );
261      mvprintw( 0, COL_MODO, "%10s", modo == INSERCAO ? "INSERCAO" :
262                                      " ADICAO" );
263      move( lemb_y, lemb_x );
264  }
265
266
267  /*
268   * movimenta
269   * movimenta o cursor da janela, verificando casos ilegais
270   */
271
272  movimenta( y, x )
273  int y;
274  int x;  /* posicao desejada */
275  {
276
277      if( y < INI_Y ||    /* tentando entrar na linha de status ? */
278          move( y, x ) == ERR ) { /* caindo fora da janela ? */
279          beep();
280          return( ERR );
281      }
282      return( OK );
283  }
284
285
286  /*
287   * acao
288   * pergunta ao usuario o que ele quer fazer (sair ou gravar)
289   * e desempenha a acao
```

```
290   */
291
292  acao( arquivo, ap_res )
293  char    *arquivo;   /* nome do arquivo do usuario */
294  int *ap_res;     /* onde colocar o resultado: OK ou ERR */
295  {
296
297      caractere   resp;   /* resposta do usuario */
298      int ret;    /* valor de retorno da funco acao() */
299
300      *ap_res = OK;   /* supoe que nao havera erro */
301      ret = CONTINUA;
302      resp = pergunta( "Sai(s) ou Grava(g) ? ");
303      switch( PADRONIZA( resp ) ) {
304      case ACAO_SAI:
305          ret = FIM;
306          break;
307      case ACAO_GRAVA:
308          if( (*ap_res = grava( arquivo )) == ERR ) {
309              pergunta( "Erro de grav. Tecle algo. ");
310          }
311          break;
312      default:
313          beep();     /* usuario se arrependeu ou errou */
```

```
314             break;
315         }
316
317         return( ret );
318     }
319
320
321     /*
322      * pergunta
323      * faz uma pergunta e retorna a resposta (um caractere)
324      */
325
326     pergunta( perg )
327     char    *perg;
328     {
329
330         caractere   resp;
331         int lemb_y, lemb_x;     /* para voltar o cursor */
332
```

```
333        getyx( stdscr, lemb_y, lemb_x );
334        mvaddstr( 0, COL_ACAO, perg );
335        refresh();
336        resp = getch();
337        move( 0, COL_ACAO );
338        clrtoeol();
339        move( lemb_y, lemb_x );
340        return( resp );
341    }
342
343
344    /*
345     * grava
346     * grava o texto no arquivo do usuario
347     */
348
349    grava( arquivo )
350    char     *arquivo;   /* nome do arquivo do usuario */
351    {
352
353        int      ret;    /* valor de retorno da funcao grava() */
354        register int    lin;    /* para varrer as linhas da janela */
355        register int    col;    /* para varrer as colunas da janela */
356        int      col_final;  /* coluna final a gravar na linha */
357        FILE         *fp;    /* arquivo aberto para gravacao */
358        int      lemb_y, lemb_x; /* para voltar o cursor */
359
360        if( (fp = fopen( arquivo, "w")) == (FILE *)NULL ) {
361            fprintf(stderr, "Nao pode gravar em %s\n", arquivo );
362            return( ERR );
363        }
364
365        getyx( stdscr, lemb_y, lemb_x );
366        ret = OK;
367        for( lin = INI_Y; ret == OK && lin < LINES; lin++ ) {
368            /* procura a ultima coluna que interessa nesta linha */
369            for( col_final = COLS -1;
```

```
370                 col_final >= 0 && mvinch( lin, col_final ) == ' ';
371                 col_final-- )
372                ;
373
374           for( col = 0; col <= col_final; col++ ) {
375               if( fputc( mvinch( lin, col ), fp ) == EOF ) {
376                   ret = ERR;
377                   break;
378               }
379           }
380
381           if( fputc( '\n', fp ) == EOF ) {
382               ret = ERR;
383           }
384       }
385
386       if( fclose( fp ) == EOF ) {
387           ret = ERR;
388       }
389       move( lemb_y, lemb_x );
390       return( ret );
391   }
392
393
394   /*
395    * ult_nome
396    * retorna o ultimo componente de um nome de arquivo
397    */
398
399   char *
400   ult_nome( s )
401   register char   *s; /* nome original do arquivo */
402   {
403
404       char    *strrchr();
405       char    *p;
406
407       p = strrchr( s, '/' );
408       return( p == NULL ? s : p + 1 );
409   }
```

7.4 DESCRIÇÃO DO PROGRAMA edit2

```
1   /*
2    * edit2 - um editor de tela aperfeicoado
3    *
4    * Sintaxe: edit2 arquivo
5    */
6
7   #include    <curses.h>
8   #include    <signal.h>
9   #include    <isctype.h>
10  #include    <string.h>
11
12
13   /* definicao de constantes e macros */
14
15   #define MAXCOLUNA   132 /* maior tela aceitavel */
16
17   /* posicao inicial do texto */
18   #define INI_Y   1   /* linha inicial do texto */
19   #define INI_X   0   /* coluna inicial do texto */
20
21   /* modos de operacao */
22   #define INSERCAO    0
23   #define ADICAO      1
24
25   #define CTRL(c)       ((c) - 'A' + '\001')
26   #define min(x, y)     ((x) < (y) ? (x) : (y))
27
28   /* comandos do editor */
29   #define ACAO          CTRL('A')
30   #define MUDAMODO      CTRL('V')
31   #define ESQUERDA      CTRL('S')
32   #define DIREITA       CTRL('D')
33   #define CIMA          CTRL('E')
34   #define BAIXO         CTRL('X')
35   #define PROXLINHA     ('\n')
36   #define ZERAATEFIM    CTRL('F')
37   #define ZERALINHA     CTRL('B')
38   #define REM_CARAC     CTRL('C')
39   #define REM_LINHA     CTRL('L')
40   #define INS_LINHA     CTRL('I')
41   #define ZERATUDO      CTRL('T')
42   #define ROLABAIXO     CTRL('W')
43   #define ROLACIMA      CTRL('Z')
44   #define REDESENHA     CTRL('R')
45
46   /* acoes possiveis */
```

```
47  #define ACAO_SAI     'S'
48  #define ACAO_GRAVA   'G'
49
50  /* regioes da linha de status */
51  #define COL_ACAO     (COLS - 30) /* regiao "acao" */
52  #define COL_ARQUIVO (COLS/2 - 7)    /* regiao "arquivo" */
53  #define COL_MODO     1         /* regiao "modo" */
54
55  /* digitar s, S ou CONTROLE S deve ser a mesma coisa */
56  #define PADRONIZA(c)     (toupper((c) | 0x40))
57
58  /* o que fazer apos uma acao */
59  #define CONTINUA     0    /* continua a editar */
60  #define FIM      1    /* termina o programa */
61
62
63
64  /*
65   * Rotina principal
66   * Inicializa o curses
67   * Le o arquivo do usuario
68   * Interpreta os comandos do usuario
69   * sai, fornecendo o status do comando
70   */
71
72  main( argc, argv )
73  int argc;
74  char   *argv[];
75  {
76
77      if( inicializa( argc, argv ) == ERR ) { /* nao inicializou */
78          exit( 1 );
79      } else if( le_arq( argv[0], argv[1] ) == ERR    /* nao leu */
80          || comandos( argv[1] ) == ERR ) {   /* erro nos comandos */
81
82          tchau( 1 ); /* nao retorna */
83      } else {
84          tchau( 0 ); /* nao retorna */
85      }
86  }
87
```

```
 88
 89  /*
 90   * tchau
 91   * sai, fornecendo o status do sistema
 92   */
 93
 94  tchau( status )
 95  int status; /* status do programa */
 96  {
 97
 98      endwin();   /* encerra o curses */
 99      exit( status );
100  }
101
102
103  /*
104   * inicializa
105   * verifica a sintaxe do comando
106   * inicializa curses e o terminal
107   */
108
109  inicializa( argc, argv )
110  int argc;
111  char    *argv[];
112  {
113
114      if( argc != 2 ) {
115          fprintf(stderr, "Sintaxe: %s arquivo\n", argv[0] );
116          return( ERR );
117      } else {    /* tudo OK */
118          signal( SIGINT, SIG_IGN );
119          initscr();
120          noecho();
121          cbreak();
122          nl();
123          noxon();
124          setscrreg( INI_Y, LINES - 1 );
125          idlok( stdscr, TRUE );
126          keypad( stdscr, TRUE );
127          modoabc( TRUE );
128          acent( TRUE );
```

```
129     }
130     return( OK );
131 }
132
133
134 /*
135  * le_arq
136  * abre e le o arquivo do usuario
137  * colocando o texto na janela stdscr
138  */
139
140 le_arq( prog, arquivo )
141 char    *prog;      /* nome do programa */
142 char    *arquivo;   /* nome do arquivo do usuario */
143 {
144
145     char        *fgets();
146     register int    lin;    /* para varrer as linhas */
147     FILE        *fp;    /* arquivo aberto para leitura */
148     int     tam;    /* tamanho da leitura */
149     char        buf[MAXCOLUNA + 2]; /* buffer de leitura */
150
151     tam = min( MAXCOLUNA, COLS + 2 );   /* +2 por causa do \n
152                         * e do \0 finais
153                         */
154     if( (fp = fopen( arquivo, access( arquivo, 0 ) >= 0 ? "r" : "w+" )) ==
155                 (FILE *)NULL ) {
156         fprintf(stderr, "%s: nao pode abrir %s\n", prog, arquivo );
157         return( ERR );
158     }
159
160     for( lin = INI_Y; lin < LINES; lin++ ) {
161         if( fgets( buf, tam, fp ) == (char *)NULL ) {
162             if( feof( fp ) ) {  /* fim do arquivo */
163                 break;
164             } else {        /* erro de leitura */
165                 fprintf(stderr, "%s: erro de leitura em %s\n",
166                     prog, arquivo );
167                 return( ERR );
168             }
169         }
```

```
170             mvaddstr( lin, 0, buf );    /* coloca o texto na janela */
171         }
172
173         fclose( fp );
174         return( OK );
175     }
176
177
178     /*
179      * comandos
180      * inicializa o cursor, a linha de status e o modo de operacao
181      * interpreta os comandos do usuario
182      */
183
184     comandos( arquivo )
185     char    *arquivo;   /* nome do arquivo do usuario */
186     {
187
188         int      modo;        /* modo de operacao */
189         int      pos_y, pos_x;   /* posicao atual do cursor */
190         caractere   cmd;          /* caractere digitado pelo usuario */
191         int      resultado;  /* acao deu erro ? */
192
193         move( INI_Y, INI_X );
194         modo = INSERCAO;     /* modo inicial de operacao */
195         sta_arq( arquivo );
196         sta_modo( modo );
197
198         while( TRUE ) {
199             getyx( stdscr, pos_y, pos_x );
200             refresh();
201             switch( cmd = getch() ) {
202             case ACAO:
203                 if( acao( arquivo, &resultado ) == FIM ) {
204                     return( resultado );
205                 }
206                 break;
207             case MUDAMODO:
208                 modo = modo == INSERCAO ? ADICAO : INSERCAO;
209                 sta_modo( modo );   /* mostra novo modo */
210                 break;
```

```
211         case ESQUERDA:
212         case KEY_LEFT:
213             movimenta( pos_y, pos_x - 1 );
214             break;
215         case DIREITA:
216         case KEY_RIGHT:
217             movimenta( pos_y, pos_x + 1 );
218             break;
219         case CIMA:
220         case KEY_UP:
221             movimenta( pos_y - 1, pos_x );
222             break;
223         case BAIXO:
224         case KEY_DOWN:
225             movimenta( pos_y + 1, pos_x );
226             break;
227         case PROXLINHA:
228             movimenta( pos_y + 1, 0 );
229             break;
230         case ZERAATEFIM:
231             clrtobot();
232             break;
233         case ZERALINHA:
234             clrtoeol();
235             break;
236         case REM_CARAC:
237             delch();
238             break;
239         case REM_LINHA:
240             deleteln();
241             break;
242         case INS_LINHA:
243             insertln();
244             break;
245         case ZERATUDO:
246             clear();
247             sta_arq( arquivo );
248             sta_modo( modo );
249             move( INI_Y, INI_X );
250             break;
251         case ROLABAIXO:
```

```
252                 nscroll( -1 );
253                 movimenta( pos_y, pos_x );
254                 break;
255             case ROLACIMA:
256                 nscroll( 1 );
257                 movimenta( pos_y, pos_x );
258                 break;
259             case REDESENHA:
260                 clearok( stdscr, TRUE );
261                 break;
262             default:
263                 if( cmd == erasechar() ) {
264                     if( movimenta( pos_y, pos_x - 1 ) == OK ) {
265                         delch();
266                     }
267                 } else if( isprint( cmd ) ) {   /* da para imprimir ? */
268                     if( modo == ADICAO ) {
269                         addch( cmd );
270                     } else {
271                         insch( cmd );
272                         movimenta( pos_y, pos_x + 1 );
273                     }
274                 } else {
275                     beep();
276                 }
277                 break;
278             }
279         }
280     }
281
282
283     /*
284      * sta_arq
285      * exibe o nome do arquivo na linha de status
286      * (somente o ultimo componente, porque o nome completo
287      * pode ser grande)
288      */
289
290     sta_arq( arquivo )
291     char    *arquivo;   /* nome do arquivo do usuario */
292     {
```

```
293
294       char    *ult_nome();
295       int lemb_y, lemb_x; /* para voltar o cursor */
296
297       getyx( stdscr, lemb_y, lemb_x );
298       mvprintw( 0, COL_ARQUIVO, "%14s", ult_nome( arquivo ) );
299       move( lemb_y, lemb_x );
300   }
301
302
303   /*
304    * sta_modo
305    * exibe o modo de operacao na linha de status
306    */
307
308   sta_modo( modo )
309   int modo;        /* o modo de operacao */
310   {
311
312       int lemb_y, lemb_x;     /* para voltar o cursor */
313
314       getyx( stdscr, lemb_y, lemb_x );
315       standout();
316       mvprintw( 0, COL_MODO, "%10s", modo == INSERCAO ? "INSERCAO" :
317                                  " ADICAO" );
318       standend();
319       move( lemb_y, lemb_x );
320   }
321
322
323   /*
324    * movimenta
325    * movimenta o cursor da janela, verificando casos ilegais
326    */
327
328   movimenta( y, x )
329   int y;
330   int x;  /* posicao desejada */
331   {
332
333       if( y < INI_Y ||    /* tentando entrar na linha de status ? */
```

```
334             move( y, x ) == ERR ) { /* caindo fora da janela ? */
335             beep();
336             return( ERR );
337         }
338         return( OK );
339     }
340
341
342     /*
343      * acao
344      * pergunta ao usuario o que ele quer fazer (sair ou gravar)
345      * e desempenha a acao
346      */
347
348     acao( arquivo, ap_res )
349     char    *arquivo;   /* nome do arquivo do usuario */
350     int *ap_res;    /* onde colocar o resultado: OK ou ERR */
351     {
352
353         caractere   resp;   /* resposta do usuario */
354         int ret;    /* valor de retorno da funco acao() */
355
356         *ap_res = OK;   /* supoe que nao havera erro */
357         ret = CONTINUA;
358         resp = pergunta( "Sai(s) ou Grava(g) ? ");
359         switch( PADRONIZA( resp ) ) {
360         case ACAO_SAI:
361             ret = FIM;
362             break;
363         case ACAO_GRAVA:
364             if( (*ap_res = grava( arquivo )) == ERR ) {
365                 pergunta( "Erro de grav. Tecle algo. ");
366             }
367             break;
368         default:
369             beep();     /* usuario se arrependeu ou errou */
370             break;
371         }
372
373         return( ret );
374     }
375
376
```

```
377  /*
378   * pergunta
379   * faz uma pergunta e retorna a resposta (um caractere)
380   */
381
382  pergunta( perg )
383  char    *perg;
384  {
385
386      caractere   resp;
387      int lemb_y, lemb_x;     /* para voltar o cursor */
388
389      getyx( stdscr, lemb_y, lemb_x );
390      mvaddstr( 0, COL_ACAO, perg );
391      refresh();
392      resp = getch();
393      move( 0, COL_ACAO );
394      clrtoeol();
395      move( lemb_y, lemb_x );
396      return( resp );
397  }
398
399
400  /*
401   * grava
402   * grava o texto no arquivo do usuario
403   */
404
405  grava( arquivo )
406  char    *arquivo;   /* nome do arquivo do usuario */
407  {
408
409      int     ret;    /* valor de retorno da funcao grava() */
410      register int    lin;    /* para varrer as linhas da janela */
411      register int    col;    /* para varrer as colunas da janela */
412      int     col_final;  /* coluna final a gravar na linha */
413      FILE        *fp;    /* arquivo aberto para gravacao */
414      int     lemb_y, lemb_x; /* para voltar o cursor */
415
416      if( (fp = fopen( arquivo, "w")) == (FILE *)NULL ) {
417          fprintf(stderr, "Nao pode gravar em %s\n", arquivo );
418          return( ERR );
419      }
420
```

```
421      getyx( stdscr, lemb_y, lemb_x );
422      ret = OK;
423      for( lin = INI_Y; ret == OK && lin < LINES; lin++ ) {
424          /* procura a ultima coluna que interessa nesta linha */
425          for( col_final = COLS -1;
426               col_final >= 0 && mvinch( lin, col_final ) == ' ';
427               col_final-- )
428              ;
429
430          for( col = 0; col <= col_final; col++ ) {
431              if( fputc( mvinch( lin, col ), fp ) == EOF ) {
432                  ret = ERR;
433                  break;
434              }
435          }
436
437          if( fputc( '\n', fp ) == EOF ) {
438              ret = ERR;
439          }
440      }
441
442      if( fclose( fp ) == EOF ) {
443          ret = ERR;
444      }
445      move( lemb_y, lemb_x );
446      return( ret );
447  }
448
449
450  /*
451   * ult_nome
452   * retorna o ultimo componente de um nome de arquivo
453   */
454
455  char *
456  ult_nome( s )
457  register char   *s; /* nome original do arquivo */
458  {
459
460      char    *strrchr();
```

```
461        char    *p;
462
463        p = strrchr( s, '/' );
464        return( p == NULL ? s : p + 1 );
465    }
```

ARQUIVO isctype.h

```
 1   /*
 2    * isctype.h
 3    *
 4    * Semelhante a ctype.h, mas funciona para caracteres ABICOMP
 5    */
 6
 7   #define _U  01
 8   #define _L  02
 9   #define _N  04
10   #define _S  010
11   #define _P  020
12   #define _C  040
13   #define _B  0100
14   #define _X  0200
15
16   extern  char    _ctype[];
17
18   #define isalpha(c)  ((_ctype+1)[(c)&0377]&(_U|_L))
19   #define isupper(c)  ((_ctype+1)[(c)&0377]&_U)
20   #define islower(c)  ((_ctype+1)[(c)&0377]&_L)
21   #define isdigit(c)  ((_ctype+1)[(c)&0377]&_N)
22   #define isxdigit(c) ((_ctype+1)[(c)&0377]&(_X|_N))
23   #define isspace(c)  ((_ctype+1)[(c)&0377]&_S)
24   #define ispunct(c)  ((_ctype+1)[(c)&0377]&_P)
25   #define isalnum(c)  ((_ctype+1)[(c)&0377]&(_U|_L|_N))
26   #define isprint(c)  ((_ctype+1)[(c)&0377]&(_P|_U|_L|_N|_B))
27   #define isgraph(c)  ((_ctype+1)[(c)&0377]&(_P|_U|_L|_N))
28   #define iscntrl(c)  ((_ctype+1)[(c)&0377]&_C)
29   #define isascii(c)  ((unsigned)(c)<=0177)
30   #define _toupper(c) ((c)-'a'+'A')
31   #define _tolower(c) ((c)-'A'+'a')
32   #define toascii(c)  ((c)&0177)
33   #define toupper(c)  ( islower(c) ? _toupper(c) : (c) )
34   #define tolower(c)  ( isupper(c) ? _tolower(c) : (c) )
```

```
1   #include   <isctype.h>
2
3   char     _ctype[] = { 0,
4
5   /*       0      1      2      3      4      5      6      7  */
6
7   /*  0*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
8   /* 10*/ _C,    _S|_C, _S|_C, _S|_C, _S|_C, _S|_C, _C,    _C,
9   /* 20*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
10  /* 30*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
11  /* 40*/ _S|_B, _P,    _P,    _P,    _P,    _P,    _P,    _P,
12  /* 50*/ _P,    _P,    _P,    _P,    _P,    _P,    _P,    _P,
13  /* 60*/ _N|_X, _N|_X, _N|_X, _N|_X, _N|_X, _N|_X, _N|_X, _N|_X,
14  /* 70*/ _N|_X, _N|_X, _P,    _P,    _P,    _P,    _P,    _P,
15  /*100*/ _P,    _U|_X, _U|_X, _U|_X, _U|_X, _U|_X, _U|_X, _U,
16  /*110*/ _U,    _U,    _U,    _U,    _U,    _U,    _U,    _U,
17  /*120*/ _U,    _U,    _U,    _U,    _U,    _U,    _U,    _U,
18  /*130*/ _U,    _U,    _U,    _P,    _P,    _P,    _P,    _P,
19  /*140*/ _P,    _L|_X, _L|_X, _L|_X, _L|_X, _L|_X, _L|_X, _L,
20  /*150*/ _L,    _L,    _L,    _L,    _L,    _L,    _L,    _L,
21  /*160*/ _L,    _L,    _L,    _L,    _L,    _L,    _L,    _L,
22  /*170*/ _L,    _L,    _L,    _P,    _P,    _P,    _P,    _C,
23  /* CARACTERES ABICOMP COMECAM AQUI */
24  /*200*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
25  /*210*/ _C,    _S|_C, _S|_C, _S|_C, _S|_C, _S|_C, _C,    _C,
26  /*220*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
27  /*230*/ _C,    _C,    _C,    _C,    _C,    _C,    _C,    _C,
28  /*240*/ _S|_B, _U,    _U,    _U,    _U,    _U,    _U,    _U,
29  /*250*/ _U,    _U,    _U,    _U,    _U,    _U,    _U,    _U,
30  /*260*/ _U,    _U,    _U,    _U,    _U,    _U,    _U,    _U,
31  /*270*/ _U,    _U,    _U,    _P,    _P,    _P,    _P,    _P,
32  /*300*/ _P,    _L,    _L,    _L,    _L,    _L,    _L,    _L,
33  /*310*/ _L,    _L,    _L,    _L,    _L,    _L,    _L,    _L,
34  /*320*/ _L,    _L,    _L,    _L,    _L,    _L,    _L,    _L,
35  /*330*/ _L,    _L,    _L,    _P,    _P,    _P,    _P,    _P,
36  /*340*/ 0,     0,     0,     0,     0,     0,     0,     0,
37  /*350*/ 0,     0,     0,     0,     0,     0,     0,     0,
38  /*360*/ 0,     0,     0,     0,     0,     0,     0,     0,
39  /*370*/ 0,     0,     0,     0,     0,     0,     0,     _C
40  };
```

7.5 DESCRIÇÃO DO PROGRAMA edit3

```
1   /*
2    * edit3 - um editor de tela capenga
3    *
4    * Sintaxe: edit3 [arquivo]
5    */
6
7   #include    <curses.h>
8   #include    <signal.h>
9   #include    <isctype.h>
10  #include    <string...h>
11
12
13   /* definicao de constantes e macros */
14
15   #define MAXCOLUNA   132 /* maior tela aceitavel */
16
17   /* posicao inicial do texto */
18   #define INI_Y   1   /* linha inicial do texto */
19   #define INI_X   0   /* coluna inicial do texto */
20
21   /* modos de operacao */
22   #define INSERCAO    0
23   #define ADICAO      1
24
25   #define CTRL(c)      ((c) - 'A' + '\001')
26   #define min(x, y)    ((x) < (y) ? (x) : (y))
27
28   /* comandos do editor */
29   #define ACAO         CTRL('A')
30   #define MUDAMODO     CTRL('V')
31   #define ESQUERDA     CTRL('S')
32   #define DIREITA      CTRL('D')
33   #define CIMA         CTRL('E')
34   #define BAIXO        CTRL('X')
35   #define PROXLINHA    ('\n')
36   #define ZERAATEFIM   CTRL('F')
37   #define ZERALINHA    CTRL('B')
38   #define REM_CARAC    CTRL('C')
39   #define REM_LINHA    CTRL('L')
40   #define INS_LINHA    CTRL('I')
41   #define ZERATUDO     CTRL('T')
42   #define ROLABAIXO    CTRL('W')
43   #define ROLACIMA     CTRL('Z')
44   #define REDESENHA    CTRL('R')
45   #define COPIABLOCO   CTRL('K')
```

```
46  #define ZERABLOCO   CTRL('T')
47  #define DELBLOCO    '\033'  /* ESC */
48  #define LEARQ       CTRL('Q')
49
50  /* acoes possiveis */
51  #define ACAO_SAI    'S'
52  #define ACAO_GRAVA  'G'
53  #define ACAO_SHELL  '!'
54
55  /* regioes da linha de status */
56  #define COL_ARQUIVO (COLS/2 - 7)    /* regiao "arquivo" */
57  #define COL_MODO    1        /* regiao "modo" */
58  #define COL_BLOCO   (COLS - 30) /* regiao "bloco" */
59
60  /* digitar s, S ou CONTROLE S deve ser a mesma coisa */
61  #define PADRONIZA(c)    (isalpha(c) ? toupper((c) | 0x40) : (c))
62
63  /* o que fazer apos uma acao */
64  #define CONTINUA    0   /* continua a editar */
65  #define FIM     1   /* termina o programa */
66
67
68  /*
69   * estrutura que controla o bloco
70   */
71
72  struct bloco {
73      bool    existe;     /* o bloco existe ? */
74      short   y_inf;      /* linha inicial do bloco */
75      short   x_inf;      /* coluna inicial do bloco */
76      short   y_sup;      /* linha final do bloco */
77      short   x_sup;      /* coluna final do bloco */
78      caractere   c_inf_inf;  /* o caracteres dos quatro cantos
79                   * do bloco (se precisarmos desenhar
80                   * a borda sem atributos de video)
81                   * ...
82                   */
83      caractere   c_inf_sup;
84      caractere   c_sup_inf;
85      caractere   c_sup_sup;  /* ... ate aqui */
86  } bloco = {
```

```
87      FALSE,      /* no inicio do programa, o bloco nao existe */
88      0, 0, 0, 0, /* devem ser zero para indicar que nunca
89                   * existiu bloco
90                   */
91      ' ', ' ', ' ', ' '  /* estes valores nao importam */
92  };
93
94
95  /*
96   * macros usadas pelas rotinas que nao devem
97   * conhecer a estrutura do bloco mas que
98   * precisam informacao sobre a localizacao
99   * do bloco
100  */
101
102 #define Y_INF   (bloco.y_inf)
103 #define X_INF   (bloco.x_inf)
104 #define Y_SUP   (bloco.y_sup)
105 #define X_SUP   (bloco.x_sup)
106
107
108 /*
109  * O desenho das bordas do bloco pode usar atributos
110  * de video se o terminal for "razoavel"
111  */
112 #define PODE_USAR_STANDOUT  (magic_cookie_glitch < 1 && \
113                 enter_standout_mode != NULL)
114
115
116 char    *mens();
117 JANELA  *janmsg; /*janela usada para imprimir mensagens e ler respostas*/
118 char    arq_usuario[100] = "";  /* nome do arquivo sendo editado */
119
120
121 /*
122  * Rotina principal
123  * Verifica a sintaxe do comando
124  * Inicializa o curses
125  * Le o arquivo do usuario
126  * Interpreta os comandos do usúario
127  * sai, fornecendo o status do comando
128  */
```

```
129
130   main( argc, argv )
131   int argc;
132   char    *argv[];
133   {
134
135       if( argc > 2 ) {
136           fprintf(stderr, "Sintaxe: %s [arquivo]\n", argv[0] );
137           exit( 1 );
138       }
139
140       if( argv[1] != NULL ) {
141           strcpy( arq_usuario, argv[1] );
142       }
143
144       if( inicializa() == ERR          /* nao inicializou */
145        || argv[1] && le_arq( argv[1], INI_Y, INI_X ) == ERR /* nao leu arq */
146         || comandos() == ERR ) {       /* erro nos comandos */
147
148           tchau( 1 ); /* nao retorna */
149       } else {
150           tchau( 0 ); /* nao retorna */
151       }
152   }
153
154
155   /*
156    * tchau
157    * sai, fornecendo o status do sistema
158    */
159
160   tchau( status )
161   int status: /* status do programa */
162   {
163
164       endwin();   /* encerra o curses*/
165       exit( status );
166   }
167
168
169   /*
```

```
170   * inicializa
171   * inicializa curses e o terminal
172   */
173
174  inicializa()
175  {
176
177      signal( SIGINT, SIG_IGN );
178      initscr();
179      noecho();
180      cbreak();
181      nl();
182      noxon();
183      setscrreg( INI_Y, LINES - 1 );
184      idlok( stdscr, TRUE );
185      keypad( stdscr, TRUE );
186      modoabc( TRUE );
187      acent( TRUE );
188
189      /* cria e inicializa a janela de mensagens */
190      if( (janmsg = newwin( 3, COLS, INI_Y, INI_X )) == NULL ) {
191          return( ERR );
192      }
193      idlok( janmsg, TRUE );
194      keypad( janmsg, TRUE );
195      wmodoabc( janmsg, TRUE );
196      wacent( janmsg, TRUE );
197      return( OK );
198  }
199
200
201  /*
202   * le_arq
203   * abre e le o arquivo do usuario
204   * colocando o texto na janela stdscr
205   */
206
207  le_arq( arquivo, y, x )
208  char    *arquivo;   /* nome do arquivo do usuario */
209  int y, x;       /* posicao onde colocar o texto do arquivo */
210  {
```

```
211
212         char        *fgets();
213         register int    lin;    /* para varrer as linhas */
214         FILE        *fp;    /* arquivo aberto para leitura */
215         int     tam;    /* tamanho da leitura */
216         char        buf[MAXCOLUNA + 2]; /* buffer de leitura */
217         int     ret;
218
219         ret = OK;
220         tam = min( MAXCOLUNA, COLS + 2 );   /* +2 por causa do \n
221                             * e do \0 finais
222                             */
223         if( (fp = fopen( arquivo, "r")) == (FILE *)NULL ) {
224             mens( "Nao pode abrir %s. Tecle <RETURN>. ", arquivo );
225         } else {
226             for( lin = y; lin < LINES; lin++ ) {
227                 if( fgets( buf, tam, fp ) == (char *)NULL ) {
228                     if( ! feof( fp ) ) {    /* erro de leitura */
229                         mens( "Erro de leitura em %s. Tecle <RETURN>. ",
230                             arquivo );
231                         ret = ERR;
232                     }
233                     break;
234                 }
235                 mvaddstr( lin, x, buf ); /* coloca o texto na janela */
236             }
237             fclose( fp );
238         }
239         move( y, x );
240         return( ret );
241     }
242
243
244     /*
245      * comandos
246      * inicializa o cursor, a linha de status e o modo de operacao
247      * interpreta os comandos do usuario
248      */
249
250     comandos()
251     {
```

```
252
253     int     modo;        /* modo de operacao */
254     int     pos_y, pos_x;  /* posicao atual do cursor */
255     caractere   cmd;        /* caractere digitado pelo usuario */
256     int     resultado;  /* acao deu erro ? */
257     char        *arq;       /* nome do arquivo a ler */
258
259     move( INI_Y, INI_X );
260     modo = INSERCAO;    /* modo inicial de operacao */
261     sta_arq( arq_usuario );
262     sta_modo( modo );
263
264     while( TRUE ) {
265         getyx( stdscr, pos_y, pos_x );
266         refresh();
267         switch( cmd = getch() ) {
268         case ACAO:
269             if( acao( &resultado ) == FIM ) {
270                 return( resultado );
271             }
272             break;
273         case MUDAMODO:
274             modo = modo == INSERCAO ? ADICAO : INSERCAO;
275             sta_modo( modo );   /* mostra novo modo */
276             break;
277         case ESQUERDA:
278         case KEY_LEFT:
279             movimenta( pos_y, pos_x - 1 );
280             break;
281         case DIREITA:
282         case KEY_RIGHT:
283             movimenta( pos_y, pos_x + 1 );
284             break;
285         case CIMA:
286         case KEY_UP:
287             movimenta( pos_y - 1, pos_x );
288             break;
289         case BAIXO:
290         case KEY_DOWN:
291             movimenta( pos_y + 1, pos_x );
292             break;
```

```
293         case PROXLINHA:
294             movimenta( pos_y + 1, 0 );
295             break;
296         case ZERAATEFIM:
297             clrtobot();
298             break;
299         case ZERALINHA:
300             clrtoeol();
301             break;
302         case REM_CARAC:
303             delch();
304             break;
305         case REM_LINHA:
306             deleteln();
307             break;
308         case INS_LINHA:
309             insertln();
310             break;
311         case ZERATUDO:
312             clear();
313             sta_arq( arq_usuario );
314             sta_modo( modo );
315             move( INI_Y, INI_X );
316             break;
317         case ROLABAIXO:
318             nscroll( -1 );
319             movimenta( pos_y, pos_x );
320             break;
321         case ROLACIMA:
322             nscroll( 1 );
323             movimenta( pos_y, pos_x );
324             break;
325         case REDESENHA:
326             clearok( stdscr, TRUE );
327             break;
328         case COPIABLOCO:
329             trata_bloco();
330             break;
331         case LEARQ:
332             arq = mens( "Nome do arquivo a ler: " );
333             le_arq( arq, pos_y, pos_x );
```

```
334                 break;
335             default:
336                 if( cmd == erasechar() ) {
337                     if( movimenta( pos_y, pos_x - 1 ) == OK ) {
338                         delch();
339                     }
340                 } else if( isprint( cmd ) ) {   /* da para imprimir ? */
341                     if( modo == ADICAO ) {
342                         addch( cmd );
343                     } else {
344                         insch( cmd );
345                         movimenta( pos_y, pos_x + 1 );
346                     }
347                 } else {
348                     beep();
349                 }
350                 break;
351         }
352     }
353 }
354
355
356 /*
357  * sta_arq
358  * exibe o nome do arquivo na linha de status
359  * (somente o ultimo componente, porque o nome completo
360  * pode ser grande)
361  */
362
363 sta_arq( arquivo )
364 char    *arquivo;   /* nome do arquivo do usuario */
365 {
366
367     char    *ult_nome();
368     int lemb_y, lemb_x; /* para voltar o cursor */
369
370     getyx( stdscr, lemb_y, lemb_x );
371     mvprintw( 0, COL_ARQUIVO, "%14s", ult_nome( arquivo ) );
372     move( lemb_y, lemb_x );
373 }
374
```

```
375
376  /*
377   * sta_modo
378   * exibe o modo de operacao na linha de status
379   */
380
381  sta_modo( modo )
382  int modo;        /* o modo de operacao */
383  {
384
385      int lemb_y, lemb_x;      /* para voltar o cursor */
386
387      getyx( stdscr, lemb_y, lemb_x );
388      standout();
389      mvprintw( 0, COL_MODO, "%10s", modo == INSERCAO ? "INSERCAO" :
390                                     " ADICAO" );
391      standend();
392      move( lemb_y, lemb_x );
393  }
394
395
396  /*
397   * movimenta
398   * movimenta o cursor da janela, verificando casos ilegais
399   */
400
401  movimenta( y, x )
402  int y;
403  int x;  /* posicao desejada */
404  {
405
406      if( y < INI_Y ||     /* tentando entrar na linha de status ? */
407          move( y, x ) == ERR ) { /* caindo fora da janela ? */
408          beep();
409          return( ERR );
410      }
411      return( OK );
412  }
413
414
415  /*
```

```
416    * acao
417    * pergunta ao usuario o que ele quer fazer
418    * (sair, gravar ou disparar um shell)
419    * e desempenha a acao
420    */
421
422   acao( ap_res )
423   int *ap_res;     /* onde colocar o resultado: OK ou ERR */
424   {
425
426       char    *resp;  /* resposta do usuario */
427       int ret;    /* valor de retorno da funco acao() */
428       int tam_vert;   /* tamanho vertical da janela a gravar */
429       int tam_hor;    /* tamanho horizontal da janela a gravar */
430       char    *arq;       /* nome do arquivo a gravar */
431       JANELA  *jan;       /* janela a gravar */
432       int ini_y, ini_x;   /* posicao inicial da janela a gravar */
433       int (*istat)(); /* acao para SIGINT deve ser guarada */
434                   /* antes de disparar um shell */
435
436       *ap_res = OK;   /* supoe que nao havera erro */
437       ret = CONTINUA;
438       resp = mens( "Sai(s), Grava(g) ou Shell(!) ? ");
439       switch( PADRONIZA( *resp ) ) {
440       case ACAO_SAI:
441           ret = FIM;
442           break;
443       case ACAO_GRAVA:
444           if( bloco.existe ) {    /* gravacao do bloco */
445               tam_vert = Y_SUP - Y_INF + 1;
446               tam_hor  = X_SUP - X_INF + 1;
447               ini_y = 0;
448               ini_x = 0;
449               if( (jan = subwin( stdscr, tam_vert, tam_hor,
450                           Y_INF, X_INF )) == NULL ) {
451                   mens( "Gravacao do bloco falhou. Tecle <RETURN> " );
452                   break;
453               }
454               arq = mens( "Nome do arquivo para gravacao do bloco: " );
455           } else {    /* gravacao de stdscr inteira */
456               tam_vert = LINES;
```

```
457             tam_hor = COLS;
458             ini_y = INI_Y;
459             ini_x = INI_X;
460             jan = stdscr;
461             if( *arq_usuario == '\0' ) {
462                 strcpy( arq_usuario,
463                     mens( "Nome do arquivo para gravacao: " ) );
464                 sta_arq( arq_usuario );
465             }
466             arq = arq_usuario;
467         }
468         if( (*ap_res = grava( arq, jan, tam_vert, tam_hor, ini_y, ini_x ))
469                     == ERR ) {
470             mens( "Erro de gravacao. Tecle <RETURN>. ");
471         }
472         if( bloco.existe ) {
473             delwin( jan );  /* deleta janela temporaria */
474         }
475         break;
476     case ACAO_SHELL:
477         /* dispara um shell */
478         apag_tela();
479         reset_shell_mode();
480         istat = signal( SIGINT, SIG_DFL );
481         system( "exec sh" );
482         signal( SIGINT, istat );
483         reset_prog_mode();
484         clearok( stdscr, TRUE );    /* forca redesenho da tela */
485         break;
486     default:
487         beep();     /* usuario se arrependeu ou errou */
488         break;
489     }
490
491     return( ret );
492 }
493
494
495 /*
496  * mens
497  * faz uma pergunta e retorna a resposta (uma cadeia)
```

```
498    */
499
500  char *
501  mens( msg, param )
502  char    *msg;
503  char    *param;
504  {
505
506      static char resposta[MAXCOLUNA];
507
508      mvwprintw( janmsg, 1, 1, msg, param );
509      wclrtoeol( janmsg );
510      box( janmsg, '|', '-' );
511      wrefresh( janmsg );
512      echo();
513      wgetstr( janmsg, resposta );
514      noecho();
515      touchwin( stdscr );
516      return( resposta );
517  }
518
519
520  /*
521   * grava
522   * grava o texto de uma janela no arquivo do usuario
523   */
524
525  grava( arquivo, janela, tam_vert, tam_hor, ini_y, ini_x )
526  char    *arquivo;   /* nome do arquivo do usuario */
527  JANELA  *janela;    /* janela a gravar */
528  int tam_vert, tam_hor;  /* tamanho da janela a gravar */
529  int ini_y, ini_x;   /* posicao inicial da janela a gravar */
530  {
531
532      int     ret;    /* valor de retorno da funcao grava() */
533      register int    lin;    /* para varrer as linhas da janela */
534      register int    col;    /* para varrer as colunas da janela */
535      int     col_final;  /* coluna final a gravar na linha */
536      FILE        *fp;    /* arquivo aberto para gravacao */
537      int     lemb_y, lemb_x; /* para voltar o cursor */
538
```

```
539      if( (fp = fopen( arquivo, "w")) == (FILE *)NULL ) {
540          mens( "Nao pode criar arquivo %s. Tecle algo. ", arquivo );
541          return( ERR );
542      }
543
544      getyx( janela, lemb_y, lemb_x );
545      ret = OK;
546      for( lin = ini_y; ret == OK && lin < tam_vert; lin++ ) {
547          /* procura a ultima coluna que interessa nesta linha */
548          for( col_final = tam_hor - 1;
549               col_final >= 0 && mvwinch( janela, lin, col_final ) == ' ';
550               col_final-- )
551              ;
552
553          for( col = 0; col <= col_final; col++ ) {
554              if( fputc( mvwinch( janela, lin, col ), fp ) == EOF ) {
555                  ret = ERR;
556                  break;
557              }
558          }
559
560          if( fputc( '\n', fp ) == EOF ) {
561              ret = ERR;
562          }
563      }
564
565      if( fclose( fp ) == EOF ) {
566          ret = ERR;
567      }
568      wmove( janela, lemb_y, lemb_x );
569      return( ret );
570  }
571
572
573  /*
574   * ult_nome
575   * retorna o ultimo componente de um nome de arquivo
576   */
577
578  char *
579  ult_nome( s )
```

```
580   register char   *s; /* nome original do arquivo */
581   {
582
583       char    *strrchr();
584       char    *p;
585
586       p = strrchr( s, '/' );
587       return( p == NULL ? s : p + 1 );
588   }
589
590
591   /*
592    * trata_bloco
593    * enquanto um bloco esta definido, o laco
594    * de comandos eh este, e nao o presente na rotina
595    * "comandos"
596    */
597
598   trata_bloco()
599   {
600
601       int pos_y, pos_x;   /* posicao do cursor antes
602                            * da digitacao de um comando
603                            */
604       int fim_y, fim_x;   /* localizacao o fim do bloco */
605       int resultado;  /* resultado da acao */
606       bool    em_bloco;   /* usado para sair do modo bloco */
607       caractere   cmd;        /* comando digitado pelo usuario */
608
609       getyx( stdscr, pos_y, pos_x );
610
611       /* definicao do bloco */
612       if( Y_INF == 0 ) {
613           /* nunca houve bloco antes */
614           fim_y = LINES - 1;
615           fim_x = COLS - 1;
616       } else {
617           /* tenta manter o tamanho do bloco anterior */
618           fim_y = min( pos_y + Y_SUP - Y_INF, LINES-1 );
619           fim_x = min( pos_x + X_SUP - X_INF, COLS-1 );
620       }
```

```
621        def_bloco( pos_y, pos_x, fim_y, fim_x );
622        em_bloco = TRUE;
623        while( em_bloco ) {
624            getyx( stdscr, pos_y, pos_x );
625            refresh();
626            switch( cmd = getch() ) {
627            case ESQUERDA:
628                movimenta( pos_y, pos_x - 1 );
629                break;
630            case DIREITA:
631                movimenta( pos_y, pos_x + 1 );
632                break;
633            case CIMA:
634                movimenta( pos_y - 1, pos_x );
635                break;
636            case BAIXO:
637                movimenta( pos_y + 1, pos_x );
638                break;
639            case KEY_LEFT:
640            case CTRL('H'): /* muitos terminais geram isso com <-- */
641                if( movimenta( pos_y, pos_x - 1 ) == OK ) {
642                    /* X_INF tem prioridade sobre X_SUP */
643                    if( pos_x == X_INF ) {
644                        def_bloco( Y_INF, pos_x - 1, Y_SUP, X_SUP );
645                        pos_x--;
646                    }
647                    if( pos_x == X_SUP ) {
648                        def_bloco( Y_INF, X_INF, Y_SUP, pos_x - 1 );
649                    }
650                }
651                break;
652            case KEY_RIGHT:
653                if( movimenta( pos_y, pos_x + 1 ) == OK ) {
654                    /* X_SUP tem prioridade sobre X_INF */
655                    if( pos_x == X_SUP ) {
656                        def_bloco( Y_INF, X_INF, Y_SUP, pos_x + 1 );
657                        pos_x++;
658                    }
659                    if( pos_x == X_INF ) {
660                        def_bloco( Y_INF, pos_x + 1, Y_SUP, X_SUP );
661                    }
```

```
662                 }
663                 break;
664             case KEY_UP:
665                 if( movimenta( pos_y - 1, pos_x ) == OK ) {
666                     /* Y_INF tem prioridade sobre Y_SUP */
667                     if( pos_y == Y_INF ) {
668                         def_bloco( pos_y - 1, X_INF, Y_SUP, X_SUP );
669                         pos_y--;
670                     }
671                     if( pos_y == Y_SUP ) {
672                         def_bloco( Y_INF, X_INF, pos_y - 1, X_SUP );
673                     }
674                 }
675                 break;
676             case KEY_DOWN:
677                 if( movimenta( pos_y + 1, pos_x ) == OK ) {
678                     /* Y_SUP tem prioridade sobre Y_INF */
679                     if( pos_y == Y_SUP ) {
680                         def_bloco( Y_INF, X_INF, pos_y + 1, X_SUP );
681                         pos_y++;
682                     }
683                     if( pos_y == Y_INF ) {
684                         def_bloco( pos_y + 1, X_INF, Y_SUP, X_SUP );
685                     }
686                 }
687                 break;
688             case ZERABLOCO:
689                 branq_bloco();
690                 del_bloco();
691                 em_bloco = FALSE;
692                 break;
693             case DELBLOCO:
694                 del_bloco();
695                 em_bloco = FALSE;
696                 break;
697             case ACAO:
698                 if( acao( &resultado ) == FIM ) {
699                     del_bloco();
700                     em_bloco = FALSE;
701                 }
702                 break;
```

```
703             case COPIABLOCO:
704                 copia_bloco();
705                 break;
706             case REDESENHA:
707                 clearok( stdscr, TRUE );
708                 break;
709             default:
710                 beep();
711                 break;
712         }
713     }
714 }
715
716
717 /*
718  * def_bloco
719  * define um bloco delineado pelos parametros recebidos
720  *
721  * preenche a estrutura "bloco" com a informacao sobre
722  * a localizacao do bloco, desenha a borda para identificar
723  * o bloco e marca o bloco como existente
724  */
725
726 def_bloco( y_inf, x_inf, y_sup, x_sup )
727 register int    y_inf, x_inf, y_sup, x_sup; /* localizacao */
728 {
729
730     if( bloco.existe ) {
731         del_bloco();    /* deleta o bloco existente para
732                          * definir outro
733                          */
734     }
735
736     Y_INF = y_inf;
737     X_INF = x_inf;
738     Y_SUP = y_sup;
739     X_SUP = x_sup;
740
741     desenha_borda();
742
743     bloco.existe = TRUE;
```

```
763         standend();
764         move( lemb_y, lemb_x );
765     }
766
767     /*
768      * desenha_borda
769      *
770      * representacao visual do bloco na janela
771      * Se for possivel usar atributos de video, usa-os
772      * senao, coloca um asterisco nos quatro cantos
773      * do bloco, salvando os caracteres ai presentes para
774      * restauracao posterior
775      */
776
777     desenha_borda()
778     {
779
780         int         lemb_y, lemb_x; /* para voltar o cursor */
781         register int    y, x;       /* para varrer as bordas */
744         sta_bloco( TRUE );
745     }
746
747     /*
748      * sta_bloco
749      * exibe se estamos no modo bloco ou nao na linha de status
750      */
751
752     sta_bloco( modo )
753     int modo;   /* TRUE: modo bloco, FALSE: modo normal */
754     {
755
756         int lemb_y, lemb_x;     /* para voltar o cursor */
757
758         getyx( stdscr, lemb_y, lemb_x );
759         if( modo ) {
760             standout();
761         }
762         mvprintw( 0, COL_BLOCO, "%10s", modo ? "BLOCO" : "" );
782         caractere   c;      /* para pegar o que esta na janela */
783
784         getyx( stdscr, lemb_y, lemb_x );
```

```
785
786         if( PODE_USAR_STANDOUT ) {
787             for( x = X_INF; x <= X_SUP; x++ ) {
788                 c = mvinch( Y_INF, x );
789                 addch( c | A_STANDOUT );     /* linha inicial */
790                 c = mvinch( Y_SUP, x );
791                 addch( c | A_STANDOUT );     /* linha final */
792             }
793             for( y = Y_INF; y <= Y_SUP; y++ ) {
794                 c = mvinch( y, X_INF );
795                 addch( c | A_STANDOUT );     /* coluna inicial */
796                 c = mvinch( y, X_SUP );
797                 addch( c | A_STANDOUT );     /* coluna final */
798             }
799         } else {     /* tem que colocar '*' nos quatro cantos */
800             bloco.c_inf_inf = mvinch( Y_INF, X_INF );
801             addch( '*' );
802             bloco.c_inf_sup = mvinch( Y_INF, X_SUP );
803             addch( '*' );
804             bloco.c_sup_inf = mvinch( Y_SUP, X_INF );
805             addch( '*' );
806             bloco.c_sup_sup = mvinch( Y_SUP, X_SUP );
807             addch( '*' );
808         }
809         move( lemb_y, lemb_x );
810     }
811
812
813     /*
814      * del_bloco
815      * deleta o bloco e remove as bordas
816      */
817
818     del_bloco()
819     {
820
821         elim_borda();
822         bloco.existe = FALSE;
823         sta_bloco( FALSE );
824     }
825
```

```
826
827  /*
828   * elim_borda
829   *
830   * elimina as bordas, removendo os atributos de video
831   * ou eliminando os '*' dos quatro cantos
832   */
833
834  elim_borda()
835  {
836
837      int      lemb_y, lemb_x; /* para voltar o cursor */
838      register int    y, x;        /* para varrer as bordas */
839      caractere   c;      /* para pegar o que esta na janela */
840
841      getyx( stdscr, lemb_y, lemb_x );
842
843      if( PODE_USAR_STANDOUT ) {
844          for( x = X_INF; x <= X_SUP; x++ ) {
845              c = mvinch( Y_INF, x );
846              addch( c & A_TEXTO );   /* linha inicial */
847              c = mvinch( Y_SUP, x );
848              addch( c & A_TEXTO );   /* linha final */
849          }
850          for( y = Y_INF; y <= Y_SUP; y++ ) {
851              c = mvinch( y, X_INF );
852              addch( c & A_TEXTO );   /* coluna inicial */
853              c = mvinch( y, X_SUP );
854              addch( c & A_TEXTO );   /* coluna final */
855          }
856      } else {    /* tem que remover os '*' nos quatro cantos */
857          mvaddch( Y_INF, X_INF, bloco.c_inf_inf );
858          mvaddch( Y_INF, X_SUP, bloco.c_inf_sup );
859          mvaddch( Y_SUP, X_INF, bloco.c_sup_inf );
860          mvaddch( Y_SUP, X_SUP, bloco.c_sup_sup );
861      }
862      move( lemb_y, lemb_x );
863  }
864
865
866  /*
```

```
867    * branq_bloco
868    *
869    * preenche a area do bloco com brancos
870    */
871   branq_bloco()
872   {
873
874       int      lemb_y, lemb_x;
875       register int    y, x;
876
877       getyx( stdscr, lemb_y, lemb_x );
878       for( y = Y_INF; y <= Y_SUP; y++ ) {
879           for( x = X_INF; x <= X_SUP; x++ ) {
880               mvaddch( y, x, ' ' );
881           }
882       }
883       move( lemb_y, lemb_x );
884   }
885
886   /*
887    * copia_bloco
888    *
889    * copia o bloco marcado na posicao do cursor
890    * Isto eh feito criando uma subjanela do tamanho do bloco
891    * e copiando-a para uma nova janela do mesmo tamanho
892    * com "overwrite". A mesma coisa eh feita para copiar
893    * a nova janela no lugar de destino
894    */
895
896   copia_bloco()
897   {
898
899       int      pos_y, pos_x;   /* onde queremos copiar */
900       int      tam_vert;    /* tamanho vertical do bloco */
901       int      tam_hor;     /* tamanho horizontal do bloco */
902       JANELA      *sjanorig;  /* subjanela de origem */
903       JANELA      *sjandest;  /* subjanela de destino */
904       JANELA      *guardajan; /* janela temporaria para
905                        * copiar o bloco
906                        */
907
```

```
908
909       getyx( stdscr, pos_y, pos_x );
910       tam_vert = Y_SUP - Y_INF + 1;
911       tam_hor  = X_SUP - X_INF + 1;
912       sjanorig = sjandest = guardajan = NULL;
913
914       /* elimina a borda do bloco na janela stdscr
915        * para nao copiar atributos de video ou os '*'
916        */
917       elim_borda();
918
919       if( (sjanorig = subwin( stdscr, tam_vert, tam_hor,
920                     Y_INF, X_INF )) == NULL
921        || (sjandest = subwin( stdscr, tam_vert, tam_hor,
922                     pos_y, pos_x )) == NULL
923        || (guardajan = newwin( tam_vert, tam_hor, 0, 0 )) == NULL
924        || overwrite( sjanorig, guardajan ) == ERR
925        || overwrite( guardajan, sjandest ) == ERR ) {
926
927           beep();
928           mens( "Copia falhou. Tecle <RETURN>. " );
929       } else {     /* copia deu certo */
930           wnoutrefresh( sjandest );
931       }
932
933       /* elimine as janelas temporarias */
934       if( sjanorig != NULL ) {
935           delwin( sjanorig );
936       }
937       if( sjandest != NULL ) {
938           delwin( sjandest );
939       }
940       if( guardajan != NULL ) {
941           delwin( guardajan );
942       }
943
944
945       desenha_borda();
946   }
947
948
```

```
949  /*
950   * apag_tela
951   * Apaga a tela do terminal sem usar CURSES
952   */
953
954  apag_tela()
955  {
956
957       int _sai_carac(); /* rotina interna do CURSES */
958
959       tputs( clear_screen, 1, _sai_carac );
960       fflush( stdout );
961  }
```

APÊNDICE A

DIFERENÇAS DO PADRÃO ANS PARA O K&R

Aqui estão resumidas as principais diferenças existentes entre o padrão ANS para a linguagem C e a definição original documentada por Kernighan e Ritchie no livro "The C Programming Language" (que doravante passa a ser referenciado apenas como K&R).

Estas diferenças, em sua maioria, visam garantir uma maior confiabilidade e portabilidade dos programas desenvolvidos em C. Dentre as diferenças, pode-se citar uma comparação mais rigorosa na verificação de tipos incompatíveis, a possibilidade de se declarar protótipos de funções que garantem a verificação do número e tipo dos parâmetros que são passados para as funções, e outras extensões mais.

O objetivo deste anexo é fornecer informações que facilitem o transporte para o padrão ANS de programas escritos segundo K&R, ou escritos para o compilador C sob o sistema operacional SOD. Este último segue basicamente o descrito por K&R, com exceção de algumas diferenças, mencionadas neste anexo.

As principais implementações do padrão ANS são descritas a seguir.

. Identificadores

  O número de caracteres significativos para diferenciar nome de identificadores (incluindo os utilizados em diretivas do pré-processador) passa a ser de 31 caracteres.

. Palavras-Chave

Não existem as palavras-chave asm, entry e fortran mencionadas originalmente por K&R. Em contrapartida, foram criadas as palavras-chave const, enum, signed, void e volatile.

Os vocábulos enum e void são também palavras chave no compilador C para o sistema operacional SOD (versões com e sem PPF).

. Constantes Inteiras

Foi introduzido o sufixo U (ou u), para indicar constantes do tipo unsigned. Se usado em conjunto com L (ou l) formando UL (ou ul), indica constante do tipo unsigned long.

. Constantes Caractere

São permitidas constantes de dois caracteres, como em 'AB' , '\n\n' ou '\n\012'.

Pode-se também especificar uma constante caractere com valor em hexadecimal, precedendo-se a seqüência pelos caracteres \x.

Exemplo:

```
'\x0A'
'\x15'
```

Foi introduzida a seqüência padronizada '\a', que indica "alerta" (**audible bell** ou "BEL").

. Constantes Reais

Foram introduzidos os sufixos f (ou F) e l (ou L), que fazem a conversão das constantes reais para os tipos float e long double, respectivamente.

. Cadeias de Caracteres

Cadeias de caracteres consecutivas são concatenadas, resultando em uma única cadeia sem limite de tamanho.

Exemplo:

```
cadeia = "exemplo da "
         "concatenacao d" "e cadeias";
```

que equivale a:

```
cadeia = "exemplo da concatenacao de cadeias";
```

Observação:

A concatenação de cadeias de caracteres consecutivas é permitida no compilador C para o sistema operacional SOD, apenas na versão com PPF.

. Conversões

Algumas mudanças significativas na conversão de tipos dentro de expressões foram introduzidas. Convém uma prévia consulta às regras que se encontram explicitamente documentadas neste manual, no Capítulo 4, item 4.11.3, Regras de Conversões Aritméticas.

. Operadores

Foi criado o operador unário mais (+), que deve ser utilizado para garantir a ordem de avaliação de uma subexpressão dentro de uma expressão comutativa, visto que esta pode ser reorganizada, visando otimizar a avaliação.

Exemplo:

```
a = b + +(c + d);
```

ou

```
a = b * +(c * d);
```

garantindo que a subexpressão envolvendo os operandos c e d seja avaliada primeiramente.

. Operadores de Atribuição e de Comparação

Estruturas (ou Uniões) podem ser atribuídas, desde que sejam do mesmo tipo. A comparação entre estruturas, no entanto, não é permitida.

Na atribuição entre ponteiros, eles devem apontar para objetos de mesmo tipo, um deles deve ser do tipo "ponteiro para void", ou o operando da direita deve ser o ponteiro nulo (NULL).

Na comparação entre ponteiros, eles devem apontar para objetos de mesmo tipo; um deles deve ser do tipo "ponteiro para void", ou o ponteiro nulo (NULL).

As atribuições e comparações entre ponteiros e inteiros não são permitidas. Assim, tais operações devem estar eventualmente acompanhadas por um operador de conversão automática, a fim de que esta mistura de tipos seja válida.

Exemplo:

```
char *pc;
int *pi;

pc = pi;      /* INVALIDO */
```

O certo seria:

```
pc = (char *) pi;
```

. Especificadores de Tipo

Os seguintes especificadores de tipo foram introduzidos:

signed char	:	Caractere com sinal;
unsigned short	:	Inteiro curto sem sinal;
unsigned long	:	Inteiro longo sem sinal;
long double	:	Real longo (substitui long float do K&R);
enum	:	Tipo enumerado;
void	:	Tipo "vazio".

Observação:

Os tipos signed char e unsigned long estão implementados no compilador C para o sistema operacional SOD, apenas na versão com PPF.

Os tipos enum e void também estão implementados no compilador C para o sistema operacional SOD, nas versões com e sem PPF.

Outras extensões são os modificadores de tipo:

. signed : Equivale ao oposto de unsigned. Indica representação de valores "com sinal";

. const : Indica valor constante, ou seja, um objeto qualificado com este tipo não pode ter seu valor alterado por um comando de atribuição ou por efeitos colaterais;

. volatile : Indica valor volátil, isto é, representa objetos que podem ter seus valores alterados por condições externas ao programa. Esta forma previne o compilador de que não devem ser feitas referências supostas deste valor para otimizações ou alocações em registradores de máquina.

Foi acrescentado também o tipo ponteiro para void (isto é, ponteiro genérico).

. Funções

Na declaração de funções foi introduzido o conceito opcional de protótipo de função, que permite incluir uma lista de tipo dos parâmetros no cabeçalho da declaração e que ativa um mecanismo de verificação do número e tipo dos parâmetros nas chamadas posteriores da função prototipada.

Exemplo:

```
int f(long par1, int par2)
{
  .
    .
      .
}
void teste()
{
   char *pc;

   f(1,2);   /* A constante 1 e' convertida
                para long */
   f(1,2,3); /* ERRO: numero de parametros
                      invalido */
   f(pc, 2); /* ERRO: tipo do parametro
                      invalido */

}
```

A compatibilidade entre o tipo do parâmetro declarado e o valor passado na chamada a função obedece às mesmas regras da atribuição.

Todas as funções da biblioteca C possuem um cabeçalho associado, do qual consta a declaração do protótipo da função. E importante, sempre que se usar uma função da biblioteca, incluir o arquivo com os cabeçalhos associados, para que o uso correto das funções seja verificado pelo compilador.

O uso de protótipos de funções permite a detecção, em tempo de compilação, de erros freqüentes. A portabilidade é também aumentada, pois erros em passagem de parâmetros podem ocorrer em uma máquina, e não ocorrer em outras.

Se pfn é do tipo "ponteiro para função retornando um determinado tipo", a expressão pfn() significa o mesmo que (*pfn) ().

O operador & (endereço) pode ser aplicado a uma função.

Não é permitida a redeclaração como auto de um parâmetro formal de uma função.

Observação:

As declarações ASM e LPS aceitas no compilador C para o SOD não são aceitas pelo padrão ANS.

. Estruturas e Uniões

Estruturas e Uniões podem ser atribuídas, passadas como parâmetro e retornadas por funções, desde que sejam do mesmo tipo.

Campos de bits podem ser declarados como inteiros do tipo signed ou unsigned.

O acesso a membros de estruturas/uniões é mais rígido. Um membro de uma estrutura não pode ser acessado através de uma estrutura de tipo diferente.

Exemplo:

```
struct a { int ma1, ma2; } *pa;
struct b { int mb1, mb2; } *pb;

pb → ma1;      /* INVALIDO */
```

O certo seria:

```
((struct a *) pb) → mal;
```

Este mecanismo, além de mais seguro, permite que membros de estruturas diferentes tenham o mesmo nome.

Exemplo:

```
struct a { int ml; } *pa;
struct b { int outro; long ml; } *pb;
```

. Inicializações

Uniões podem ser inicializadas por valores do tipo de seu primeiro membro.

Tipos derivados (vetores, estruturas e uniões) com classe de armazenamento automática podem ser inicializados por expressões constantes. Neste caso, têm seus valores de inicialização restaurados a cada entrada na função.

Exemplo:

```
f()
{
    auto int vet[5] = {1, 2, 3, 4, 5};
}
```

O conteúdo de vet é inicializado a cada ativação da função f.

. Comando Switch

A expressão de seleção de um comando switch pode ter tipo long (signed ou unsigned). Neste caso, o valor das expressões constantes nas cláusulas case é convertido para este tipo.

. Diretivas do Pré-Processador

O símbolo #, que identifica uma diretiva do pré-processador, pode vir precedido ou seguido de caracteres brancos.

. Macrossubstituição

O nome de uma macro aparecendo no texto de substituição não é reexpandido.

Nomes de macros podem ser redefinidos, desde que tenham o mesmo texto.

Os seguintes identificadores são macros pré-definidas:

- __STDC__ : Contém 1, se compilação compatível com padrão ANS;
- __FILE__ : Contém o nome do arquivo que está sendo compilado;
- __LINE__ : Contém o número da linha que está sendo complilada;
- __DATE__ : Contém a data de compilação do programa;
- __TIME__ : Contém a hora de compilação de programa.

. Operador #

Foi introduzido o operador # do pré-processador. este operador, que deve aparecer antes de um parâmetro no texto de uma macro, faz o parâmetro ser transformado em uma cadeia.

Exemplo:

```
#define imprime(var)  printf(var " = %d", var);

imprime(x);
```

que é substituído por:

```
printf("X" "= %d", x);
```

.

. Operador ##

Foi introduzido o operador ## do pré-processador. Este operador concatena o texto anterior e o seguinte em um único símbolo.

Exemplo:

```
#define imp(n)  printf("%d %d" ,x##n ,y##n);

        imp(1);
```

que é substituído por:

```
printf("%d %%%%%d",x1 ,y1);
```

. Compilação Condicional

Introduziu-se o operador defined(<identificador>) que tem valor 1 quando o <identificador> está previamente definido, ou 0, quando não. Assim:

#if defined(id) é equivalente a ##ifdef id

e

#if !defined(id) é equivalente a #ifndef id

. Novas Diretivas do Pré-Processador

#include: identificador;

#elif : equivalente a um #else-#if;

#error : fornece uma mensagem de erro na compilação;

#pragma : utilizada para diretivas específicas de sistemas. No SOX, atualmente, nenhuma #pragma é reconhecida;

\# : diretiva nula - serve apenas como espacejamento.

Observação:

A diretiva #module existente no compilador C para o sistema operacional SOD não é aceita pelo padrão.

Fornecemos a seguir um breve resumo das rotinas e outras variáveis e definições do CURSES/TERMINFO. Observe que a discussão completa encontra-se nos Módulos 2 e 3. Para cada item abaixo, fornecemos os capítulos deste módulo que discutem o item, e as linhas dos programas edit1, edit2 e edit3, presentes nos itens 7.3, 7.4 e 7.5, respectivamente.

A_ACENTO

Máscara para saber se um caractere está acentuado ou não. (Ver também A_ATRIBUTOS, A_TEXTO e A_SEM_ACENTO).

A_ATRIBUTOS

Máscara para obter os atributos de um caractere. As possibilidades são:

A_STANDOUT	Atributo padrão de destaque
A_UNDERLINE	Sublinhado
A_REVERSE	Reverso
A_BLINK	Piscante
A_DIM	Atenuado
A_BOLD	Intenso
A_NORMAL	Normal

Ver também A_TEXTO, A_ACENTO, A_SEM_ACENTO

A_ATTRIBUTES

Equivalente a A_ATRIBUTO.

void
acent(flag)
Equivalente a wacent(stdscr, flag).

Referência: 3.2.2
edit2: 128
edit3: 187

A_CHARTEXT
Equivalente a A_TEXTO.

int
addch(c)
Equivalente a waddch(stdscr, c).

Referência: 2.2, 2.4, 3.2.1, 3.2.2, 4.3.1, 4.3.3, 4.3.4, 5.2
edit1: 215
edit2: 269
edit3: 342, 789, 791, 795 797, 801, 803, 805, 807, 846, 848, 852, 854, 857, 858, 859, 860, 880

int
addstr(str)
Equivalente a waddstr(stdscr, str).

Referência: 2.2, 2.3, 2.4, 4.2.2, 4.3.4
edit1: 156, 334
edit2: 170, 390
edit3: 235

A_SEM_ACENTO
Máscara usada para obter um caractere sem acento (com o oitavo bit desligado).

A_TEXTO
Máscara usada para obter o texto de um caractere sem atributos. (Ver também A_ATRIBUTOS, A_ACENTO, A_SEM_ACENTO.)

Referência: 4.3.3, 4.3.3
edit3: 846, 848, 852, 854

void
attroff(atributos)
Equivalente a wattroff(stdscr, atributos).

Referência: 5.2

void
attron(atributos)
Equivalente a wattron(stdscr, atributos).

Referência: 5.2

void
attrset(atributos)
Equivalente a wattrset(stdscr, atributos).

int
baudrate()
Retorna a velocidade de saída do terminal em bits por segundo.

void
beep()
Aciona a campainha do terminal.

Referência: 2.4, 3.2.1, 4.2.4, 4.3.4
edit1: 221, 279, 313
edit2: 275, 335, 369
edit3: 348, 408, 487, 927

bool
Tipo booleano; equivalente a char.

Referência: 4.3.2, 4.3.4
edit3: 73, 606

void
box(jan, vert, hor)
Desenha uma borda na janela jan, usando o caractere vert nas bordas verticais, e hor nas bordas horizontais.

Referência: 4.2.1
edit3: 510

caractere

Tipo (em C) usado para representar e armazenar caracteres no CURSES. (Ver também chtype.)

Referência: 2.4, 4.3.2, 4.3.3, 4.3.4
edit1: 176, 297, 330
edit2: 190, 353, 386
edit3: 78, 83, 84, 85, 255, 607, 782, 839

void
cbreak()

Coloca o terminal no modo CBREAK, no qual caracteres digitados são entregues imediatamente ao programa. Caracteres que geram sinais e controlam o fluxo são interpretados normalmente.

Referência: 2.3
edit1: 112
edit2: 121
edit3: 180

chtype

Equivalente a "caractere".

void
clear()

Equivalente a wclear(stdscr).

Referência: 3.2.1
edit2: 246
edit3: 312

void
clearok(jan, flag)
Se indicador for TRUE, o próximo wrefresh dado na janela jan forçará a limpeza e redesenho total da tela.

Referência: 3.2.1, 4.2.4, 4.3.4
edit2: 260
edit3: 326, 484, 707

void
clrtobot()
Equivalente a wclrtobot(stdscr).

Referência: 3.2.1
edit2: 231
edit3: 297

void
clrtoeol()
Equivalente a wclrtoeol(stdscr).

Referência: 2.4, 3.2.1
edit1: 338
edit2: 234, 394
edit3: 300

COLUNAS
Número de colunas do terminal (ver COLS, LINES e LINHAS).

COLS
Equivalente a COLUNAS (vér LINES e LINHAS).

Referência: 2.3, 2.4, 4.2.1, 4.2.2, 4.3.3, 4.3.4
edit1: 42, 43, 137, 369
edit2: 51, 52, 151, 425
edit3:

JANELA *
curscr
Tela virtual usada para atualizar a tela do terminal com doupdate.

Referência: 4.3.4

void
def_prog_mode()
O modo corrente do terminal é salvo como "modo programa". Volta-se a este modo com a rotina reset_prog_mode.

void
def_shell_mode()
O modo corrente do terminal é salvo como "modo shell". Volta-se a este modo com a rotina reset_shell_mode.

void
delay_output(miliseg)
Uma pausa de miliseg milissegundos é feita na saída para o terminal.

void
delch()
Equivalente a wdelch(stdscr).

Referência: 3.2.1
edit2: 237, 265
edit3: 303, 338

void
deleteln()
Equivalente a wdeleteln(stdscr).

Referência: 3.2.1
edit2: 240
edit3: 306

void
delwin(jan)
A janela jan é destruída.

Referência: 4.3.4
edit3: 473, 935, 938, 941

int
desacent(c)
O equivalente do caractere c sem acento é retornado.

Referência: 3.2.2

void
doupdate()
A tela é redesenhada, de forma a ficar idêntica à tela virtual curscr.

Referência: 4.3.4, 5.1

void
echo()
Caracteres obtidos com wgetch são ecoados imediatamente na tela.

Referência: 4.2.1
edit3: 512

void
endwin()
Usada para sair do CURSES. O terminal volta ao modo shell.

Referência: 2.3
edit1: 89
edit2: 98
edit3: 164

void
erase()
Equivalente a werase(stdscr).

int
erasechar()

O caractere de apagamento de caractere do terminal é retornado.

Referência: 3.2.1
edit2: 263
edit3: 336

ERR

Valor retornado pela maioria da rotinas, em caso de erro (ver OK e ERRO).

Referência: 2.3, 2.4, 4.2.1, 4.2.2, 4.2.4, 4.3.4
edit1: 68, 70, 71, 107, 143, 153, 278, 280, 294, 308, 362, 376, 382, 387
edit2: 77, 79, 80, 116, 157, 167, 334, 336, 350, 364, 418, 432, 438, 443
edit3: 144, 145, 146, 191, 231, 407, 409, 423, 469, 541, 555, 561, 566, 924, 925

ERRO

Equivalente a ERR (ver ERR e OK).

FALSE

Valor igual a 0.

Referência: 3.2.1, 4.3.2, 4.3.3, 4.3.4, 5.1, 6
edit3: 78, 691, 695, 700, 753, 822, 823

void
fixterm()

Ver reset_prog_mode.

void
flash()
Faz a tela piscar, para atrair a atenção do usuário. Se não for possível, flash faz soar a campainha do terminal.

void
flushinp()
Descarta quaisquer caracteres digitados, mas não lidos pelo programa.

int
getch()
Equivalente a wgetch(stdscr).

Referência: 2.3, 2.4, 3.2.1, 3.2.2, 4.2.3, 4.3.4, 5.1
edit1: 187, 336
edit2: 201, 392
edit3: 267, 626

void
getstr(str)
Equivalente a wgetstr(stdscr, str).

void
gettmode()
Não tem efeito.

void
getyx(jan, y, x)

Armazena a posição do cursor da janela jan em y e x.

Referência: 2.3, 2.4, 3.2.3, 4.2.2, 4.2.3, 4.3.3, 4.3.4
edit1: 185, 243, 260, 333, 365
edit2: 199, 297, 314, 389, 421
edit3: 265, 370, 387, 544, 609, 624, 758, 784, 841, 877, 909

int
has_ic()

Retorna TRUE, se o terminal tem inserção e deleção de caracteres por hardware.

int
has_il()

Retorna TRUE, se o terminal tem inserção e deleção de linhas por hardware.

void
idlok(jan, flag)

Se indicador for TRUE, CURSES tenta usar inserção e deleção de linhas por hardware, na atualização da janela jan.

Referência: 3.2.1, 4.2.1
edit2: 125
edit3: 184, 193

int
inch()
Equivalente a winch(stdscr).

Referência: 2.2, 2.4, 4.3.1, 4.3.3, 4.3.4
edit1: 370, 375
edit2: 426, 431
edit3: 788, 790, 794, 796, 800, 802, 804, 806, 845, 847, 851, 853

JANELA *
initscr()
Inicializa CURSES e retorna stdscr.

Referência: 2.3, 6
edit1: 110
edit2: 119
edit3: 178

int
insch(c)
Equivalente a winsch(jan, c).

Referência: 2.2, 2.4, 3.2.1, 3.2.2
edit1: 217
edit2: 271
edit3: 344

void
insertln()
Equivalente a winsertln(stdscr).

Referência: 3.2.1
edit2: 243
edit3: 309

void
intrflush(jan, flag)
Se indicador for TRUE, os caracteres enfileirados para saída para o terminal são descartados na ocorrência de uma interrupção. O argumento jan é ignorado.

JANELA
Tipo (em C) de uma janela. Identificadores de janelas são do tipo JANELA *. (Ver WINDOW.)

Referência: 4.2.1, 4.2.2, 4.2.4, 4.3.4
edit3: 117, 431, 527, 902, 903, 904

void
keypad(jan, flag)
Se indicador for TRUE, o usuário pode usar teclas que geram seqüências, e wgetch(jan) retornará um código único para representar tais teclas.

Referência: 3.2.1, 4.2.1
edit2: 126
edit3: 185, 194

KEY_*

Teclas que não possuem código reservado no conjunto ABICOMP (ou ASCII) são representadas por números únicos. Os valores possíveis são mostrados abaixo:

KEY_BREAK	Tecla break
KEY_DOWN	Seta para baixo
KEY_UP	Seta para cima
KEY_LEFT	Seta para a esquerda
KEY_RIGHT	Seta para a direita
KEY_HOME	Tecla HOME (seta para cima e esquerda)
KEY_BACKSPACE	Retrocesso
KEY_F0	Teclas de função (até 64 teclas)
KEY_F(1)	
...	
KEY_F(63)	
KEY_DL	Deleta linha
KEY_IL	Insere linha
KEY_DC	Deleta caractere
KEY_IC	Insere car. ou entra em modo de inserção
KEY_EIC	Sai do modo de inserção de caractere
KEY_CLEAR	Limpa tela
KEY_EOS	Limpa até fim da tela
KEY_EOL	Limpa até fim da linha
KEY_SF	Rola uma linha para frente
KEY_SR	Rola uma linha para trás
KEY_NPAGE	Próxima página
KEY_PPAGE	Página anterior
KEY_STAB	Seta tab
KEY_CTAB	Zera tab
KEY_CATAB	Zera todos os tabs
KEY_ENTER	Enter ou Send
KEY_SRESET	Reset parcial
KEY_RESET	Reset total
KEY_PRINT	Imprime ou copia

KEY_LL	Home em baixo (esq. em baixo)
KEY_A1	A1 A2 A3
KEY_A3	B1 B2 B3
KEY_B2	C1 C2 C3
KEY_C1	Disposição da ilha
KEY_C3	

Referência:3.2.1, 4.3.4
edit2: 212, 216, 220, 224
edit3: 278, 282, 286, 290, 639, 652, 664, 676

int
killchar()

O caractere de apagamento de linha do terminal é retornado.

void
leaveok(jan, flag)

Se indicador for TRUE, o cursor é deixado em qualquer posição conveniente (para o CURSES) após um wrefresh(jan).

LINES

Equivalente a LINHAS (ver COLS e COLUNAS).

Referência: 2.3, 2.4, 3.2.1, 4.2.1, 4.2.2, 4.3.4
edit1: 146, 367
edit2: 124, 160, 423
edit3: 183, 226, 456, 614, 618

LINHAS

Número de linhas no terminal (ver LINES, COLS e COLUNAS).

char *
longname()
Retorna o nome do terminal corrente por extenso.

void
modoabc(flag)
Equivalente a wmodoabc(stdscr, flag).

Referência: 3.2.2, 4.2.1
edit2: 127
edit3: 186

int
move(y, x)
Equivalente a wmove(stdscr, y, x).

Referência: 2.2, 2.3, 2.4, 3.2.1, 3.2.3, 4.2.2, 4.3.3, 4.3.4
edit1: 179, 245, 263, 276, 337, 339, 389
edit2: 193, 249, 299, 319, 334, 393, 395, 445
edit3: 239, 259, 315, 372, 407, 764, 809, 862, 883

void
mvcur(yo, xo, yd, xd)
Move o cursor da tela para (yd, xd), supondo que o cursor atual esteja na posição (yo, xo).

Referência: 6

void
mvprintw(y, x, formato [, arg] ...)
Move o cursor da janela stdscr para (y, x) e imprime formato de acordo com os argumentos arg.

Referência: 2.3, 2.4, 3.2.3, 4.3.3
edit1: 244, 261
edit2: 298, 316
edit3: 371, 389, 762

int
mvscanw(y, x, formato [, arg] ...)
Move o cursor da janela stdscr para (y, x) e lê a entrada de acordo com formato, deixando os itens lidos em arg.

mvxxxxxx(y, x, jan, ...)
Para a maioria das rotinas do CURSES, equivale a:
wmove(jan, y, x) = ERR ? ERR : xxxxxx(jan, ...)

void
mvwin(jan, y, x)
Move a janela jan para a posição (y, x) na tela.

Referência: 5.3

void
mvwprintw(y, x, jan, formato [, arg] ...)
Como mvprintw, mas para a janela jan.

Referência: 4.2.1
edit3: 508

int
mvwscanw(*y*, *x*, *jan*, *formato* [, *arg*] ...)
Como mvscanw, mas para a janela jan.

JANELA *
newpad(numlinhas, numcolunas)
Cria uma janela dinâmica de numlinhas linhas por numcolunas colunas.

Referência: 5.3

TELA *
newterm(nome, fd_entr, fd_sai)
Deve ser chamado para terminais adicionais manipulados pelo programa. Nome é o nome do terminal; fd_entr e fd_sai são os descritores de acesso ao terminal. (Ver set_term.)

JANELA *
newwin(numlinhas, numcolunas, *y*, *x*)
Uma janela de tamanho numlinhas por numcolunas é criada na posição (y, x) da tela.

Referência: 4.2.1, 4.3.4, 5.3
edit3: 190, 923

void
nl()

Força o mapeamento de CR e NL para a seqüência CR NL na saída. Na entrada, força o mapeamento de CR para NL.

Referência: 2.3
edit1: 113
edit2: 122
edit3: 181

void
nocbreak()

Retira o terminal do modo CBREAK. Após esta chamada, caracteres digitados são entregues ao programa, após a digitação de CR ou NL.

void
nodelay(jan, flag)

Se indicador for TRUE, transforma a rotina wgetch(jan) numa rotina não-bloqueada. Se não houver caracteres pendentes na entrada, wgetch retorna ERR.

Referência: 5.1

void
noecho()

Não ecoa os caracteres lidos com wgetch.

Referência: 2.3, 4.2.1
edit1: 111
edit2: 120
edit3: 179, 514

void
nonl()
Não mapeia os caracteres CR e NL na entrada ou saída.

Referência: 2.3

void
noraw()
Retira o terminal do modo RAW (ver raw).

void
noxon()
Desabilita o tratamento do protocolo XON-XOFF na entrada.

Referência: 2.3
edit1: 114
edit2: 123
edit3: 182

void
nscroll(n)
Equivalente a wnscroll(stdscr, n).

Referência: 3.2.1
edit2: 252, 256
edit3: 318, 322

OK

Valor retornado pela maioria da rotinas, em caso de sucesso (ver ERR e ERRO).

Referência: 2.3, 2.4, 3.2.1, 4.2.1, 4.2.2, 4.2.4, 4.3.4
edit1: 116, 160, 282 ,294, 300, 366, 367
edit2: 130, 174, 264, 338, 350, 356, 422, 423
edit3: 197, 219, 337, 411, 423, 436, 545, 546, 641, 653, 665, 677

void
overlay(janorig, jandest)

Copia o texto de janorig na janela jandest. Brancos presentes em janorig não são copiados.

void
overwrite(janorig, jandest)

Como overlay, mas copia todos os caracteres.

Referência: 4.3.4
edit3: 892, 924, 925

void
pnoutrefresh(jan, yo, xo, y_inf, x_inf, y_sup, x_sup)

Copia um retângulo da janela dinâmica jan na posição (y_inf, x_inf) da tela virtual curscr. O retângulo inicia na posição (yo, xo) de jan. Na tela virtual, o retângulo é definido pelas posições (y_inf, x_inf) e (y_sup, x_sup).

void
prefresh(jan, yo, xo, y_inf, x_inf, y_sup, x_sup)
Como pnoutrefresh, mas chama doupdate para atualizar a tela.

Referência: 5.3

void
printw(*formato* , *arg* ...)
Como mvprintw, mas sem movimentação do cursor da janela.

void
putp(*str*)
Equivalente a tputs(str, 1, putchar).

void
raw()
Coloca o terminal no modo RAW. Este modo é semelhante ao modo CBREAK (ver cbreak), mas não trata os caracteres de interrupção e de controle de fluxo.

void
refresh()
Equivalente a wrefresh(stdscr).

Referência: 2.4, 4.2.1, 4.2.3, 4.3.4, 5.1, 5.3
edit1: 186, 335
edit2: 200, 391
edit3: 266, 625

void
reset_prog_mode()
Coloca o terminal no **modo programa** (ver def_prog_mode).

Referência: 4.2.4
edit3: 483

void
reset_shell_mode()
Coloca o terminal no **modo shell** (ver def_shell_mode).

Referência: 4.2.4
edit3: 479

void
resetterm()
Ver reset_shell_mode.

void
resetty()
Restaura o modo do terminal salvo com a rotina savetty.

void
_sai_carac(c)
Envia o caractere c para a tela.

Referência: 4.2.4, 6
edit3: 957, 959

void
saveterm()
Ver def_prog_mode.

void
savetty()
Salva o modo do terminal. Este modo pode ser restaurado com a rotina resetty.

int
scanw(formato , arg ...)
Como mvscanw, mas sem movimentação do cursor.

void
scroll(jan)
Equivalente a wnscroll(jan, 1).

Referência: 3.2.1

void
scrollok(jan, flag)
Se indicador for TRUE, a região de rolamento da janela jan será rolado se um caractere for digitado na última posição da região de rolamento, ou se NL for colocado na última linha desta região.

TELA *
set_term(term)
Term passa a ser o terminal controlado pela CURSES. O terminal anterior é retornado.

int
setscrreg(prim_linha, ult_linha)
Equivalente à wsetscrreg(stdscr, prim_linha, ult_linha).

Referência: 3.2.1
edit2: 124
edit3: 183

void
setterm(nome)
Equivalente a setupterm(nome, 1, NULL).

int
setupterm(nome, fd, reterr)
A descrição do terminal "nome" é lida numa estrutura interna. Caracteres para o terminal serão enviados para o descritor fd. Se reterr for diferente de NULL, o valor de qualquer erro será armazenado em *reterr, e setupterm retornará ERR, em caso de erro. Se reterr for NULL, setupterm imprime uma mensagem em caso de erro, e chama exit.

Referência: 6

void
standend()
Equivalente a wstandend(stdscr).

Referência: 3.2.3, 4.3.1, 4.3.3
edit2: 318
edit3: 391, 763

void
standout()

Equivalente a wstandout(stdscr).

Referência: 3.2.3, 4.3.1, 4.3.3
edit2: 315
edit3: 388, 790

JANELA *
stdscr

Janela-padrão criada com initscr, do tamanho da tela do terminal.

Referência: 2.2, 2.3, 2.4, 3.2.1, 3.2.2, 3.2.3, 4.3.1, 4.3.3
edit1: 123, 185, 243, 260, 333, 365
edit2: 125, 126, 137, 199, 260, 297, 314, 389, 421
edit3: 184, 185, 204, 265, 326, 370, 387, 449, 455, 460, 484, 515, 609, 624, 707, 758, 784, 841, 877, 909, 914, 919, 921

JANELA *
subwin(janorig, numlinhas, numcolunas, y, x)

Cria uma subjanela da janela-mãe janorig. A subjanela corresponde ao retângulo (y,x) a (y+numlinhas,x+numcolunas) da janela-mãe.

Referência: 4.3.4
edit3: 449, 919, 921

int
tem_pendencia()
Retorna TRUE, se houver caracteres pendentes na entrada (caracteres digitados, mas não lidos).

Referência: 5.1

void
touchwin(jan)
O próximo wnoutrefresh ou wrefresh aplicado a esta janela copiará todo o texto da janela para a tela virtual curscr.

Referência: 4.2.1, 4.3.4
edit3: 515

char *
tparm(str, p1, p2, ..., p9)
Aplica os parâmetros p1 a p9 à cadeia str e retorna o resultado.

Referência: 6

void
tputs(str, numlin, putc)
A cadeia str é enviada para o terminal, caractere a caractere, com a rotina (*putc)(). Se a cadeia str especificar uma atraso, este será multiplicado por numlin (o número de linhas afetadas).

Referência: 4.2.4, 6
edit3: 959

TRUE

Valor igual a 1.

Referência: 2.4, 3.2.1, 3.2.2, 4.2.1, 4.2.3, 4.2.4, 4.3.3, 4.3.4, 5.1

edit1: 184
edit2: 125, 126, 127, 128, 198, 260
edit3: 185, 185, 186, 187, 194, 195, 196, 264. 326, 484, 622, 707, 743, 744, 753

int
typeahead(fd)

A verificação de caracteres pendentes durante doupdate é feita usando o descritor fd. Se fd for -1, a verificação não é feita.

Referência: 5.1

char *
unctrl(c)

Retorna uma representação visível do caractere c como cadeia.

char *
_versao()

Retorna o número da versão da descrição TERMINFO do terminal.

void
vidattr(atributos)

Como vidputs, mas usa putchar para imprimir os caracteres.

void
vidputs(atributos, putc)
Liga os atributos indicados no terminal, usando a rotina (*putc)() para enviar os caracteres ao terminal.

void
wacent(jan, flag)
Se indicador for TRUE, permite que caracteres acentuados da língua portuguesa sejam digitados e exibidos na janela jan. Com indicador FALSE, tais caracteres são desacentuados.

Referência: 4.2.1
edit3: 196

int
waddch(jan, c)
Adiciona o caractere c na janela jan, na posição atual do cursor.

Referência: 4.2.1

int
waddstr(jan, str)
Adiciona a cadeia str na janela jan na posição atual do cursor.

Referência: 4.3.4

void
wattroff(jan, atributos)
Desliga os atributos indicados da janela jan.

void
wattron(jan, atributos)
Liga os atributos indicados da janela jan.

void
wattrset(jan, atributos)
Liga os atributos indicados da janela jan, desligando os demais.

void
wclear(jan)
Preenche a janela jan de brancos. A próxima chamada a wrefresh(jan) apagará a tela e a redesenhará por inteiro.

void
wclrtobot(jan)
Preenche a janela jan de brancos, a partir da posição atual do cursor, até o fim da janela.

void
wclrtoeol(jan)
Preenche a linha atual da janela jan com brancos, a partir da posição atual do cursor.

Referência: 4.2.1
edit3: 509

void
wdelch(jan)
Remove a caractere sob o cursor da janela jan.

void
wdeleteln(jan)
A linha atual da janela jan é removida.

void
werase(jan)
Preenche a janela jan com brancos.

int
wgetch(jan)
Obtém um caractere do terminal, através da janela jan.

void
wgetstr(jan, str)
Lê uma cadeia da janela jan e a armazena em str.

Referência: 4.2.1
edit3: 513

int
winch(jan)
Retorna o caractere na posição do cursor da janela jan.

Referência: 4.2.2
edit3: 549, 554

WINDOW
Equivalente a JANELA.

int
winsch(jan, c)
Insere o caractere c na posição atual do cursor da janela jan.

void
winsertln(jan)
Insere uma linha em branco na posição atual do cursor da janela jan.

void
wmodoabc(jan, flag)
Se indicador for TRUE, converte seqüências de acentuação em letras acentuadas, na entrada, e letras acentuadas em seqüências de acentuação, na saída.

Referência: 4.2.1
edit3: 195

int
wmove(jan, y, x)
Move o cursor da janela jan para a posição (y, x).

Referência: 4.2.1, 4.2.2
edit3: 568

void
wnoutrefresh(jan)
Copia a informação mudada em jan para a tela virtual curscr.

Referência: 4.3.4
edit3: 930

void
wnscroll(jan, n)
Rola a região de rolamento da janela jan de n linhas para cima. Se n for negativo, o rolamento é para baixo.

void
wprintw(jan, formato , arg ...)
Equivalente a mvprintw, mas sem movimentação prévia do cursor.

Referência: 4.2.1, 5.2
edit3: 508

void
wrefresh(jan)
Atualiza a janela jan na tela do terminal.

Referência: 4.2.1, 4.2.4, 4.3.4
edit3: 511

int
wscanw(jan, formato , arg ...)
Equivalente a mvwscanw, mas sem movimentação prévia do cursor.

int
wsetscrreg(jan, prim_linha, ult_linha)
Define a região de rolamento da janela jan como sendo da linha prim_linha à linha ult_linha.

void
wstandend(jan)
Equivalente a wattrset(jan, 0).

void
wstandout(jan)
Liga o atributo A_STANDOUT da janela jan.

void
xon()
Habilita o tratamento pelo sistema operacional dos caracteres de controle de fluxo provenientes do terminal.

Controlador
Em inglês, "driver".

Indicador
Armazenador de condição. Em inglês, "flag".

Octeto
Unidade de informação composta por oito bits. Em inglês, "byte".

Página intencionalmente em branco.

Computadores e Sistemas Brasileiros S/A.

MÓDULO 2

Conhecendo o TERMINFO ...

Página intencionalmente em branco.

CAPÍTULO 1

APRESENTANDO O TERMINFO...

TERMINFO consiste em uma coleção de descrições de terminais e de rotinas usadas para manipulá-los. As rotinas que usam as descrições dos terminais estão apresentadas no Módulo 3 deste manual. Neste módulo que estamos iniciando, é descrito o formato-fonte das descrições de terminais.

Uma descrição de terminal cadastra todas as características deste terminal, de acordo com um modelo geral, semelhante ao padrão ANSI X3.64. Uma característica pode especificar:

- Se o terminal tem ou não determinada funcionalidade (por exemplo, se ele utiliza o conjunto de caracteres ABICOMP);

- O tamanho de determinado recurso (por exemplo, o número de linhas na tela do terminal);

- Seqüências de caracteres para controlar o terminal (por exemplo, como apagar a tela);

- Seqüências de caracteres enviadas pelo terminal para o computador (por exemplo, os caracteres enviados, ao pressionar a tecla de função F1).

Um programa pode controlar um terminal de forma independente de hardware, fazendo uso das rotinas do TERMINFO, para manipulá-lo diretamente, ou usando as rotinas do CURSES, para manipular objetos de mais alto nível (janelas).

1.1 SINTAXE GERAL

Um arquivo-fonte pode descrever mais de um terminal, descrição esta que deve começar na primeira coluna de uma linha. Se ela tiver continuidade em linhas subseqüentes, estas devem ter pelo menos um caractere branco (espaço ou tabulação) no seu início.

Uma descrição consiste em campos, separados por uma vírgula. Qualquer espaço em branco presente após a vírgula de separação de campos é descartada. Uma linha iniciando com o caractere '#' é um comentário. Um campo consiste em um nome de característica, possivelmente seguido de **informações adicionais**. Se um caractere '.' estiver presente no início de um campo, o campo inteiro é descartado. Se dois ou mais campos de uma descrição tiverem o mesmo **nome de característica**, o primeiro é retido, e os demais são descartados.

Para compilar um arquivo-fonte contendo descrições de terminais ver **tic** nas rotinas do TERMINFO, no Módulo 3 deste manual.

1.2 SINTAXE DOS NOMES DO TERMINAL

O primeiro campo de uma descrição é um campo especial fornecendo os nomes pelos quais o terminal é conhecido, separados pelo caractere '|'. O primeiro nome dado é o mais comumente usado. O último nome dado (chamado **nome extenso**) é uma descrição mais extensa do terminal, seu modelo, etc. Os demais nomes são sinônimos do primeiro. Nenhum nome, com exceção do nome extenso, deve conter letras maiúsculas ou espaços em branco. Finalmente, os nomes dos terminais cadastrados no TERMINFO devem ser únicos.

1.3 **RESUMO DAS CARACTERÍSTICAS**

Cada característica reconhecida pelo TERMINFO possui três nomes. O **nome da variável** é o símbolo pelo qual o programador utiliza a característica com as rotinas do TERMINFO. Quando usa CURSES, as características do terminal não são usadas diretamente. O nome **terminfo** é o nome da característica usada no arquivo-fonte descrevendo o terminal. O nome **termcap** é sinônimo do nome de variável, sendo usado quando o programador usa as rotinas de compatibilização TERMCAP (descritas no Módulo 3 deste manual). Nomes TERMINFO têm, no máximo, cinco caracteres. Os nomes TERMCAP sempre têm dois caracteres.

As características podem ser do tipo:

- Booleano (o terminal tem ou não uma determinada funcionalidade);

- Numérico (o tamanho de uma determinada característica);

- Cadeia (seqüências de caracteres enviadas para, ou recebidas do terminal).

As características do tipo cadeia podem ser de saída (enviadas ao terminal) ou de entrada (recebidas do terminal). Os nomes TERMINFO descrevendo seqüências de entrada sempre começam com key_ (as demais seqüências são de saída). Estas últimas podem ser dependentes de **parâmetros** (por exemplo, para movimentar o cursor, a seqüência enviada ao terminal depende da linha e coluna desejadas). Qualquer característica deste tipo deve ser entregue à rotina **tparm**, para substituição de parâmetros, antes de ser enviada para o terminal com a rotina **tputs**.

Seqüências de saída podem especificar atrasos a serem embutidos na transmissão da seqüência. Em alguns casos, o atraso específico deve ser multiplicado pelo número de linhas da tela afetadas pela seqüência.

Uma característica particular foge às regras expostas. Um campo do tipo:

```
use=nome
```

significa que todas as características do terminal chamado **nome** são incorporadas à descrição corrente. Permanece a regra de que apenas a primeira ocorrência de uma característica é considerada. Se um nome TERMINFO for seguido do caractere , esta característica é cancelada, isto é, não faz parte da descrição. Por exemplo,

```
ttyxxx|terminal exemplo,
rev , use=ttyabc
```

define um terminal chamado **ttyxxx**, com todas as características do terminal **ttyabc**, mas sem a característica **rev**.

1.4 EXEMPLO

Exemplo das descrições de terminais usados pela COBRA.

```
TI200,
   am,xmc#1,cols#80,lines#25,bel=^G,bs=^H,
   cr=^M,cub1=^R,cud1=^S,cuf1=^T,cuu1=^Q,
   cup=^_%p1%'\s'%+%c%p2%'\s'%+%c,dch1=^X,
   dl1=\E^X,home=^_\040\040,clear=^L,ff=^L,
   ind=^J,il1=\E^Z,nel=^J^M,ht=^I,rev=\ET,
   blink=\EB,bold=\EA,invis=\EH,sms0=\ET,
   sgr0=\E@,smul=\E`,rmul=\E@,rms0=\E@,
   kcuu1=^Q,Kcuf1=^T,kcud1=^S,kcub1=^R,
   cnorm=^_^F,civis=^_^U,vit=V1.1,

TI300,
   am,cols#80,lines#28,ab,bel=^G,bs=^H,
   cr=^M,cub1=\E[D,cud1=\E[B,cuf1=\E[C,
   cuu1=\E[H,cup=\E[%i%p1%d;%p2%dH,
   dch1=\E[P,dl1=\E[M,ed=\E[J,home=\E[H,
   clear=\E[H\E[J,el=\E[K,ff=^L,ind=\ED,
   il1=\E[L,nel=^J,ht=^I,ri=\EM,rev=\E[7m,
   blink=\E[5m,bold=\E[1m,invis=\E[8m,
   sms0=\E[7m,sgr0=\E[0m,smul=\E[4m,
   rmul=\E[0m,rms0=\E[0m,khome=\233H,
   kcuf1=\233C,kcud1=\233B,
   kcub1=\233D,kcuu1=\233A,
   cnorm=\E[3;9v,civis=\E[3;0v,vit=V1.1,
```

Onde:

TI 200 - Descritor usado para os terminais TI 200, TI 300 no modo TI 200 e monitor PC.

TI 300 - Descritor usado para o terminal TI 300 no modo ANSI.

Observação:

O descritor referente ao monitor do PC (TI 300 PC. EM) é igual ao descritor do terminal TI 200 e seu arquivo terminfo equivalente é criado quando da implantação do sistema.

CAPÍTULO 2

CARACTERIZANDO O TERMINFO...

A tabela que se segue resume as características do TERMINFO, indicando, inclusive, o número de parâmetros para as características de saída, e se o atraso é do tipo multiplicativo. O número de seqüência dado refere-se às descrições detalhadas das características.

2.1 CARACTERÍSTICAS EM ORDEM DE NOME DE VARIÁVEL

. CARACTERÍSTICAS BOOLEANAS

. CARACTERÍSTICAS BOOLEANAS

SEQ	NOME DA VARIÁVEL	NOME TERMINFO	NOME TERMCAP
1	abicomp	ab	ab
2	abicomp_glitch	abg	ag
3	auto_left_margin	bw	bw
4	auto_right_margin	am	am
5	beehive_glitch	xsb	xb
6	ceol_standout_glitch	xhp	xs
7	eat_newline_glitch	xenl	xn
8	erase_overstrike	eo	eo
9	generic_type	gn	gn
10	hard_copy	hc	hc
11	has_meta_key	km	km
12	has_status_line	hs	hs
13	insert_null_glitch	in	in
14	memory_above	da	da
15	memory_below	db	db
16	move_insert_mode	mir	mi

17	move_standout_mode	msgr	ms
18	over_strike	os	os
19	status_line_esc_ok	eslok	es
20	teleray_glitch	xt	xt
21	tilde_glitch	hz	hz
22	transparent_underline	ul	ul
23	xon_xoff	xon	xo

. CARACTERÍSTICAS NUMÉRICAS

SEQ	NOME DA VARIÁVEL	NOME TERMINFO	NOME TERMCAP
24	columns	cols	co
25	init_tabs	it	it
26	lines	lines	li
27	lines_of_memory	lm	lm
28	magic_cookie_glitch	xmc	sg
29	padding_baud_rate	pb	pb
30	virtual_terminal	vt	vt
31	width_status_line	wsl	ws

. CARACTERISTICAS DO TIPO CADEIA

SEQ	NOME DA VARIÁVEL	NOME TERMINFO	NOME TERMCAP	NÚMERO DE PARÂMETROS	ATRASO MULTIPLIC.
32	back_tab	cbt	bt		
33	bell	bel	bl		
34	carriage_return	cr	cr		sim
35	change_scroll_region	csr	cs	2	
36	clear_all_tabs	tbc	ct		
37	clear_screen	clear	cl		sim
38	clr_eol	el	ce		
39	clr_eos	ed	cd		sim
40	column_address	hpa	ch	1	
41	command_character	cmdch	CC		
42	cursor_address	cup	cm	2	
43	cursor_down	cud1	do		
44	cursor_home	home	ho		
45	cursor_invisible	civis	vi		
46	cursor_left	cub1	le		
47	cursor_mem_address	mrcup	CM		
48	cursor_normal	cnorm	ve		
49	cursor_right	cuf1	nd		
50	cursor_to_ll	ll	ll		
51	cursor_up	cuu1	up		
52	cursor_visible	cvvis	vs		
53	delete_character	dch1	dc		sim
54	delete_line	dl1	dl		sim
55	dis_status_line	dsl	ds		
56	down_half_line	hd	hd		
57	e_e_alt	eae	EE		
58	enter_alt_charset_mode	smacs	as		
59	enter_blink_mode	blink	mb		
60	enter_bold_mode	bold	md		
61	enter_ca_mode	smcup	ti		
62	enter_delete_mode	smdc	dm		
63	enter_dim_mode	dim	mh		
64	enter_insert_mode	smir	im		
65	enter_protected_mode	prot	mp		
66	enter_reverse_mode	rev	mr		
67	enter_secure_mode	invis	mk		
68	enter_standout_mode	smso	so		
69	enter_underline_mode	smul	us		
70	erase_chars	ech	ec	1	
71	e_s_alt	eas	ES		
72	exit_alt_charset_mode	rmacs	ae		
73	exit_attribute_mode	sgr0	me		
74	exit_ca_mode	rmcup	te		
75	exit_delete_mode	rmdc	ed		
76	exit_insert_mode	rmir	ei		
77	exit_standout_mode	rmso	se		
78	exit_underline_mode	rmul	ue		

79	flash_screen	flash	vb	
80	form_feed	ff	ff	sim
81	from_status_line	fsl	fs	
82	init_1string	is1	i1	
83	init_2string	is2	is	
84	init_3string	is3	i3	
85	init_file	if	if	
86	init_prog	iprog	iP	
87	insert_character	ich1	ic	
88	insert_line	il1	al	sim
89	insert_padding	ip	ip	sim
90	key_a1	ka1	K1	
91	key_a3	ka3	K3	
92	key_b2	kb2	K2	
93	key_c1	kc1	K4	
94	key_c3	kc3	K5	
95	key_backspace	kbs	kb	
96	key_catab	ktbc	k;	
97	key_clear	kclr	kC	
98	key_ctab	kctab	kt	
99	key_dc	kdch1	kD	
100	key_dl	kdl1	kL	
101	key_down	kcud1	kd	
102	key_eic	krmir	kM	
103	key_eol	kel	kE	
104	key_eos	ked	kS	
105	key_f0	kf0	k0	
106	key_f1	kf1	k1	
107	key_f2	kf2	k2	
108	key_f3	kf3	k3	
109	key_f4	kf4	k4	
110	key_f5	kf5	k5	
111	key_f6	kf6	k6	
112	key_f7	kf7	k7	
113	key_f8	kf8	k8	
114	key_f9	kf9	k9	
115	key_f10	kf10	ka	
116	key_home	khome	kh	
117	key_ic	kich1	kI	
118	key_il	kil1	kA	
119	key_left	kcub1	kl	
120	key_ll	kll	kH	
121	key_npage	knp	kN	
122	key_ppage	kpp	kP	
123	key_right	kcuf1	kr	
124	key_sf	kind	kF	
125	key_sr	kri	kR	
126	key_stab	khts	kT	

127	key_up	kcuu1	ku	
128	keypad_local	rmkx	ke	
129	keypad_xmit	smkx	ks	
130	lab_f0	lf0	l0	
131	lab_f1	lf1	l1	
132	lab_f2	lf2	l2	
133	lab_f3	lf3	l3	
134	lab_f4	lf4	l4	
135	lab_f5	lf5	l5	
136	lab_f6	lf6	l6	
137	lab_f7	lf7	l7	
138	lab_f8	lf8	l8	
139	lab_f9	lf9	l9	
140	lab_f10	lf10	la	
141	map_ent	mape	ME	
142	map_saida	maps	MS	
143	meta_off	rmm	mo	
144	meta_on	smm	mm	
145	newline	nel	nw	
146	pad_char	pad	pc	
147	parm_dch	dch	DC	1
sim				
148	parm_delete_line	dl	DL	1
sim				
149	parm_down_cursor	cud	DO	1
sim				
150	parm_ich	ich	IC	1
sim				
151	parm_index	indn	SF	1
152	parm_insert_line	il	AL	1
sim				
153	parm_left_cursor	cub	LE	1
154	parm_right_cursor	cuf	RI	1
sim				
155	parm_rindex	rin	SR	1
156	parm_up_cursor	cuu	UP	1
sim				
157	pkey_key	pfkey	PK	2
158	pkey_local	pfloc	pl	2
159	pkey_xmit	pfx	px	2
160	print_screen	mc0	ps	
161	prtr_non	mc5p	pO	
162	prtr_off	mc4	pf	
163	prtr_on	mc5	po	
164	repeat_char	rep	rp	2

sim

165	reset_1string	rs1	r1	
166	reset_2string	rs2	r2	
167	reset_3string	rs3	r3	
168	reset_file	rf	rf	
169	restore_cursor	rc	rc	
170	row_address	vpa	cv	1
171	s_e_alt	sae	SE	
172	save_cursor	sc	sc	
173	scroll_forward	ind	sf	
174	scroll_reverse	ri	sr	
175	set_attributes	sgr	sa	6
176	set_tab	hts	st	
177	set_window	wind	wi	4
178	s_s_alt	sas	SS	
179	tab	ht	ta	
180	to_status_line	tsl	ts	
181	underline_char	uc	uc	
182	up_half_line	hu	hu	
183	vers_i_t	vit	VI	

2.2 CARACTERÍSTICAS EM ORDEM DE NOME TERMINFO

SEQ	NOME TERMINFO	NOME DA VARIÁVEL	NOME TERMCAP
1	ab	abicomp	ab
2	abg	abicomp_glitch	ag
4	am	auto_right_margin	am
33	bel	bell	bl
59	blink	enter_blink_mode	mb
60	bold	enter_bold_mode	md
3	bw	auto_left_margin	bw
32	cbt	back_tab	bt
45	civis	cursor_invisible	vi
37	clear	clear_screen	cl
41	cmdch	command_character	CC
48	cnorm	cursor_normal	ve
24	cols	columns	co
34	cr	carriage_return	cr
35	csr	change_scroll_region	cs
153	cub	parm_left_cursor	LE
46	cub1	cursor_left	le
149	cud	parm_down_cursor	DO
43	cud1	cursor_down	do
154	cuf	parm_right_cursor	RI
49	cuf1	cursor_right	nd
42	cup	cursor_address	cm
156	cuu	parm_up_cursor	UP
51	cuu1	cursor_up	up
52	cvvis	cursor_visible	vs
14	da	memory_above	da
15	db	memory_below	db
147	dch	parm_dch	DC
53	dch1	delete_character	dc
63	dim	enter_dim_mode	mh
148	dl	parm_delete_line	DL
54	dl1	delete_line	dl
55	dsl	dis_status_line	ds

57	eae	e_e_alt	EE
71	eas	e_s_alt	ES
70	ech	erase_chars	ec
39	ed	clr_eos	cd
38	el	clr_eol	ce
8	eo	erase_overstrike	eo
19	eslok	status_line_esc_ok	es
80	ff	form_feed	ff
79	flash	flash_screen	vb
81	fsl	from_status_line	fs
9	gn	generic_type	gn
10	hc	hard_copy	hc
56	hd	down_half_line	hd
44	home	cursor_home	ho
40	hpa	column_address	ch
12	hs	has_status_line	hs
179	ht	tab	ta
176	hts	set_tab	st
182	hu	up_half_line	hu
21	hz	tilde_glitch	hz
150	ich	parm_ich	IC
87	ich1	insert_character	ic
85	if	init_file	if
152	il	parm_insert_line	AL
88	il1	insert_line	al
13	in	insert_null_glitch	in
173	ind	scroll_forward	sf
151	indn	parm_index	SF
67	invis	enter_secure_mode	mk
89	ip	insert_padding	ip
86	iprog	init_prog	iP
82	is1	init_1string	i1
83	is2	init_2string	is
84	is3	init_3string	i3
25	it	init_tabs	it
90	ka1	key_a1	K1
91	ka3	key_a3	K3

92	kb2	key_b2	K2
95	kbs	key_backspace	kb
93	kc1	key_c1	K4
94	kc3	key_c3	K5
97	kclr	key_clear	kC
98	kctab	key_ctab	kt
119	kcub1	key_left	kl
101	kcud1	key_down	kd
123	kcuf1	key_right	kr
127	kcuu1	key_up	ku
99	kdch1	key_dc	kD
100	kdl1	key_dl	kL
104	ked	key_eos	kS
103	kel	key_eol	kE
105	kf0	key_f0	k0
106	kf1	key_f1	k1
115	kf10	key_f10	ka
107	kf2	key_f2	k2
108	kf3	key_f3	k3
109	kf4	key_f4	k4
110	kf5	key_f5	k5
111	kf6	key_f6	k6
112	kf7	key_f7	k7
113	kf8	key_f8	k8
114	kf9	key_f9	k9
116	khome	key_home	kh
126	khts	key_stab	kT
117	kich1	key_ic	kI
118	kil1	key_il	kA
124	kind	key_sf	kF
120	kll	key_ll	kH
11	km	has_meta_key	km
121	knp	key_npage	kN
122	kpp	key_ppage	kP
125	kri	key_sr	kR
102	krmir	key_eic	kM
96	ktbc	key_catab	k;

130	lf0	lab_f0	l0
131	lf1	lab_f1	l1
140	lf10	lab_f10	la
132	lf2	lab_f2	l2
133	lf3	lab_f3	l3
134	lf4	lab_f4	l4
135	lf5	lab_f5	l5
136	lf6	lab_f6	l6
137	lf7	lab_f7	l7
138	lf8	lab_f8	l8
139	lf9	lab_f9	l9
26	lines	lines	li
50	ll	cursor_to_ll	ll
27	lm	lines_of_memory	lm
141	mape	map_ent	ME
142	maps	map_saida	MS
160	mc0	print_screen	ps
162	mc4	prtr_off	pf
163	mc5	prtr_on	po
161	mc5p	prtr_non	pO
16	mir	move_insert_mode	mi
47	mrcup	cursor_mem_address	CM
17	msgr	move_standout_mode	ms
145	nel	newline	nw
18	os	over_strike	os
146	pad	pad_char	pc
29	pb	padding_baud_rate	pb
157	pfkey	pkey_key	PK
158	pfloc	pkey_local	pl
159	pfx	pkey_xmit	px
65	prot	enter_protected_mode	mp
169	rc	restore_cursor	rc
164	rep	repeat_char	rp
66	rev	enter_reverse_mode	mr
168	rf	reset_file	rf
174	ri	scroll_reverse	sr
155	rin	parm_rindex	SR
72	rmacs	exit_alt_charset_mode	ae

74	rmcup	exit_ca_mode	te
75	rmdc	exit_delete_mode	ed
76	rmir	exit_insert_mode	ei
128	rmkx	keypad_local	ke
143	rmm	meta_off	mo
77	rmso	exit_standout_mode	se
78	rmul	exit_underline_mode	ue
165	rs1	reset_1string	r1
166	rs2	reset_2string	r2
167	rs3	reset_3string	r3
171	sac	s_e_alt	SE
178	sas	s_s_alt	SS
172	sc	save_cursor	sc
175	sgr	set_attributes	sa
73	sgr0	exit_attribute_mode	me
58	smacs	enter_alt_charset_mode	as
61	smcup	enter_ca_mode	ti
62	smdc	enter_delete_mode	dm
64	smir	enter_insert_mode	im
129	smkx	keypad_xmit	ks
144	smm	meta_on	mm
68	smso	enter_standout_mode	so
69	smul	enter_underline_mode	us
36	tbc	clear_all_tabs	ct
180	tsl	to_status_line	ts
181	uc	underline_char	uc
22	ul	transparent_underline	ul
183	vit	vers_i_t	VI
170	vpa	row_address	cv
30	vt	virtual_terminal	vt
177	wind	set_window	wi
31	wsl	width_status_line	ws
7	xenl	eat_newline_glitch	xn
6	xhp	ceol_standout_glitch	xs
28	xmc	magic_cookie_glitch	sg
23	xon	xon_xoff	xo
5	xsb	beehive_glitch	xb
20	xt	teleray_glitch	xt

2.3 CARACTERÍSTICAS EM ORDEM DE NOME TERMCAP

SEQ	NOME TERMCAP	NOME DA VARIÁVEL	NOME TERMINFO
1	ab	abicomp	ab
72	ae	exit_alt_charset_mode	rmacs
2	ag	abicomp_glitch	abg
88	al	insert_line	il1
152	AL	parm_insert_line	il
4	am	auto_right_margin	am
58	as	enter_alt_charset_mode	smacs
33	bl	bell	bel
32	bt	back_tab	cbt
3	bw	auto_left_margin	bw
41	CC	command_character	cmdch
39	cd	clr_eos	ed
38	ce	clr_eol	el
40	ch	column_address	hpa
37	cl	clear_screen	clear
42	cm	cursor_address	cup
47	CM	cursor_mem_address	mrcup
24	co	columns	cols
34	cr	carriage_return	cr
35	cs	change_scroll_region	csr
36	ct	clear_all_tabs	tbc
170	cv	row_address	vpa
14	da	memory_above	da
15	db	memory_below	db
53	dc	delete_character	dch1
147	DC	parm_dch	dch
54	dl	delete_line	dl1
148	DL	parm_delete_line	dl
62	dm	enter_delete_mode	smdc
43	do	cursor_down	cud1
149	DO	parm_down_cursor	cud
55	ds	dis_status_line	dsl

150	IC	parm_ich	ich
85	if	init_file	if
64	im	enter_insert_mode	smir
13	in	insert_null_glitch	in
86	iP	init_prog	iprog
89	ip	insert_padding	ip
83	is	init_2string	is2
25	it	init_tabs	it
105	k0	key_f0	kf0
90	K1	key_a1	ka1
106	k1	key_f1	kf1
92	K2	key_b2	kb2
107	k2	key_f2	kf2
91	K3	key_a3	ka3
108	k3	key_f3	kf3
93	K4	key_c1	kc1
109	k4	key_f4	kf4
94	K5	key_c3	kc3
110	k5	key_f5	kf5
111	k6	key_f6	kf6

70	ec	erase_chars	ech
75	ed	exit_delete_mode	rmdc
57	EE	e_e_alt	eae
76	ei	exit_insert_mode	rmir
8	eo	erase_overstrike	eo
71	ES	e_s_alt	eas
19	es	status_line_esc_ok	eslok
80	ff	form_feed	ff
81	fs	from_status_line	fsl
9	gn	generic_type	gn
10	hc	hard_copy	hc
56	hd	down_half_line	hd
44	ho	cursor_home	home
12	hs	has_status_line	hs
182	hu	up_half_line	hu
21	hz	tilde_glitch	hz
82	i1	init_1string	is1
84	i3	init_3string	is3
87	ic	insert_character	ich1

112	k7	key_f7	kf7
113	k8	key_f8	kf8
114	k9	key_f9	kf9
96	k;	key_catab	ktbc
115	ka	key_f10	kf10
118	kA	key_il	kil1
95	kb	key_backspace	kbs
97	kC	key_clear	kclr
99	kD	key_dc	kdch1
101	kd	key_down	kcud1
128	ke	keypad_local	rmkx
103	kE	key_eol	kel
124	kF	key_sf	kind
116	kh	key_home	khome
120	kH	key_ll	kll
117	kI	key_ic	kich1
100	kL	key_dl	kdl1
119	kl	key_left	kcub1
11	km	has_meta_key	km
102	kM	key_eic	krmir
121	kN	key_npage	knp
122	kP	key_ppage	kpp
123	kr	key_right	kcuf1
125	kR	key_sr	kri
129	ks	keypad_xmit	smkx
104	kS	key_eos	ked
98	kt	key_ctab	kctab
126	kT	key_stab	khts
127	ku	key_up	kcuu1
130	l0	lab_f0	lf0
131	l1	lab_f1	lf1
132	l2	lab_f2	lf2
133	l3	lab_f3	lf3
134	l4	lab_f4	lf4
135	l5	lab_f5	lf5
136	l6	lab_f6	lf6

137	17	lab_f7	lf7
138	18	lab_f8	lf8
139	19	lab_f9	lf9
140	la	lab_f10	lf10
46	le	cursor_left	cub1
153	LE	parm_left_cursor	cub
26	li	lines	lines
50	ll	cursor_to_ll	ll
27	lm	lines_of_memory	lm
59	mb	enter_blink_mode	blink
60	md	enter_bold_mode	bold
73	me	exit_attribute_mode	sgr0
141	ME	map_ent	mape
63	mh	enter_dim_mode	dim
16	mi	move_insert_mode	mir
67	mk	enter_secure_mode	invis
144	mm	meta_on	smm
143	mo	meta_off	rmm
65	mp	enter_protected_mode	prot
66	mr	enter_reverse_mode	rev
142	MS	map_saida	maps
17	ms	move_standout_mode	msgr
49	nd	cursor_right	cuf1
145	nw	newline	nel
18	os	over_strike	os
29	pb	padding_baud_rate	pb
146	pc	pad_char	pad
162	pf	prtr_off	mc4
157	PK	pkey_key	pfkey
158	pl	pkey_local	pfloc
161	pO	prtr_non	mc5p
163	po	prtr_on	mc5
160	ps	print_screen	mc0
159	px	pkey_xmit	pfx
165	r1	reset_1string	rs1
166	r2	reset_2string	rs2
167	r3	reset_3string	rs3
169	rc	restore_cursor	rc
168	rf	reset_file	rf
154	RI	parm_right_cursor	cuf

164	rp	repeat_char	rep
175	sa	set_attributes	sgr
172	sc	save_cursor	sc
77	se	exit_standout_mode	rmso
171	SE	s_e_alt	sae
151	SF	parm_index	indn
173	sf	scroll_forward	ind
28	sg	magic_cookie_glitch	xmc
68	so	enter_standout_mode	smso
155	SR	parm_rindex	rin
174	sr	scroll_reverse	ri
178	SS	s_s_alt	sas
176	st	set_tab	hts
179	ta	tab	ht
74	te	exit_ca_mode	rmcup
61	ti	enter_ca_mode	smcup
180	ts	to_status_line	tsl
181	uc	underline_char	uc
78	ue	exit_underline_mode	rmul
22	ul	transparent_underline	ul
51	up	cursor_up	cuu1
156	UP	parm_up_cursor	cuu
69	us	enter_underline_mode	smul
79	vb	flash_screen	flash
48	ve	cursor_normal	cnorm
45	vi	cursor_invisible	civis
183	VI	vers_i_t	vit
52	vs	cursor_visible	cvvis
30	vt	virtual_terminal	vt
177	wi	set_window	wind
31	ws	width_status_line	wsl
5	xb	beehive_glitch	xsb
7	xn	eat_newline_glitch	xenl
23	xo	xon_xoff	xon
6	xs	ceol_standout_glitch	xhp
20	xt	teleray_glitch	xt

CAPÍTULO 3

CLASSIFICANDO AS CARACTERÍSTICAS...

. Características Gerais

```
auto_left_margin     columns          lines
auto_right_margin    flash_screen     over_strike
bell hard_copy
```

. Movimentação do Cursor

```
carriage_return       exit_ca_mode
column_address        lines_of_memory
cursor_address        newline
cursor_down           parm_down_cursor
cursor_home           parm_left_cursor
cursor_left           parm_right_cursor
cursor_mem_address    parm_up_cursor
cursor_right          restore_cursor
cursor_to_ll          row_address
cursor_up             save_cursor
enter_ca_mode
```

. Apagamento de Regiões da Tela

```
clear_screen          clr_eos
clr_eol               erase_chars
```

. Inserção e Remoção de Linhas

```
delete_line           parm_delete_line
insert_line           parm_insert_line
```

. Inserção e Remoção de Caracteres

delete_character
enter_delete_mode
enter_insert_mode
exit_delete_mode
exit_insert_mode
insert_character
insert_null_glitch
insert_padding
move_insert_mode
parm_dch
parm_ich

. Rolamento de Regiões da Tela

change_scroll_region
memory_above
memory_below
parm_index
parm_rindex
scroll_forward
scroll_reverse

. Atributos de Vídeo

ceol_standout_glitch
cursor_invisible
cursor_normal
cursor_visible
enter_alt_charset_mode
enter_blink_mode
enter_bold_mode
enter_dim_mode
enter_protected_mode
enter_reverse_mode
enter_secure_mode
enter_standout_mode
enter_underline_mode
erase_overstrike
exit_alt_charset_mode
exit_attribute_mode
exit_standout_mode
exit_underline_mode
magic_cookie_glitch
move_standout_mode
set_attributes
transparent_underline
underline_char

. Ilha de Teclas

has_meta_key
key_a1
key_a3
key_b2
key_f2
key_f3
key_f4
key_f5
key_up
keypad_local
keypad_xmit
lab_f0

key_backspace	key_f6	lab_f1
key_c1	key_f7	lab_f10
key_c3	key_f8	lab_f2
key_catab	key_f9	lab_f3
key_clear	key_home	lab_f4
key_ctab	key_ic	lab_f5
key_dc	key_il	lab_f6
key_dl	key_left	lab_f7
key_down	key_ll	lab_f8
key_eic	key_npage	lab_f9
key_eol	key_ppage	meta_off
key_eos	key_right	meta_on
key_f0	key_sf	pkey_key
key_f1	key_sr	pkey_local
key_f10	key_stab	pkey_xmit

- Tabulação e Inicialização

back_tab	init_file	reset_3string
clear_all_tabs	init_prog	reset_file
init_1string	init_tabs	set_tab
init_2string	reset_1string	tab
init_3string	reset_2string	

- Informações de Preenchimento e Atrasos

pad_char	padding_baud_rate	xon_xoff

- Linha de Status

dis_status_line	status_line_esc_ok
from_status_line	to_status_line
has_status_line	width_status_line

. Acentuação de Caracteres

abicomp	e_e_alt	s_e_alt
abicomp_glitch	e_s_alt	s_s_alt

. Controle do Módulo Impressor

print_screen	prtr_off
prtr_non	prtr_on

. Miscelânea

command_character	map_ent	up_half_line
down_half_line	map_saida	vers_i_t
form_feed	repeat_char	virtual_terminal
generic_type	set_window	

. Características que Fogem ao Modelo de Terminal

beehive_glitch	teleray_glitch
eat_newline_glitch	tilde_glitch

CAPÍTULO 4

DESCREVENDO AS CARACTERÍSTICAS...

4.1 CARACTERÍSTICAS BOOLEANAS

A sintaxe é simplesmente o nome TERMINFO da característica que se deseja incluir na descrição. Por exemplo:

```
ttyxxx|terminal exemplo,
 ab, am, ...
```

indica um terminal abicomp (**ab**) com margem direita automática (**am**).

A descrição detalhada das características booleanas vem a seguir.

1. **abicomp** (ab, ab)

   Se verdadeiro, o terminal reconhece e gera caracteres acentuados da língua portuguesa, usando o padrão ABICOMP. A representação pode ser em 8 ou 7 bits, dependendo das características **eae**, **eas**, **sae** e **sas**.

2. **abicomp_glitch** (abg, ag)

   Se verdadeiro, o terminal emite a seqüência <acento><ASCII BS> quando um acento ('`~^) é digitado. Um verdadeiro acento (sem BS) é gerado, quando um espaço é teclado após o acento.

3. **auto_left_margin** (bw, bw)

   Se verdadeiro, a emissão de **cub1** com o cursor na posição (y,0) move o cursor para a posição (y-1, **cols**-1), se y > 0. Para y igual a 0, o efeito é indefinido.

   Se falso, o comando acima deixa o cursor numa posição indefinida.

4. **auto_right_margin** (am, am)

   Se verdadeiro, a exibição de um caractere visível enquanto o cursor está na posição (y,**cols**-1) move o cursor para a posição (y+1,0), se y < **lines**-1. Para y igual a **lines**-1, o efeito é indefinido e freqüentemente causa um rolamento da tela. O efeito de enviar **cuf1** enquanto o cursor está na posição (y,**cols**-1) não é definido.

   Se falso, o comando acima deixa o cursor na posição (y,**cols**-1).

5. **beehive_glitch** (xsb, xb)

   Se verdadeiro, o terminal não transmite os caracteres ESC ou ^C. Para estes terminais, a tecla F1 deve ser usada para representar ESC, e F2 para representar ^C.

6. **ceol_standout_glitch** (xhp, xs)

Se verdadeiro, indica que **el** deve ser emitido, para remover atributos de vídeo.

Se falso, a exibição de texto normal acima de caracteres com atributos de vídeo remove os atributos.

7. **eat_newline_glitch** (xenl, xn)

É verdadeiro, se um caractere de alimentação de linha (ASCII LF) é ignorado após um posicionamento automático do cursor devido a **am**.

Também é verdadeiro, se o posicionamento automático do cursor devido a **am** só ocorre após a recepção de um caractere adicional (além do caractere que normalmente causaria o efeito **am**).

8. **erase_overstrike** (eo, eo)

Se verdadeiro, um espaço apaga todos os caracteres presentes na posição do cursor, apesar do terminal ter a característica de sobreposição (**os**).

9. **generic_type** (gn, gn)

Se verdadeiro, esta descrição de terminal não representa um terminal específico, mas uma porta de comunicação dando acesso a vários tipos de terminais (via rede, por exemplo).

10. **hard_copy** (hc, hc)

    Se verdadeiro, o terminal é do tipo impressor. Observe que um terminal de vídeo que também possui um módulo impressor não deve receber a característica **hc**.

11. **has_meta_key** (km, km)

    Reservado.

12. **has_status_line** (hs, hs)

    Se verdadeiro, o terminal possui uma linha adicional de status acessada com os comandos **tsl**, **fsl** e **dsl**.

13. **insert_null_glitch** (in, in)

    Se verdadeiro, enquanto o terminal está no modo de inserção (**smir**), espaços em branco à direita do cursor só serão movidos para a direita, se forem espaços digitados anteriormente. Espaços em branco resultantes do apagamento de uma região da tela (por exemplo, com **clear**) não são movidos.

    Se falso, enquanto o terminal está no modo de inserção, todos os caracteres à direita do cursor são movidos para a direita, quando um caractere é inserido.

14. **memory_above** (da, da)

Se verdadeiro, indica que o rolamento para baixo (com **ri** ou **rin**) pode exibir linhas não-brancas que estavam presentes no topo da tela, anteriormente.

15. **memory_below** (db, db)

Se verdadeiro, indica que o rolamento para cima (com **ind** ou indn) ou a remoção de linhas (com **dl1** ou **dl**) pode exibir linhas não-brancas que estavam presentes na base da tela, anteriormente.

16. **move_insert_mode** (mir, mi)

Se verdadeiro, o cursor pode ser movido enquanto o terminal está no modo de inserção (**smir**).

Se falso, **rmir** deve ser usado para sair do modo de inserção, antes de mover o cursor.

17. **move_standout_mode** (msgr, ms)

Se verdadeiro, o cursor pode ser movido, enquanto o terminal está no modo de destaque (**smso**).

Se falso, **rmso** deve ser usado para sair do modo de destaque, antes de mover o cursor.

18. **over_strike** (os, os)

    Se verdadeiro, um caractere colocado na tela sobrepõe o que já está presente nesta posição, deixando ambos visíveis. Sempre é o caso para terminais impressores (**hc**).

19. **status_line_esc_ok** (eslok, es)

    Se verdadeiro, seqüências de escape e comandos especiais (por exemplo **el**) funcionam corretamente, enquanto o cursor está posicionado na linha de status (ver **hs**).

20. **teleray_glitch** (xt, xt)

    Se verdadeiro, quando o terminal recebe um caractere de tabulação, ele move o cursor para a próxima parada de tabulação, sobrepondo espaços em todos os caracteres presentes entre a posição inicial e a posição final do cursor. **Xt** também indica que o terminal tem a característica **xhp**, mesmo que esta não esteja cadastrada.

    Se falso, o posicionamento do cursor com o caractere de tabulação não destrói os caracteres entre as duas posições.

21. **tilde_glitch** (hz, hz)

    Se verdadeiro, o terminal não imprime o caractere ~ (til).

22. **transparent_underline** (ul, ul)

Se verdadeiro, o terminal pode sublinhar caracteres com o caractere '_'. Isto é uma forma restrita da característica de sobreposição (**os**).

23. **xon_xoff** (xon, xo)

Se verdadeiro, o terminal produz caracteres de controle de fluxo (^S, ^Q). Nenhum caractere de preenchimento é transmitido pela rotina **tputs** para tais terminais.

4.2 CARACTERÍSTICAS NUMÉRICAS

A sintaxe é:

nome#n

onde **nome** é um nome TERMINFO, e **n** é um número não-negativo. Por exemplo:

```
ttyxxx|terminal exemplo,
     cols#80, lines#24, ...
```

indica um terminal com oitenta colunas e 24 linhas.

Segue-se a descrição detalhada das características numéricas. Observe que, na discussão, uma referência a **n** indica o número dado após o caractere '**#**'.

24. **columns** (cols, co)

    Cada linha do terminal tem **n** colunas.

25. **init_tabs** (it, it)

    O terminal tem tabulações implementadas em hardware a cada **n** colunas.

26. **lines** (lines, li)

    A tela do terminal possui **n** linhas. Só se aplica a terminais de vídeo para os quais o conceito de "tela" faz sentido.

27. **lines_of_memory** (lm, lm)

O terminal tem **n** linhas de memória, com **n** **lines**. Se **n** for igual a zero, o terminal tem mais linhas de memória do que o tamanho da tela, mas este número não é fixo.

28. **magic_cookie_glitch** (xmc; sg)

O terminal insere **n** espaços em branco, cada vez que os atributos de vídeo são mudados. Se a mudança de atributos não deixa espaços em branco na tela, mas os novos atributos afetam todos os caracteres até o fim da linha, especifique **xmc#0**. Se uma mudança de atributos não afeta os caracteres já presentes na tela, mas apenas os novos caracteres enviados ao terminal, não especifique **xmc**.

29. **padding_baud_rate** (pb, pb)

Os atrasos gerados pelo driver de terminal após os caracteres **cr**, **ind**, **cubl**, **ff** e **tab** podem ser ignorados abaixo de uma determinada velocidade de transmissão. Esta velocidade é especificada em bits por segundo, com **pb**.

30. **virtual_terminal** (vt, vt)

Reservado.

31. **width_status_line** (wsl, ws)

N é a largura da linha de status em colunas, caso esta não tenha o mesmo número de colunas presentes nas linhas normais.

4.3 CARACTERÍSTICAS DO TIPO CADEIA

A sintaxe é a seguinte:

nome=cadeia

onde **nome** é um nome TERMINFO, e **cadeia** é a seqüência de caracteres enviada para o terminal (se **nome** for uma característica de saída), ou recebida do terminal (se **nome** for uma característica de entrada).

A cadeia pode incluir seqüências de escape, resumidas abaixo:

Seqüência	Significado
\E	ESCAPE
\e	ESCAPE
^x	controle x
\n	nova linha
\l	alimentação de linha
\r	retorno do carro
\t	tabulação
\b	volta cursor
\f	alimentação de formulário
\s	espaço
\^	^
\\	\
\,	,
\:	:
\0	nulo (igual ao valor \200, para não confundir com o terminador de **cadeia**, mas é tratado como nulo pela maioria dos terminais)
\nnn	um octeto igual a nnn em octal

Características de saída podem incluir atrasos, fornecidos pela rotina **tputs** do TERMINFO. Os atrasos podem ser embutidos em qualquer lugar da **cadeia** e têm a seguinte sintaxe:

$<n> **n** milissegundos de atraso.
$<n.m> Até uma casa decimal é permitida.
$<n*> O caractere '*' indica que o atraso deve ser multiplicado pelo número de linhas afetadas pela seqüência.

Exemplo:

```
ttyxxx|terminal exemplo,
     el=\EK$<4>, ...
```

indica que, para apagar o restante da linha, deve-se enviar os caracteres ESC e K ao terminal, seguidos de 4 milissegundos de atraso.

Observe que se o terminal tem a característica booleana **xon**, os atrasos podem ser especificados, mas não serão gerados pela rotina **tputs**.

4.3.1 TRATAMENTO DE PARÂMETROS

Parâmetros de características de saída são manipulados, usando-se uma pilha de valores e operadores especiais de manipulação de pilha. Expressões podem ser dadas na **cadeia** associada a uma característica para fazer cálculos com os parâmetros e imprimi-los em determinados formatos. As expressões são dadas em notação polonesa inversa. O caractere '**%**' introduz um operador especial com os significados dados abaixo. Nas explicações, **emp**(x) significa empilhar o valor x, **desemp**() significa desempilhar um valor e usá-lo na expressão.

%% Imprime '%'.
%d Imprime desemp() como número decimal.
%3d Imprime desemp() como número decimal com pelo menos três espaços.
%03d Como %3d mas completa os três caracteres com zeros à esquerda.
%c Imprime desemp() como caractere.
%s Imprime desemp() como cadeia.
%p1 emp(primeiro parâmetro).
...
%p9 emp(nono parâmetro).
%Pa Coloque desemp() na variável a.
%Pb Coloque desemp() na variável b.
...
%Pz Coloque desemp() na variável z.
%ga emp(variável a).
%gb emp(variável b).
...
%gz emp(variável z).
%'c' emp(constante do tipo caractere 'c').
% nn emp(inteiro constante nn).
%l emp(strlen(desemp())).
%+ emp(desemp() + desemp()).
%- emp(desemp() - desemp()).
%* emp(desemp() * desemp()).
%/ emp(desemp() / desemp()).
%m emp(desemp() mod desemp()).
%& emp(desemp() & desemp()).
%| emp(desemp() | desemp()).
% emp(desemp() ^ desemp()).
%~ emp(~desemp()).
%= emp(desemp() == desemp()).
% emp(desemp() desemp()).
% emp(desemp() desemp()).
%! emp(! desemp()).
%i Adiciona 1 aos primeiros dois parâmetros (para posicionamento do cursor em terminais ANSI).
%? expr %t parte-então %e parte-senão %;

Expressão condicional equivalente a

if(expr) then parte-então
else parte-senão

"%e parte-senão" é opcional. A parte senão pode ser uma condição, para construir uma expressão do tipo "senão-se"

Exemplo: Uso de expressões condicionais:

```
%?c1%tb1%ec2%tb2%ec3%tb3%e4
```

Neste caso, c1, c2, e c3 são condições e b1, b2, b3 e b4 são expressões.

Exemplo: Posicionamento do cursor para terminais ANSI:

```
cup=\E[%i%p1%d;%p2%dH
```

Neste caso, os caracteres **ESC** e [são emitidos. Seguem-se o número da linha em decimal, um ponto-e-vírgula, o número da coluna em decimal, e, finalmente, o caractere H. Observe que %i foi usado, porque terminais ANSI numeram linhas e colunas a partir de 1 e não de 0.

Exemplo: Posicionamento de cursor indexado com espaço:

```
cip=\EY%p1%'\s'%+%c%p2%'\s'%+%c
```

Neste caso, os caracteres ESC e Y são emitidos.

Seguem-se um caractere cujo valor é igual ao número da linha adicionado a '\s' e um segundo caractere igual ao número da coluna adicionado a '\s'.

Segue-se a descrição detalhada das características do tipo **cadeia.**

32. **back_tab** (cbt, bt)

    Move o cursor para a esquerda, até a próxima parada de tabulação nesta direção.

33. **bell** (bel, bl)

    Emite um sinal sonoro.

34. **carriage_return** (cr, cr)

    Move o cursor para a primeira coluna da linha na qual ele está atualmente posicionado.

35. **change_scroll_region** (csr, cs)

    #1: linha inicial da região de rolamento
    #2: linha final da região de rolamento

    O terminal possui uma região de rolamento destrutiva, e esta pode ser alterada com **csr** para as linhas #1 a #2. Observe que uma região de rolamento destrutiva deixa a linha final da região em branco, após a combinação de comandos **ri/ind** e/ou a combinação de comandos **il1/dl1.** A posição do cursor não é definida após o comando **csr.**

36. **clear_all_tabs** (tbc, ct)

Remove todos os pontos de tabulação reconhecidos pelo terminal.

37. **clear_screen** (clear, cl)

Apaga a tela do terminal, deixando o cursor na posição (0,0).

38. **clr_eol** (el, ce)

Apaga a linha corrente do terminal, a partir da posição do cursor, até o fim da linha, inclusive. O cursor permanece na sua posição original.

39. **clr_eos** (ed, cd)

Apaga a tela do terminal, a partir da posição atual do cursor, até o fim da mesma, inclusive. O cursor permanece na sua posição original. Este comando só deve ser dado a partir do início de uma linha.

40. **column_address** (hpa, ch)

Move o cursor para a coluna #1, permanecendo na mesma linha.

41. **command_character** (cmdch, CC)

Se o terminal tem um caractere de comando especial configurável, um valor fixo é definido com **cmdch**, e a descrição do terminal deve usar este valor em todas as características relevantes.

42. **cursor_address** (cup, cm)

#1: linha desejada
#2: coluna desejada

Move o cursor para a posição (#1,#2).

43. **cursor_down** (cud1, do)

Move o cursor uma linha para baixo, sem afetar o texto na tela. Se o cursor estiver inicialmente na última linha da tela, este comando tem efeito indefinido.

44. **cursor_home** (home, ho)

Move o cursor para a posição (0,0) da tela. **Home** não deve ser dado, se for simulado com **cup.**

45. **cursor_invisible** (civis, vi)

Torna o cursor invisível. Se **civis** for especificado, **cnorm** também deve sê-lo.

46. **cursor_left** (cub1, le)

Move o cursor uma coluna para a esquerda, sem afetar o texto na tela. Se o cursor estiver inicialmente na primeira coluna da linha, este comando tem efeito indefinido.

47. **cursor_mem_address** (mrcup, CM)

Se o terminal tem a característica **lm**, e o cursor pode ser movido para a posição (#1,#2) relativamente ao início da memória, e não da tela, o comando necessário é dado com **mrcup**.

48. **cursor_normal** (cnorm, ve)

Desfaz os efeitos de **civis** e **cvvis**, deixando o cursor visível, com intensidade normal.

49. **cursor_right** (cuf1, nd)

Move o cursor uma coluna para a direita, sem afetar o texto na tela. Se o cursor estiver inicialmente na última coluna da linha, este comando tem efeito indefinido.

50. **cursor_to_ll** (ll, ll)

Move o cursor para a posição (**lines**-1,0). **Ll** não deve ser dado, se for simulado com **cup**.

51. **cursor_up** (cuu1, up)

Move o cursor uma linha para cima, sem afetar o texto na tela. Se o ursor estiver inicialmente na primeira linha da tela, este comando tem efeito indefinido.

52. **cursor_visible** (cvvis, vs)

Torna o cursor mais visível. Pode ser usado para transformar um cursor do tipo "sublinha" em cursor tipo "quadrado", por exemplo.

53. **delete_character** (dch1, dc)

Remove o caractere sob o cursor, deslocando os caracteres à direita uma posição para a esquerda.

54. **delete_line** (dl1, dl)

Remove a linha atual do cursor, deslocando as linhas abaixo uma posição para cima. Este comando deve ser dado a partir da primeira coluna de uma linha.

55. **dis_status_line** (dsl, ds)

Remove ou apaga a linha de status.

56. **down_half_line** (hd, hd)

Move meia linha para baixo. Normalmente usado para imprimir caracteres subscritos em terminais impressores.

57. **e_e_alt** (eae, EE)

Para terminais ABICOMP (ver **ab**), significa que caracteres acentuados digitados no terminal são entregues ao computador, usando-se seqüências de 7 bits, iniciando com o primeiro caractere de **eae**. Ambos ou nenhum de **eae** e **sae** devem ser especificados. Se **eae** não existe, o terminal entrega caracteres acentuados, usando 8 bits.

58. **enter_alt_charset_mode** (smacs, as)

Usa o conjunto alternativo de caracteres do terminal (por exemplo, grego, semigráfico, etc.).

59. **enter_blink_modec** (blink, mb)

Inicia o modo piscante de atributos de vídeo.

60. **enter_bold_mode** (bold, md)

Inicia o modo intensificado de atributos de vídeo.

61. **enter_ca_mode** (smcup, ti)

Comando de inicialização do terminal para usar comandos de movimentação do cursor.

62. **enter_delete_mode** (smdc, dm)

Comando para entrar no modo de remoção de caracteres. Este comando é enviado para o terminal, antes de **dch1** ou **dch**, para remover caracteres.

63. **enter_dim_mode** (dim, mh)

Inicia o modo atenuado de atributos de vídeo.

64. **enter_insert_mode** (smir, im)

Coloca o terminal no modo de inserção de caracteres. Este comando é enviado para o terminal, antes de **ich1** ou **ich**, se estas características existirem. Poucos terminais usam ambos, **smir** e **ich1**. **Smir** é preferível, quando existe.

65. **enter_protected_mode** (prot, mp)

Inicia o modo protegido de atributos de vídeo.

66. **enter_reverse_mode** (rev, mr)

Inicia o modo inverso de atributos de vídeo.

67. **enter_secure_mode** (invis, mk)

Inicia o modo invisível de atributos de vídeo.

68. **enter_standout_mode** (smso, so)

Inicia o modo de destaque de atributos de vídeo. Este modo corresponde normalmente a vídeo inverso com intensidade atenuada, se possível, ou vídeo inverso puro. Freqüentemente, **smso** é idêntico a **rev**.

69. **enter_underline_mode** (smul, us)

Inicia o modo de sublinhamento de atributos de vídeo.

70. **erase_chars** (ech, ec)

#1: número de caracteres a apagar

Apaga (pela sobreposição de caracteres em branco) #1 caracteres sob e à direita do cursor. A posição do cursor não é alterada.

71. **e_s_alt** (eas, ES)

Para terminais ABICOMP (ver **ab**), significa que caracteres acentuados enviados ao terminal são representados por seqüências de 7 bits, iniciando-se com o primeiro caractere de **eas**. Ambos ou nenhum de **eas** e **sas** devem ser especificados. Se **eas** não existe, caracteres acentuados são entregues usando-se 8 bits.

72. **exit_alt_charset_mode** (rmacs, ae)

Encerra o uso do conjunto alternativo de caracteres, voltando ao conjunto normal (ver smacs).

73. **exit_attribute_mode** (sgr0, me)

Encerra todos os modos de atributos de vídeo. Caracteres adicionados à tela após **sgr0** não receberão atributos de vídeo.

74. **exit_ca_mode** (rmcup, te)

Comando usado para desfazer o efeito de **smcup**.

75. **exit_delete_mode** (rmdc, ed)

Encerra o modo de remoção de caracteres (ver **smdc**). Este comando, se existir, é enviado para o terminal, após **dch1** ou **dch**.

76. **exit_insert_mode** (rmir, ei)

Encerra o modo de inserção de caracteres (ver **smir**). Este comando, se existir, é enviado para o terminal, após **ich1** ou **ich**.

77. **exit_standout_mode** (rmso, se)

Encerra o modo de destaque de atributos de vídeo (ver **smso**).

78. **exit_underline_mode** (rmul, ue)

Encerra o modo de sublinhamento de atributos de vídeo (ver **smul**).

79. **flash_screen** (flash, vb)

Faz a tela piscar. Pode ser usado para substituir **bel**, sem efeito sonoro. A posição do cursor não deve ser alterada com este comando.

80. **form_feed** (ff, ff)

Alimentação de formulário, para terminais impressores.

81. **from_status_line** (fsl, fs)

Movimenta o cursor para a mesma posição em que estava antes do último comando **tsl** dado. Este comando pode ser simulado com **rc**, se necessário.

82. **init_1string** (is1, i1)

Comando de inicialização do terminal. Estes comandos são enviados na seqüência:
iprog is1 is2 tbc hts if is3

83. **init_2string** (is2, is)

Comando de inicialização do terminal. Estes comandos são enviados na seqüência:

iprog is1 is2 tbc hts if is3

84. **init_3string** (is3, i3)

Comando de inicialização do terminal. Estes comandos são enviados na seqüência:

iprog is1 is2 tbc hts if is3

85. **init_file** (if, if)

Nome de um arquivo contendo comandos de inicialização extensos. Os comandos de inicialização são enviados na seqüência:

iprog is1 is2 tbc hts if is3

86. **init_prog** (iprog, iP)

Nome do arquivo contendo um programa a ser executado para inicializar o terminal. Os comandos de inicialização são enviados na seqüência:

iprog is1 is2 tbc hts if is3

87. **insert_character** (ichl, ic)

Comando enviado para o terminal, imediatamente antes do caractere a ser inserido. Normalmente, abre um espaço em branco na posição do cursor. Se **ichl** existir, sempre é enviado após **smir**, se este comando também existir.

88. **insert_line** (il1, al)

Insere uma linha vazia antes da linha onde o cursor está posicionado, deslocando as linhas sob e abaixo do cursor uma posição para baixo. Este comando deve ser dado a partir da primeira coluna da linha, e, após ele, o cursor deve então estar posicionado no início da nova linha vazia.

89. **insert_padding** (ip, ip)

Comando enviado para o terminal, após a inserção de cada caractere no modo de inserção. Freqüentemente, inclui informação de preenchimento.

90. **key_a1** (ka1, K1)

Seqüência de caracteres enviada pela tecla superior esquerda da ilha de nove teclas, se houver.

91. **key_a3** (ka3, K3)

Seqüência de caracteres enviada pela tecla superior direita da ilha de nove teclas, se houver.

92. **key_b2** (kb2, K2)

Seqüência de caracteres enviada pela tecla no centro da ilha de nove teclas, se houver.

93. **key_c1** (kc1, K4)

Seqüência de caracteres enviada pela tecla inferior esquerda da ilha de nove teclas, se houver.

94. **key_c3** (kc3, K5)

Seqüência de caracteres enviada pela tecla inferior direita da ilha de nove teclas, se houver.

95. **key_backspace** (kbs, kb)

Seqüência de caracteres enviada pela tecla de retrocesso.

96. **key_catab** (ktbc, k;)

Seqüência de caracteres enviada pela tecla usada para apagar todas as paradas de tabulação.

97. **key_clear** (kclr, kC)

Seqüência de caracteres enviada pela tecla de apagamento da tela.

98. **key_ctab** (kctab, kt)

Seqüência de caracteres enviada pela tecla usada para apagar a parada de tabulação na coluna da posição do cursor.

99. **key_dc** (kdchl, kD)

Seqüência de caracteres enviada pela tecla de remoção de caractere.

100. **key_dl** (kdl1, kL)

Seqüência de caracteres enviada pela tecla de remoção de linha.

101. **key_down** (kcudl, kd)

Seqüência de caracteres enviada pela tecla usada para movimentar o cursor uma posição para baixo.

102. **key_eic** (krmir, kM)

Seqüência de caracteres enviada pela tecla de encerramento do modo de inserção.

103. **key_eol** (kel, kE)

Seqüência de caracteres enviada pela tecla de apagamento do restante da linha.

104. **key_eos** (ked, kS)

Seqüência de caracteres enviada pela tecla de apagamento do restante da tela.

105. **key_f0** (kf0, k0)

Seqüência de caracteres enviada pela tecla de função F0.

106. **key_f1** (kf1, k1)

Seqüência de caracteres enviada pela tecla de função F1.

107. **key_f2** (kf2, k2)

Seqüência de caracteres enviada pela tecla de função F2.

108. **key_f3** (kf3, k3)

Seqüência de caracteres enviada pela tecla de função F3.

109. **key_f4** (kf4, k4)

Seqüência de caracteres enviada pela tecla de função F4.

110. **key_f5** (kf5, k5)

Seqüência de caracteres enviada pela tecla de função F5.

111. **key_f6** (kf6, k6)

Seqüência de caracteres enviada pela tecla de função F6.

112. **key_f7** (kf7, k7)

Seqüência de caracteres enviada pela tecla de função F7.

113. **key_f8** (kf8, k8)

Seqüência de caracteres enviada pela tecla de função F8.

114. **key_f9** (kf9, k9)

Seqüência de caracteres enviada pela tecla de função F9.

115. **key_f10** (kf10, ka)

Seqüência de caracteres enviada pela tecla de função F10.

116. **key_home** (khome, kh)

Seqüência de caracteres enviada pela tecla de posicionamento do cursor na posição (0,0).

117. **key_ic** (kichl, kI)

Seqüência de caracteres enviada pela tecla de inserção de caracteres ou pela tecla usada para iniciar o modo de inserção.

118. **key_il** (kil1, kA)

Seqüência de caracteres enviada pela tecla de inserção de linha.

119. **key_left** (kcub1, kl)

Seqüência de caracteres enviada pela tecla usada para movimentar o cursor uma posição para a esquerda.

120. **key_ll** (kll, kH)

Seqüência de caracteres enviada pela tecla de posicionamento do cursor na posição (**lines**-1,0).

121. **key_npage** (knp, kN)

Seqüência de caracteres enviada pela tecla de próxima página.

122. **key_ppage** (kpp, kP)

Seqüência de caracteres enviada pela tecla de página anterior.

123. **key_right** (kcuf1, kr)

Seqüência de caracteres enviada pela tecla usada para posicionar o cursor uma posição para a direita.

124. **key_sf** (kind, kF)

Seqüência de caracteres enviada pela tecla de rolamento para baixo.

125. **key_sr** (kri, kR)

Seqüência de caracteres enviada pela tecla de rolamento para cima.

126. **key_stab** (khts, kT)

Seqüência de caracteres enviada pela tecla de definição de parada de tabulação na coluna atual da posição do cursor.

127. **key_up** (kcuul, ku)

Seqüência de caracteres enviada pela tecla usada para posicionar o cursor uma posição para cima.

128. **keypad_local** (rmkx, ke)

Comando para inibir o funcionamento da ilha de teclas.

129. **keypad_xmit** (smkx, ks)

Comando para habilitar o funcionamento da ilha de teclas. Se esta característica não for incluída na descrição, supõe-se que a ilha de teclas, se houver, sempre está ativada.

130. **lab_f0** (lf0, l0)

Nome da tecla F0, se o nome não for "F0".

131. **lab_f1** (lf1, l1)

Nome da tecla F1, se o nome não for "F1".

132. **lab_f2** (lf2, l2)

Nome da tecla F2, se o nome não for "F2".

133. **lab_f3** (lf3, l3)

Nome da tecla F3, se o nome não for "F3".

134. **lab_f4** (lf4, l4)

Nome da tecla F4, se o nome não for "F4".

135. **lab_f5** (lf5, l5)

Nome da tecla F5, se o nome não for "F5".

136. **lab_f6** (lf6, l6)

Nome da tecla F6, se o nome não for "F6".

137. **lab_f7** (lf7, l7)

Nome da tecla F7, se o nome não for "F7".

138. **lab_f8** (lf8, l8)

Nome da tecla F8, se o nome não for "F8".

139. **lab_f9** (lf9, l9)

Nome da tecla F9, se o nome não for "F9".

140. **lab_f10** (lf10, la)

Nome da tecla F10, se o nome não for "F10".

141. **map_ent** (mape, ME)

Nome de um arquivo de mapeamento de caracteres de entrada. Este arquivo, se existir, deve conter 256 octetos, onde o iésimo octeto será o valor retornado pela rotina wgetch da biblioteca CURSES, quando o caractere lido do terminal for **i**.

142. **map_saída** (maps, MS)

Nome de um arquivo de mapeamento de caracteres de saída. Este arquivo, se existir, deve conter 256 octetos, onde o iésimo octeto será o valor enviado ao terminal, quando CURSES tenta enviar o caractere **i** ao terminal.

143. **meta_off** (rmm, mo)

Reservado.

144. **meta_on** (smm, mm)

Reservado.

145. **newline** (nel, nw)

Move o cursor para o início da próxima linha. Se ele estiver posicionado na última linha da tela, o efeito deste comando é indefinido.

146. **pad_char** (pad, pc)

O primeiro caractere desta cadeia é usado para preenchimento, se ASCII NUL não for satisfatório. Para terminais que não podem receber caracteres de preenchimento para implementar os atrasos especificados nas características do terminal, use **pad= A.** Neste caso, o atraso é implementado com um laço de espera. Não há garantia deste laço funcionar corretamente, devido à velocidade desconhecida da UCP. Além do mais, o atraso pode ser muito maior que o desejado, se a UCP estiver carregada.

147. **parm_dch** (dch, DC)

#1: Número de caracteres a remover.

Equivalente a **dch1**, repetido #1 vezes.

148. **parm_delete_line** (dl, DL)

#1: número de linhas a remover.

Equivalente a **dl1**, repetido #1 vezes.

149. **parm_down_cursor** (cud, DO)

#1: Número de posições a movimentar o cursor para baixo.

Equivalente a **cud1**, repetido #1 vezes.

150. **parm_ich** (ich, IC)

#1: Número de espaços em branco a inserir.

Insere #1 espaços em branco, sem afetar a posição do cursor.

151. **parm_index** (indn, SF)

#1: Número de linhas a rolar a tela para cima.

Equivalente a **ind**, repetido #1 vezes.

152. **parm_insert_line** (il, AL)

#1: Número de linhas a inserir.

Equivalente a **il1**, repetido #1 vezes.

153. **parm_left_cursor** (cub, LE)

#1: Número de posições a movimentar o cursor para a esquerda.

Equivalente a **cub1**, repetido #1 vezes.

154. **parm_right_cursor** (cuf, RI)

#1: Número de posições a movimentar o cursor para a direita.

Equivalente a **cuf1**, repetido #1 vezes.

155. **parm_rindex** (rin, SR)

#1: Número de linhas a rolar a tela para baixo.

Equivalente a **ri**, repetido #1 vezes.

156. **parm_up_cursor** (cuu, UP)

#1: Número de posições a movimentar o cursor para cima.

Equivalente a **cuu1**, repetido #1 vezes.

157. **pkey_key** (pfkey, PK)

#1: Número da tecla de função a programar (0 <= #1 <= 10).
#2: Cadeia de programação.

Após este comando, presionar a tecla de função F#1 equivale a digitar a cadeia #2.

158. **pkey_local** (pfloc, pl)

#1: Número da tecla de função a programar (0 <= #1 <= 10).
#2: Cadeia de programação.

Após este comando, pressionar a tecla F#1 causa a execução local da cadeia #2 pelo terminal.

159. **pkey_xmit** (pfx, px)

#1: Número da tecla de função a programar (0 <= #1 <= 10).
#2: Cadeia de programação.

Após este comando, a cadeia #2 é enviada ao computador pelo terminal, ao pressionar a tecla F#1.

160. **print_screen** (mc0, ps)

Para um terminal com módulo impressor auxiliar, causa a impressão do conteúdo da tela.

161. **prtr_non** (mc5p, p0)

#1: Número de octetos transmitidos ao módulo impressor.

Para um terminal com módulo impressor auxiliar, liga o módulo impressor, imprime os próximos #1 caracteres (inclusive um possível mc4) no módulo impressor e o desliga. O texto poderá aparecer ou não na tela do terminal. O valor máximo para o parâmetro #1 é 255.

162. **prtr_off** (mc4, pf)

Para um terminal com módulo impressor auxiliar, desliga o módulo impressor.

163. **prtr_on** (mc5, po)

Para um terminal com módulo impressor auxiliar, liga o módulo impressor. Todo texto enviado ao terminal é impresso no módulo impressor. Este texto poderá aparecer ou não na tela do terminal.

164. **repeat_char** (rep, rp)

#1: Caractere a repetir.
#2: Número de repetições.

Equivalente a enviar o caractere #1, #2 vezes.

165. **reset_1string** (rs1, r1)

Equivalente a is1, mas efetua uma inicialização mais rigorosa do terminal, mesmo que este esteja num estado "estranho".

166. **reset_2string** (rs2, r2)

Equivalente a is2, mas efetua uma inicialização mais rigorosa do terminal, mesmo que este esteja num estado "estranho".

167. **reset_3string** (rs3, r3)

Equivalente a is3, mas efetua uma inicialização mais rigorosa do terminal, mesmo que este esteja num estado "estranho".

?168. **reset_file** (rf, rf)

Equivalente a if, mas efetua uma inicialização mais rigorosa do terminal, mesmo que este esteja num estado "estranho".

169. **restore_cursor** (rc, rc)

Move o cursor para a posição salva com **sc.**

170. **row_address** (vpa, cv)

#1: Linha desejada.

Move o cursor para a linha #1, mantendo a coluna do cursor inalterada.

171. **s_e_alt** (sae, SE)

Para terminais abicomp (ver **ab**), significa que caracteres acentuados digitados no terminal são entregues ao computador, usando-se seqüências de 7 bits terminando com o primeiro caractere de **sae**. Ambos ou nenhum de **eae** e **sae** devem ser especificados. Se **sae** não existe, o terminal entrega caracteres acentuados com 8 bits.

172. **save_cursor** (sc, sc)

Lembra a posição atual do cursor, para uso posterior com o comando **rc.**

173. **scroll_forward** (ind, sf)

Rola a tela uma linha para cima. Este comando só deve ser dado a partir da posição (lines-1,0).

174. **scroll_reverse** (ri, sr)

Rola a tela uma linha para baixo. Este comando só deve ser dado a partir da posição (0,0).

175. **set_attributes** (sgr, sa)

#1: Atributo de destaque.
#2: Atributo de sublinhamento.
#3: Atributo inverso.
#4: Atributo piscante.
#5: Atributo atenuado.
#6: Atributo intensificado.

Este comando coloca o terminal na combinação de atributos indicados pelos parâmetros (Valor igual a zero significa ausência de atributo. Não-zero indica a presença de atributo.). Nem todas a combinações podem ser representadas pelo terminal.

176. **set_tab** (hts, st)

Estabelece um ponto de tabulação reconhecido pelo terminal na coluna atual do cursor.

177. **set_window** (wind, wi)

#1: Linha inicial da janela.
#2: Linha final da janela.
#3: Coluna inicial da janela.
#4: Linha final da janela.

Após este comando, todos os demais comandos da descrição do terminal afetam a janela definida pelos parâmetros #1 a #4.

178. **s_s_alt** (sas, SS)

Para terminais abicomp (ver **ab**), significa que caracteres acentuados enviados ao terminal são representados por seqüências de 7 bits terminando com o primeiro caractere de **sas**. Ambos ou nenhum de **eas** e **sas** devem ser especificados. Se **sas** não existe, caracteres acentuados são entregues, usando-se 8 bits.

179. **tab** (ht, ta)

Para um terminal com pontos de tabulação implementados em hardware, **ht** é o comando para avançar até a próxima parada de tabulação.

180. **to_status_line** (tsl, ts)

#1: Coluna desejada.

Move o cursor para a coluna #1 da linha de status. Este comando pode usar a cadeia definida em **sc**, para facilitar a operação **fsl**.

181. **underline_char** (uc, uc)

Sublinha o caractere sob o cursor e avança o cursor uma posição para a direita.

182. **up_half_line** (hu, hu)

Move meia linha para cima. Normalmente usado para imprimir caracteres sobrescritos em terminais impressores.

183. **vers_i_t** (vit, VI)

A versão da descrição deste terminal. Esta cadeia não deve ser enviada ao terminal, servindo apenas para documentação.

Página intencionalmente em branco.

Computadores e Sistemas Brasileiros S/A

MÓDULO 3

Funções das Bibliotecas

CURSES/TERMINFO/TERMCAP

Página intencionalmente em branco.

CAPÍTULO 1

APRESENTANDO AS ROTINAS...

O CURSES consiste em uma coleção de sub-rotinas e funções usadas para a manipulação de terminais, independentemente do tipo de terminal em uso. Tal independência é atingida pelo uso da biblioteca de funções chamada TERMINFO.

Para usar as rotinas do CURSES, um programa deve incluir o arquivo curses.h e deve ser ligado à biblioteca com a opção -lcurses do ligador-carregador.

Os objetos principais manipulados pelas rotinas chamam-se janelas e são áreas retangulares de texto. No programa, tais janelas são criadas com a rotina **newwin**, que retorna um objeto do tipo

JANELA *

representando a janela. De posse de uma janela, várias operações de entrada e saída podem ser efetuadas. As operações de saída afetam a janela em questão, mas a tela do terminal só é atualizada após a chamada à rotina **wrefresh**.

Após a inicialização do CURSES, feita com a rotina **initscr**, passam a existir duas janelas, ambas do tamanho da tela do terminal. A primeira, chamada stdscr, é normalmente usada pelo programa, para definir a imagem da tela. Neste sentido, é a "janela padrão" do CURSES. A segunda, chamada **curscr** (ou tela virtual) representa o que o programa deseja como imagem da tela, além de freqüentemente representar a combinação de várias janelas. O programa normalmente não manipula **curscr** diretamente, embora haja algumas

exceções (ver as rotinas **clearok** e **wrefresh**).

Janelas especiais chamadas janelas dinâmicas podem ser definidas e manipuladas. Normalmente, são maiores que a tela do terminal, e apenas partes das mesmas são exibidas de cada vez (ver rotina **newpad**.

Os caracteres lidos ou colocados em janelas são do tipo **caractere** (definido no arquivo **curses.h**). Este tipo é, na realidade, um short de dois octetos e permite maior flexibilidade na representação de caracteres. Na saída, por exemplo, os caracteres podem representar letras acentuadas (CURSES usa o conjunto de caracteres ABICOMP) e podem possuir atributos de vídeo, tais como: vídeo reverso, intensificado, etc. Na entrada, os caracteres podem representar teclas especiais, que não são definidas no conjunto ABICOMP (por exemplo, a tecla "cursor para baixo", cujo valor é **KEY_DOWN**, definido no arquivo **curses.h**).

Na inicialização do CURSES, o tipo do terminal sendo usado deve ser identificado. Para tanto, o nome do terminal é obtido da variável do ambiente TERM. A descrição deste terminal é procurada no diretório indicado pela variável do ambiente TERMINFO. Se esta variável não existe, a descrição deve estar presente no diretório **/usr/lib/terminfo**.

A convenção que se segue é usada para muitas das rotinas do CURSES. Se o nome da rotina contiver o sufixo w, ela se aplica à janela recebida como argumento. Sem este prefixo, ela se aplica à janela **stdscr**. Com o prefixo mv, o cursor da janela é primeiro movido com a rotina move, e a função correspondente é então chamada e se aplica à janela **stdscr**. Com o prefixo mvw, a rotina wmove é chamada primeiro, e a função desejada se aplica à janela especificada como argumento. Exemplos:

addch(c)
Adiciona o caractere c na janela **stdscr** na posição atual do cursor.

waddch(jan, c)
Adiciona o caractere **c** na janela **jan**, na posição atual do cursor.

mvaddch(y, x, c)
Move o cursor da janela **stdscr** para a posição (y,x) e chama **addch(c)**.

mvwaddch(jan, y, x, c)
Move o cursor da janela **jan** para a posição (y,x) e chama **waddch(jan, c)**.

As variáveis inteiras **LINHAS** e **COLUNAS** são definidas no arquivo **curses.h** e serão preenchidas pela rotina **initscr** com os valores da tela.

As constantes **ERRO** e **OK** têm, respectivamente, os valores 0 e 1.

Página intencionalmente em branco.

CAPÍTULO 2

CLASSIFICANDO AS ROTINAS...

As rotinas descritas no Capítulo 3 podem ser agrupadas como se segue:

ROTINAS GERAIS

endwin	newterm
initscr	set_term

ROTINAS DE SAIDA

addch	mvaddch	waddstr
addstr	mvaddstr	wattroff
attroff	mvdelch	wattron
attron	mvinsch	wattrset
attrset	mvprintw	wclear
beep	mvwaddch	wclrtobot
box	mvwaddstr	wclrtoeol
clear	mvwdelch	wdelch
clrtobot	mvwinsch	wdeleteln
clrtoeol	mvwprintw	werase
delay_output	nscroll	winsch
delch	overlay	winsertln
deleteln	overwrite	wmove
erase	printw	wnscroll
flash	scroll	wprintw
getyx	standend	wstandend
insch	standout	wstandout
insertln	touchwin	
move	waddch	

ROTINAS DE MANIPULAÇÃO DE JANELAS

delwin	newwin	subwin
doupdate	pnoutrefresh	wnoutrefresh
mvwin	prefresh	wrefresh
newpad	refresh	

ROTINAS DE ENTRADA

flushinp	mvinch	scanw
getch	mvscanw	tem_pendencia
getstr	mvwgetch	wgetch
inch	mvwgetstr	wgetstr
mvgetch	mvwinch	winch
mvgetstr	mvwscanw	wscanw

OPÇOES DE SAIDA

acent	modoabc	setscrreg
clearok	nl	wacent
idlok	nonl	wmodoabc
leaveok	scrollok	wsetscrreg

OPÇÕES DE ENTRADA

cbreak	nocbreak	reset_shell_mode
def_prog_mode	nodelay	resetterm
def_shell_mode	noecho	resetty
echo	noraw	saveterm
fixterm	noxon	savetty
intrflush	raw	typeahead
keypad	reset_prog_mode	xon

VALORES DO AMBIENTE

baudrate	has_ic	killchar
erasechar	has_il	longname

ROTINAS DO TERMINFO

mvcur	setupterm	tputs
putp	tic	vidattr
setterm	tparm	vidputs

ROTINAS DE COMPATIBILIDADE TERMCAP

tgetent	tgetnum	tgoto
tgetflag	tgetstr	

MISCELÂNEA

desacent	gettmode	unctrl

Página intencionalmente em branco.

CAPÍTULO 3

DESCREVENDO AS ROTINAS...

3.1 ROTINAS DA BIBLIOTECA CURSES

Como abordado anteriormente, para usar as rotinas do CURSES, um programa deve incluir o arquivo **curses.h** e ser ligado à biblioteca com a opção **-lcurses** do ligador-carregador.

As rotinas desta biblioteca são descritas a seguir.

Página intencionalmente em branco.

BAUDRATE

SINOPSE

```
int
baudrate()
```

DESCRIÇÃO SUCINTA

A rotina **baudrate** retorna a velocidade de transmissão do terminal. O valor retornado é em bits por segundo.

Página intencionalmente em branco.

BEEP

SINOPSE

```
int
beep()
```

DESCRIÇÃO SUCINTA

A rotina **beep** emite um sinal sonoro pela campainha do terminal.

CONSIDERAÇÕES ESPECIAIS

1. Se o terminal não possui este sinal sonoro (ver a descrição do terminal no TERMINFO), **beep** provoca uma piscada na tela do terminal (ver a rotina **flash**), se possível.

VALOR DE RETORNO

Se o terminal não possui um sinal sonoro e não tem a capacidade de piscar a tela, a rotina retorna ERRO. Caso contrário, retorna **OK**.

Página intencionalmente em branco.

BOX

SINOPSE

```
void
box(jan, car_vert, car_hor)
JANELA    *jan;
caractere car_vert, car_hor;
```

DESCRIÇÃO SUCINTA

A rotina box desenha um retângulo ao redor das bordas internas da janela **jan**. As linhas horizontais do retângulo são preenchidas com o caractere **car_hor**. As linhas verticais do retângulo são preenchidas com o caractere **car_vert**.

CONSIDERAÇÕES ESPECIAIS

1. Se o argumento **car_vert** for igual a zero, o caractere default '|' é assumido.

2. Se o argumento **car_hor** for igual a zero, o caractere default '-' é assumido.

Página intencionalmente em branco.

CBREAK, NOCBREAK

SINOPSE

void
cbreak()

void
no**cbreak**()

DESCRIÇÃO SUCINTA

A rotina **cbreak** (**nocbreak**) liga (desliga) o modo **cbreak** do terminal.

CONSIDERAÇÕES ESPECIAIS

1. Se o modo **cbreak** estiver desligado, os caracteres teclados no terminal só são entregues ao programa com a digitação de um caractere de nova-linha ou de retorno-de-carro (entrada sincronizada a linha).

2. Se o modo **cbreak** estiver ligado, os caracteres teclados no terminal são entregues imediatamente ao programa, sem a necessidade de se digitar um caractere de nova-linha ou de retorno-de-carro (entrada sincronizada a caractere). Os caracteres de interrupção e de controle de fluxo são tratados pelo sistema operacional. Os caracteres de edição de linha não são tratados pelo sistema operacional.

3. Se nenhuma das duas rotinas forem chamadas no programa, nenhum valor default para o modo **cbreak** é assumido (o modo existente quando da chamada a **initscr** permanece).

CLEAROK

SINOPSE

```
void
clearok( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

A rotina **clearok** permite (se **indicador** for igual a **TRUE**) ou proíbe (se **indicador** for igual a **FALSE**) a emissão de um código de controle para apagar a tela do terminal quando da próxima chamada **wrefresh** com parâmetro **jan**. Se **indicador** for igual a **TRUE**, a próxima chamada a **wrefresh** com parâmetro **jan** apagará a tela e a reexibirá por inteiro. Se **indicador** for igual a **FALSE**, estas duas ações não são realizadas. O conteúdo da janela **jan** ou da tela não é alterado com esta rotina.

CONSIDERAÇÕES ESPECIAIS

1. Se **jan** for igual a **curscr**, a próxima chamada a **wrefresh** (para qualquer janela) apagará e reexibirá a tela inteira.

Página intencionalmente em branco.

DEF_PROG_MODE, SAVETERM

SINOPSE

```
void
def_prog_mode()

void
saveterm()
```

DESCRIÇÃO SUCINTA

A rotina **def_prog_mode** guarda o estado corrente do terminal. Este estado é dito ser o estado "programa", isto é, o estado do terminal enquanto o programa quer usar as rotinas do CURSES.

CONSIDERAÇÕES ESPECIAIS

1. O estado do terminal é guardado para uso posterior pela rotina **reset_prog_mode**.

2. A rotina **def_prog_mode** é chamada automaticamente pela rotina **initscr**, assim que esta rotina tiver alterado o estado do terminal necessário para o funcionamento do CURSES.

3. A rotina **saveterm** funciona de forma idêntica a **def_prog_mode**.

Página intencionalmente em branco.

DEF_SHELL_MODE

SINOPSE

```
void
def_shell_mode()
```

DESCRIÇÃO SUCINTA

A rotina **def_shell_mode** guarda o estado corrente do terminal. Este estado é dito ser o estado "shell", isto é, o estado do terminal enquanto o programa quer usar o terminal, sem passar pelas rotinas do CURSES.

CONSIDERAÇÕES ESPECIAIS

1. A rotina **def_shell_mode** é chamada automaticamente pela rotina **initscr**, antes de mudar o estado do terminal.

2. O estado do terminal é guardado para uso posterior pela rotina **reset_shell_mode**.

Página intencionalmente em branco.

DELAY_OUTPUT

SINOPSE

```
int
delay_output( ms )
int  ms;
```

DESCRIÇÃO SUCINTA

A rotina **delay_output** gera na saída uma pausa de milissegundos.

CONSIDERAÇÕES ESPECIAIS

1. A pausa é implementada pelo envio de caracteres de preenchimento ao terminal (ver **pad_char** no TERMINFO).

2. A pausa gerada pode ser maior que o valor especificado. A resolução é igual ao tempo necessário para enviar um caractere à velocidade de transmissão corrente.

VALOR DE RETORNO

Retorna **OK**, quando a velocidade do terminal (em baud) é uma das seguintes:

50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200

Caso contrário, retorna ERRO.

DELWIN

SINOPSE

```
void
delwin(jan)
JANELA    *jan;
```

DESCRIÇÃO SUCINTA

A rotina **delwin** destrói a janela **jan**, liberando o espaço ocupado da memória. A janela não pode mais ser usada.

CONSIDERAÇÕES ESPECIAIS

1. Se a janela **jan** for uma subjanela, delwin libera o espaço da memória referente à estrutura da janela **jan**, porém não libera o espaço compartilhado com a janela na qual a subjanela está contida.

2. Ao se chamar **delwin**, as subjanelas ligadas à janela **jan** (se existirem) têm as suas áreas de imagem na tela indefinidas e não devem ser usadas. Tais subjanelas deveriam ser removidas com **delwin**, antes da janela na qual estão contidas.

Página intencionalmente em branco.

DESACENT

SINOPSE

```
int
desacent( carac )
caractere carac;
```

DESCRIÇÃO SUCINTA

A rotina **desacent** retorna como valor a letra desacentuada equivalente ao parâmetro **carac**. Por exemplo, se **carac** for igual a **ã**, a rotina **desacent** retorna **a**.

Página intencionalmente em branco.

DOUPDATE

SINOPSE

```
int
doupdate()
```

DESCRIÇÃO SUCINTA

A rotina **doupdate** realiza uma atualização da tela do terminal. CURSES usa uma janela especial, chamada **curscr** para representar o que se deseja como imagem na tela. **Doupdate** atualiza a tela do terminal de acordo com a imagem presente em **curscr**, de forma otimizada, isto é, com o mínimo de atualizações à tela. No fim da rotina **doupdate**, a tela é idêntica à imagem de **curscr**.

Esta rotina é normalmente usada em conjunto com as rotinas **wnoutrefresh** ou **pnoutrefresh**.

CONSIDERAÇÕES ESPECIAIS

1. A posição do cursor após a atualização da tela depende da última chamada à rotina **leaveok** (ver a rotina **leaveok**).

2. **Doupdate** só realiza a atualização da tela do terminal, se não houver caracteres pendentes na entrada (ver a rotina **typeahead**).

VALOR DE RETORNO

Doupdate sempre retorna o valor **OK.**

ECHO, NOECHO

SINOPSE

```
void
echo()

void
noecho()
```

DESCRIÇÃO SUCINTA

A rotina **echo** (noecho) habilita (desabilita) o eco de caracteres lidos do teclado (com as rotinas **wgetch**, **wgetstr** e **wscanw**). Se o eco estiver ligado, os caracterés lidos são ecoados na janela de leitura usando **waddch**, e a rotina **wrefresh** é chamada. Os caracteres nunca são ecoados diretamente na tela pelo controlador de terminal do sistema operacional.

CONSIDERAÇÕES ESPECIAIS

1. Se nenhuma das duas rotinas for chamada, CURSES funciona com eco ligado.

2. Teclas especiais (ver KEY... no TERMINFO) nunca são ecoadas.

Página intencionalmente em branco.

ENDWIN

SINOPSE

```
int
endwin()
```

DESCRIÇÃO SUCINTA

A rotina **endwin** deve ser chamada, ao terminar o programa, ou para, temporariamente, sair do CURSES. **Endwin** restaura o estado original do terminal corrente, usando a rotina **reset_shell_mode**.

CONSIDERAÇÕES ESPECIAIS

1. **Endwin** posiciona o cursor no vértice inferior esquerdo da tela.

2. **Endwin** não libera os espaços alocados para as janelas ou qualquer outro espaço alocado pelo CURSES.

3. Para voltar a manusear janelas usando o CURSES, as rotinas **wrefresh** ou **doupdate** podem ser chamadas.

4. **Endwin** deve ser chamada para cada terminal criado (ver a rotina **newterm**).

VALOR DE RETORNO

A rotina **endwin** normalmente retorna o valor **OK.** Em caso de erro, o valor de retorno é ERRO. A condição de erro é a seguinte:

1. **Endwin** foi chamada enquanto o terminal está no estado "shell" (ver as rotinas **reset_prog_mode** e **reset_shell_mode.**

ERASECHAR

SINOPSE

```
int
erasechar()
```

DESCRIÇÃO SUCINTA

A rotina **erasechar** retorna o caractere corrente de apagamento de caractere (ver ERASE na descrição do controlador de terminal do sistema operacional).

Página intencionalmente em branco.

FLASH

SINOPSE

```
int
flash()
```

DESCRIÇÃO SUCINTA

A rotina **flash** provoca uma piscada na tela do terminal.

CONSIDERAÇÕES ESPECIAIS

1. Se o terminal não pode realizar uma piscada na tela (ver a descrição do terminal no TERMINFO), **flash** faz o terminal emitir um sinal sonoro pela campainha, se possível (ver a rotina beep).

VALOR DE RETORNO

Se o terminal não pode provocar uma piscada na tela e não pode emitir um sinal sonoro, a rotina retorna **ERRO**. Caso contrário, retorna **OK**.

Página intencionalmente em branco.

FLUSHINP

SINOPSE

```
void
flushinp()
```

DESCRIÇÃO SUCINTA

A rotina **flushinp** esvazia a fila de entrada de caracteres digitados, mas ainda não lidos pelo programa.

Página intencionalmente em branco.

GETTMODE

SINOPSE

```
void
gettmode()
```

DESCRIÇÃO SUCINTA

A rotina **gettmode** não realiza nenhuma operação.

CONSIDERAÇÕES ESPECIAIS

1. A rotina **gettmode** existe por questão de compatibilidade com aplicações já existentes, não sendo, entretanto, necessária para novos desenvolvimentos.

Página intencionalmente em branco.

GETYX

SINOPSE

```
#define getyx( jan, y, x)       ...
JANELA    *jan;
int       y, x;
```

DESCRIÇÃO SUCINTA

Getyx é uma macro (definida no arquivo **curses.h**) que atribui às variáveis **y** e **x** o valor da linha e da coluna, respectivamente, da posição do cursor na janela **jan**. A posição do cursor na janela é relativa ao início da janela (vértice superior esquerdo, posição (0, 0)).

CONSIDERAÇÕES ESPECIAIS

1. Sendo **getyx** uma macro, **y** e **x** devem ser variáveis inteiras e não endereços de variáveis inteiras.

Página intencionalmente em branco.

HAS_IC

SINOPSE

```
int
has_ic()
```

DESCRIÇÃO SUCINTA

A rotina **has_ic** retorna **TRUE**, se o terminal é dotado do recurso de inserir e deletar caracteres na tela. Caso contrário, retorna **FALSE** (ver TERMINFO).

CONSIDERAÇÕES ESPECIAIS

1. Na atualização da tela do terminal pela rotina **doupdate**, a habilidade do terminal de inserir ou deletar caracteres não é usada.

Página intencionalmente em branco.

HAS_IL

SINOPSE

```
int
has_il()
```

DESCRIÇÃO SUCINTA

A rotina **has_il** retorna **TRUE**, se o terminal é dotado do recurso de inserir e deletar linhas na tela, ou pode simular esta habilidade, usando rolamento de regiões da tela. Caso contrário, retorna **FALSE** (ver TERMINFO).

Página intencionalmente em branco.

IDLOK

SINOPSE

```
int
idlok( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

A rotina **idlok** informa ao CURSES se ele deve (**indicador** igual a **TRUE**) ou não (**indicador** igual a **FALSE**) usar os recursos de hardware do terminal, para inserir ou remover linhas, e/ou para rolar regiões da tela.

VALOR DE RETORNO

Idlok sempre retorna o valor **OK.**

Página intencionalmente em branco.

INITSCR

SINOPSE

```
void
initscr()
```

DESCRIÇÃO SUCINTA

A rotina **initscr** deve ser a primeira rotina a ser chamada, quando se quer usar o CURSES. Initscr aloca todas as estruturas de dados e lê as características do terminal cadastradas no TERMINFO.

CONSIDERACOES ESPECIAIS

1. Se o terminal for de um tipo desconhecido pelo TERMINFO, uma mensagem de erro é impressa na saída padrão de erro, e o programa é encerrado.

2. As janelas **stdscr** (tela-padrão) e **curscr** (tela corrente) são criadas pela rotina **initscr**.

3. A primeira vez que a rotina **wrefresh** for chamada após a chamada de **initscr**, a tela do terminal é apagada.

4. Se **initscr** não conseguir alocar espaço para as estruturas do CURSES, uma mensagem de erro é impressa na saída padrão de erro, e o programa é encerrado.

5. Se o programa quiser obter um retorno sobre as condições de erro sem gerar uma mensagem de erro, ele deve usar a rotina **newterm**, ao invés de **initscr**.

A rotina **newterm** deve também ser usada no lugar de **initscr**, no caso de programas que utilizam vários terminais.

INTRFLUSH

SINOPSE

```
void
intrflush( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

A rotina **intrflush** habilita (se **indicador** for igual a **TRUE**) ou desabilita (se **indicador** for igual a **FALSE**) o esvaziamento da fila de saída do terminal, quando uma tecla de interrupção é digitada.

CONSIDERAÇÕES ESPECIAIS

1. A habilitação do esvaziamento faz com que CURSES perca noção do que realmente está na tela. A habilitação do escoamento da fila de saída agiliza o processamento de interrupção.

2. O argumento **jan** é ignorado.

3. Se a rotina **intrflush** não for chamada, o escoamento será feito ou não, dependendo do estado inicial do terminal, antes de chamar **initscr**.

Página intencionalmente em branco.

KEYPAD

SINOPSE

```
void
keypad( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇAO SUCINTA

A rotina **keypad** habilita (se **indicador** igual a **TRUE**), ou desabilita (se **indicador** igual a **FALSE**), na janela jan, o uso de teclas-função (ver definições das teclas_função em curses.h), quando forem digitadas no terminal.

CONSIDERAÇOES ESPECIAIS

1. CURSES só trata as teclas-função do terminal que estiverem cadastradas na descrição do terminal no TERMINFO.

Página intencionalmente em branco.

KILLCHAR

SINOPSE

```
int
killchar()
```

DESCRIÇÃO SUCINTA

A rotina **killchar** retorna o caractere corrente de apagamento de linha (ver KILL na descrição do controlador de terminal do sistema operacional).

Página intencionalmente em branco.

LEAVEOK

SINOPSE

```
void
leaveok( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

A rotina **leaveok** deixa o cursor da tela na posição do cursor da última janela que chamou **wrefresh**, **wnoutrefresh**, **prefresh** ou **pnoutfresh** (se **indicador** for igual a **FALSE** para esta janela), ou em posição indefinida (se **indicador** for igual a **TRUE**).

CONSIDERAÇÕES ESPECIAIS

1. Se **indicador** for igual a **TRUE**, CURSES tenta fazer com que o cursor se torne invisível, se a janela **jan** for a última sujeita à rotina **wnoutrefresh** ou **pnoutrefresh**, antes da chamada a **doupdate**.

2. CURSES assume inicialmente que **leaveok** é **FALSE** para todas as janelas criadas.

Página intencionalmente em branco.

LONGNAME

SINOPSE

```
char *
longname()
```

DESCRIÇÃO SUCINTA

A rotina **longname** retorna o nome completo do terminal corrente. Este nome é descrito no TERMINFO.

CONSIDERAÇÕES ESPECIAIS

1. A cada chamada da rotina **newterm**, este nome é alterado, porém não é restaurado com a chamada da rotina set_term.

2. O tamanho máximo do nome do terminal é de 128 caracteres.

VALOR DE RETORNO

Longname retorna um apontador para uma área estática contendo o nome do terminal.

Página intencionalmente em branco.

MACROS MV

SINOPSE

```
#define mvwaddch(jan,y,x,c) (wmove(jan,y,x)\
 == ERRO ?  ERRO:waddch(jan,c))

#define mvwaddstr(jan,y,x,str) (wmove(jan,y,x)\
 == ERRO ? ERRO:waddstr(jan,str))

#define mvwdelch(jan,y,x) (wmove(jan,y,x)\
 == ERRO ?  ERRO:wdelch(jan))

#define mvwgetch(jan,y,x) (wmove(jan,y,x)\
 == ERRO ? ERRO:wgetch(jan))

#define mvwgetstr(jan,y,x,str) (wmove(jan,y,x)\
 == ERRO ? ERRO:wgetstr(jan,str))

#define mvwinch(jan,y,x) (wmove(jan,y,x)\
 == ERRO ? ERRO:winch(jan))

#define mvwinsch(jan,y,x,c) (wmove(jan,y,x)\
 == ERRO ? ERRO:winsch(jan,c))

#define mvaddch(y,x,c) mvwaddch(stdscr,y,x,c)

#define mvaddstr(y,x,str) mvwaddstr(stdscr,y,x,str)

#define mvdelch(y,x) mvwdelch(stdscr,y,x)

#define mvgetch(y,x) mvwgetch(stdscr,y,x)

#define mvgetstr(y,x) mvwgetstr(stdscr,y,x,str)
```

```
#define mvinch(y,x) mvwinch(stdscr,y,x)

#define mvinsch(y,x,c) mvwinsch(stdscr,y,x,c)
```

DESCRIÇÃO SUCINTA

Cada uma das macros move o cursor utilizando a rotina wmove e realiza a ação a ela correspondente.

As macros acima estão definidas no arquivo **curses.h.**

VALOR DE RETORNO

As macros normalmente retornam **OK.** Em caso de erro, o valor retornado é ERRO. Os casos de erro são os mesmos da rotina **wmove** ou da rotina correspondente à macro usada.

MVWIN

SINOPSE

```
int
mvwin(jan, y, x)
JANELA    *jan;
int       y, x;
```

DESCRIÇÃO SUCINTA

A rotina **mvwin** move a janela **jan** de forma a deixar seu vértice superior esquerdo na posição: linha = y, coluna = x, relativa à tela. A posição do cursor não é alterada na janela **jan**.

CONSIDERAÇÕES ESPECIAIS

1. O movimento só ocorre se a nova posição da janela não ultrapassar os limites da tela.

2. Se a janela **jan** for uma subjanela, mvwin move a subjanela, porém ela continua compartilhando a mesma parte do texto com a janela na qual está contida.

VALOR DE RETORNO

Normalmente mvwin retorna o código **OK**. Em caso de erro (ver consideração 1), o valor retornado é ERRO.

Página intencionalmente em branco.

NEWPAD

SINOPSE

```
JANELA  *
newpad( nlinhas, ncolunas)
int nlinhas, ncolunas;
```

DESCRIÇÃO SUCINTA

A rotina **newpad** cria uma janela, alocando espaço da memória. A janela criada, que é do tipo janela dinâmica, ou **pad**, não está associada com nenhuma parte da tela, logo poderá ter tamanhos maiores que uma janela normal. A janela dinâmica criada tem **nlinhas** linhas e **ncolunas** colunas. Inicialmente, a janela dinâmica estará preenchida com caracteres brancos. Seu cursor estará posicionado em (0, 0) (vértice superior esquerdo).

CONSIDERAÇÕES ESPECIAIS

1. Normalmente a rotina **newpad** é usada, quando uma janela maior que a tela é necessária, e quando apenas partes da janela serão exibidas na tela, de cada vez.

2. Atualizações automáticas da tela (por exemplo, quando o modo eco está ligado) não ocorrem com janelas dinâmicas.

3. Atualizações de uma parte de uma janela dinâmica na tela são feitas com as rotinas **prefresh** ou **pnoutrefresh**, e não com as rotinas **wrefresh** e **wnoutrefresh** (ver as rotinas prefresh e pnoutrefresh).

VALOR DE RETORNO

Normalmente, newpad retorna um apontador para a janela dinâmica. Em caso de erro, o valor retornado é **NULL**. Os casos de erro são os seguintes:

1. O número de linhas **nlinhas** ou de colunas **ncolunas** é menor ou igual a zero.

2. Não há espaço disponível na memória para a alocação da estrutura da janela dinâmica.

NEWTERM

SINOPSE

```
TELA *
newterm( nome_term, arq_saida, arq_entrada)
char       *nome_term;
FILE       *arq_saida, *arq_entrada;
```

DESCRIÇAO SUCINTA

A rotina **newterm** cria e inicializa um novo terminal para uso pelo programa. **Newterm** lê as características do terminal **nome_term** descritas no TERMINFO. A saída do terminal criado é dada pelo argumento **arq_saída**, e sua entrada pelo argumento **arq_entrada**.

CONSIDERAÇOES ESPECIAIS

1. O CURSES manipula um terminal de cada vez. A rotina set_term é usada para mudar de terminal (ver a rotina set_term).

2. **Newterm** cria novas janelas **stdscr** e **curscr** para o novo terminal.

3. A rotina **endwin** deve ser chamada para cada terminal, ao encerrar operações com CURSES.

VALOR DE RETORNO

Normalmente, **newterm** retorna um apontador para o novo terminal, que deve ser guardado para uso na rotina **set_term**. Em caso de erro, o valor retornado é NULL. Os casos de erro são os seguintes:

1. O terminal do tipo **nome_term** não está cadastrado no TERMINFO.

2. Não há espaço na memória para alocar as estruturas do novo terminal.

NEWWIN

SINOPSE

```
JANELA *
newwin( nlinhas, ncolunas, y, x)
int nlinhas, ncolunas, y, x;
```

DESCRIÇÃO SUCINTA

A rotina **newwin** cria uma nova janela. A nova janela tem **nlinhas** linhas e **ncolunas** colunas. A posição inicial da janela (vértice superior esquerdo) é: linha = y, coluna = x, relativamente à tela. A janela, inicialmente, está preenchida com caracteres brancos. O cursor começa na posição (0, 0) da janela.

CONSIDERAÇÕES ESPECIAIS

1. Se na chamada de **newwin, nlinhas** for igual a zero, newwin assume que o número de linhas da janela é LINHAS - y.

2. Se na chamada de **newwin, ncolunas** for igual a zero, newwin assume que o número de colunas da janela é COLUNAS - x.

3. A posição inicial da janela (linha = y, coluna = x) pode estar localizada em qualquer posição relativa à tela (até em uma posição negativa), desde que o vértice inferior direito não ultrapasse a posição (LINHAS - 1, COLUNAS - 1), relativa à tela.

VALOR DE RETORNO

Newwin normalmente retorna um apontador para a nova janela. Em caso de erro, o valor retornado é **NULL**. As condições de erro são as seguintes:

1. Número negativo de linhas ou de colunas.

2. O vértice inferior direito da janela x posicionado abaixo da última linha ou à direita da última coluna da tela.

3. Não há espaço disponível na memória para as estruturas alocadas por **newwin** para a janela.

NL, NONL

SINOPSE

```
void
nl()

void
nonl()
```

DESCRIÇÃO SUCINTA

A rotina **nl** (**nonl**) liga (desliga) o modo "nova-linha" do terminal.

CONSIDERAÇÕES ESPECIAIS

1. Se o modo "nova-linha" estiver ligado, o terminal traduz na saída um caractere "nova-linha" para a seqüência de caracteres "retorno-de-carro" e "nova-linha", como também traduz na entrada um caractere "retorno-de-carro" para o caractere "nova-linha".

2. Se o modo "nova-linha" estiver desligado, os caracteres "nova-linha" e "retorno-de-carro" não são traduzidos.

3. A tradução é feita ao nível do controlador de terminal e não a nível do CURSES.

4. Se nenhuma das duas rotinas forem chamadas no programa, CURSES assume que o modo "nova-linha" está ligado.

NODELAY

SINOPSE

```
void
nodelay( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

A rotina **nodelay** informa a CURSES se a rotina wgetch deve funcionar de forma bloqueada (**indicador** igual a **FALSE**) ou não-bloqueada (indicador igual a **TRUE**).

CONSIDERAÇÕES ESPECIAIS

1. Se o valor de **indicador** for **TRUE**, a rotina **wgetch** verifica se há caracteres pendentes na fila de entrada. Se não houver, ela não espera que um caractere seja teclado e retorna ERRO.

2. Se o valor de **indicador** for **FALSE**, a rotina **wgetch** espera que um caractere seja teclado no terminal, quando a fila de entrada estiver vazia.

3. CURSES assume inicialmente que **nodelay** é **FALSE** para todas as janelas.

Página intencionalmente em branco.

OVERLAY, OVERWRITE

SINOPSE

```
void
overlay( jan1, jan2)
JANELA    *jan1, *jan2;

void
overwrite( jan1, jan2)
JANELA    *jan1, *jan2;
```

DESCRIÇÃO SUCINTA

As rotinas **overlay** e **overwrite** copiam o conteúdo da janela **jan1** na janela **jan2**. O caractere da posição (y, x) da janela **jan1** é copiado somente se esta posição também existir na janela **jan2**.

CONSIDERAÇÕES ESPECIAIS

1. Na rotina **overlay**, os espaços em branco da janela **jan1** não são copiados para a janela **jan2**. O conteúdo da janela **jan2** continuará inalterado nas posições correspondente a espaços em branco na janela **jan1**.

Página intencionalmente em branco.

PNOUTREFRESH

SINOPSE

```
int
pnoutrefresh( jandin, linha_pad, coluna_pad,
              pr_lin_tela, pr_col_tela,
              ult_lin_tela, ult_col_tela )
JANELA  *jandin;
int     linha_pad, coluna_pad;
int     pr_lin_tela,   pr_col_tela,   ult_lin_tela,
        ult_col_tela;
```

DESCRIÇAO SUCINTA

A rotina **pnoutrefresh** é similar à rotina **wnoutrefresh**, mas opera com janelas dinâmicas. Os argumentos da rotina **pnoutrefresh** têm o mesmo significado da rotina **prefresh**.

VALOR DE RETORNO

Pnoutrefresh sempre retorna **OK**.

Página intencionalmente em branco.

PREFRESH

SINOPSE

```
int
prefresh( jandin, linha_jan, coluna_jan,
          pr_lin_tela, pr_col_tela,
          ult_lin_tela, ult_col_tela )
JANELA    *jandin;
int       linha_jan, coluna_jan;
int       pr_lin_tela,  pr_col_tela,  ult_lin_tela,
          ult_col_tela;
```

DESCRIÇÃO SUCINTA

A rotina **prefresh** é similar à rotina **wrefresh**, mas manipula janelas dinâmicas. Os argumentos **linha_jan** e **coluna_jan** indicam o ponto inicial da janela jandin (a linha e a coluna da janela dinâmica, respectivamente) que aparecerá na tela do terminal, a partir da linha **pr_lin_tela** até a linha **ult_lin_tela** da tela, e da coluna **pr_col_tela** até a coluna **ult_col_tela** da tela.

CONSIDERAÇÕES ESPECIAIS

1. Se **linha_jan** ou **coluna_jan** ou **pr_lin_tela** ou **pr_col_tela** tiverem valores negativos, CURSES assume o valor zero para a(s) mesma(s).

2. Se **ult_lin_tela** ou **ult_col_tela** tiverem valores maiores que o número de linhas ou de colunas da tela, respectivamente, CURSES assume que seus valores são iguais à última linha ou coluna da tela.

3. Se o número de linhas entre **pr_lin_tela** e **ult_lin_tela** for maior que o número de linhas entre **linha_jan** e a última linha da janela dinâmica, CURSES assume que o número de linhas entre **pr_lin_tela** e **ult_lin_tela** é igual ao número de linhas entre **linha_jan** e a última linha da janela.

VALOR DE RETORNO

Prefresh sempre retorna **OK.**

RAW, NORAW

SINOPSE

```
void
raw()

void
noraw()
```

DESCRIÇÃO SUCINTA

A rotina **raw** (**noraw**) liga (desliga) o modo **raw** do terminal.

CONSIDERAÇÕES ESPECIAIS

1. O modo **raw** é semelhante ao modo **cbreak** (ver a rotina **cbreak**). A diferença reside no fato de que, no modo **raw**, os caracteres de controle de fluxo e de interrupção não são tratados pelo sistema operacional e são entregues ao programa diretamente.

2. Se nenhuma das duas rotinas for chamada no programa, nenhum valor default para o modo **raw** é assumido (permanece o modo existente no controlador de terminal, quando da chamada a **initscr**).

Página intencionalmente em branco.

RESET_PROG_MODE, FIXTERM

SINOPSE

```
void
reset_prog_mode()

void
fixterm()
```

DESCRIÇAO SUCINTA

A rotina **reset_prog_mode** restaura o estado do terminal guardado anteriormente pela rotina **def_prog_mode** (ver a rotina **def_prog_mode**).

CONSIDERAÇOES ESPECIAIS

1. A rotina **reset_prog_mode** é chamada automaticamente pela rotina **doupdate**, após uma chamada da rotina **endwin**.

2. A rotina **fixterm** funciona de forma idêntica a **reset_prog_mode**.

Página intencionalmente em branco.

RESET_SHELL_MODE, RESETTERM

SINOPSE

```
void
reset_shell_mode()

void
resetterm()
```

DESCRIÇÃO SUCINTA

A rotina **reset_shell_mode** restaura o estado do terminal guardado anteriormente pela rotina **def_shell_mode** (ver a rotina **reset_shell_mode**.

CONSIDERAÇÕES ESPECIAIS

1. A rotina é chamada automaticamente pela rotina **endwin**.

2. A rotina **resetterm** funciona de forma idêntica a **reset_shell_mode**.

Página intencionalmente em branco.

RESETTY

SINOPSE

```
void
resetty()
```

DESCRIÇÃO SUCINTA

A rotina **resetty** recupera o estado do terminal salvo pela última chamada à rotina **savetty**.

CONSIDERAÇÕES ESPECIAIS

1. A chamada à rotina **resetty** tem efeito nulo, se a rotina **savetty** não tiver sido chamada anteriormente.

Página intencionalmente em branco.

SAVETTY

SINOPSE

```
void
savetty()
```

DESCRIÇÃO SUCINTA

A rotina **savetty** guarda o estado corrente do terminal, para uso posterior pela rotina **resetty**.

Página intencionalmente em branco.

SCROLL

SINOPSE

```
void
scroll (jan)
JANELA *jan;
```

DESCRIÇÃO SUCINTA

A janela é rolada automaticamente uma linha para cima.

Página intencionalmente em branco.

SCROLLOK

SINOPSE

```
void
scrollok( jan, indicador)
JANELA    *jan;
bool      indicador;
```

DESCRIÇÃO SUCINTA

Com o indicador igual a **TRUE**, a região de rolamento da janela **jan** é rolada automaticamente, uma linha para cima, sempre que o caractere de nova-linha for adicionado à janela, com o cursor da janela presente na última linha da região de rolamento, ou quando um caractere é adicionado à janela, com o cursor da janela presente no canto inferior direito desta região. Com o indicador igual a **FALSE**, o cursor permanece na última linha.

CONSIDERAÇÕES ESPECIAIS

1. Quando o rolamento ocorre na janela, a rotina **wrefresh** é chamada, para efetuar o mesmo rolamento na tela.

Página intencionalmente em branco.

SET_TERM

SINOPSE

```
TELA *
set_term( novo_term )
TELA *novo_term;
```

DESCRIÇÃO SUCINTA

A rotina **set_term** informa a CURSES qual dos diversos terminais criados pelo programa (ver a rotina newterm) será usado a partir de agora. **Novo_term** é um apontador do terminal desejado.

CONSIDERAÇÕES ESPECIAIS

1. **Set_term** é usado apenas quando o programa utiliza vários terminais.

VALOR DE RETORNO

Set_term retorna um apontador para o terminal usado anteriormente.

Página intencionalmente em branco.

SUBWIN

SINOPSE

```
JANELA *
subwin( jan, nlinhas, ncolunas, y, x )
JANELA   *jan;
int      nlinhas, ncolunas, y, x;
```

DESCRIÇÃO SUCINTA

A rotina **subwin** cria uma subjanela dentro da janela **jan**. A nova subjanela tem **nlinhas** linhas e **ncolunas** colunas. A posição inicial da subjanela (vértice superior esquerdo) é: linha = y, coluna = x, relativa à tela e não à origem da janela **jan**. O cursor inicialmente está na posição (0, 0) da subjanela. O conteúdo da subjanela é compartilhado com a janela **jan**. Uma modificação a este conteúdo afeta ambas as janelas.

CONSIDERAÇÕES ESPECIAIS

1. Se **nlinhas** for igual a zero, as linhas da subjanela iniciam na linha y e terminam na última linha da janela **jan**.

2. Se **ncolunas** for igual a zero, as colunas da subjanela iniciam na coluna x e terminam na última coluna da janela **jan**.

3. Ao usar esta rotina, freqüentemente será necessário usar a rotina **touchwin** () antes de chamar **wrefresh** ().

VALOR DE RETORNO

Normalmente, o valor retornado é um apontador para a subjanela. Em caso de erro, o valor retornado é NULL.

Os casos de erro são os seguintes:

1. Um número negativo de linhas ou de colunas.

2. A subjanela ultrapassa os limites da janela **jan**.

3. Falta de memória na alocação da estrutura para a subjanela.

TEM_PENDENDIA

SINOPSE

```
int
tem_pendencia()
```

DESCRIÇÃO SUCINTA

A rotina **tem_pendencia** retorna **TRUE**, se houver caracteres digitados, mas não lidos, na entrada, retornando **FALSE**, caso contrário.

Página intencionalmente em branco.

TOUCHWIN

SINOPSE

```
void
touchwin( jan )
JANELA    *jan;
```

DESCRIÇÃO SUCINTA

A rotina **touchwin** informa à rotina **wnoutrefresh** que cada posição da janela **jan** foi alterada e deve ser considerada para exibição.

CONSIDERAÇÕES ESPECIAIS

1. A utilização da rotina **touchwin** é indicada, quando houver janelas com partes sobrepostas na tela física. Para exibir a janela parcial ou completamente escondida, deve-se chamar a rotina **touchwin** antes de chamar **wrefresh**. Caso contrário, o algoritmo de otimização da rotina **wrefresh** poderá não atualizar a tela física de forma correta.

Página intencionalmente em branco.

TYPEAHEAD

SINOPSE

```
void
typeahead (fd)
int  fd;
```

DESCRIÇÃO SUCINTA

A rotina **typeahead** informa a CURSES que a atualização da tela física com a rotina **doupdate** deve ser sustada, se houver caracteres pendentes no arquivo cujo número lógico associado é **fd.**

CONSIDERAÇÕES ESPECIAIS

1. Por default, o arquivo usado é o arquivo de entrada especificado na rotina **newterm.** Observe que a rotina **initscr** chama **newterm**, especificando a entrada padrão (**fd** = 0) como arquivo de entrada.

2. Se o valor de **fd** for -1, não haverá verificação de caracteres pendentes na entrada, e a rotina **doupdate** sempre atualizará a tela.

Página intencionalmente em branco.

UNCTRL

SINOPSE

```
#define unctrl( ch )...
caractere ch;
```

DESCRIÇÃO SUCINTA

A rotina **unctrl** é uma macro (definida no arquivo **unctrl.h**) que expande o caractere **ch** para uma cadeia de caracteres visíveis representando o caractere **ch.**

CONSIDERAÇÕES ESPECIAIS

1. Caracteres de controle são expandidos para a notação "°X".

2. Para a utilização da macro **unctrl**, o arquivo **unctrl** deve ser incluído.

3. O resultado da macro é do tipo char *.

Página intencionalmente em branco.

WACENT, ACENT

SINOPSE

```
void
wacent(jan, indicador)
JANELA   *jan;
int      indicador;

#define acent(indicador)   wacent(stdscr, indicador)
```

DESCRIÇÃO SUCINTA

A rotina **wacent** chamada com **indicador** igual a **TRUE** permite que caracteres acentuados sejam lidos e colocados na janela **jan.** Com **indicador** igual a **FALSE**, caracteres acentuados serão desacentuados tanto na entrada como na saída (ver a rotina **desacentFN**).

Página intencionalmente em branco.

WADDCH, ADDCH

SINOPSE

```
int
waddch(jan, c)
JANELA   *jan;
caractere c;

#define addch(c)        waddch(stdscr,c)
```

DESCRIÇÃO SUCINTA

A rotina **waddch** adiciona o caractere **c** à janela **jan** na posição atual do cursor, sobrepondo o caractere presente nesta posição antes da chamada. O cursor avança uma coluna à direita, na mesma linha. O caractere recebe os atributos correntes da janela (ver as rotinas wattron, wattrset, wattroff, wstandout e wstandend).

CONSIDERAÇÕES ESPECIAIS

1. Se o caractere **c** já possui atributos, os atributos da janela serão adicionados aos já presentes.

2. Quando o cursor avança, ele passa da última coluna de uma linha à primeira coluna da linha seguinte. O cursor não avança, se estiver na última coluna da última linha da região de rolamento (ver a rotina wsetscrreg), a não ser que o rolamento seja

permitido (ver a rotina scrollok). Neste caso, é realizado um rolamento na região de rolamento (ver a rotina wnscroll), e o cursor é posicionado na primeira coluna da nova última linha.

3. O efeito de adicionar um caractere **ASCII** LF (valor 0x0A) a uma janela é de branquear (com **ASCII** SP) todas as colunas da linha corrente, a partir da posição do cursor. O cursor passa para a primeira coluna da próxima linha, de acordo com a consideração 1 acima.

4. O efeito de adicionar um caractere **ASCII** CR (valor 0x0D) a uma janela é de posicionar o cursor na primeira coluna da linha corrente.

5. O efeito de adicionar um caractere **ASCII** BS (valor 0x08) a uma janela é de mover o cursor uma coluna à esquerda, sem ultrapassar a primeira coluna da linha corrente. Se o cursor já estiver na primeira coluna, nada acontece.

6. Ao adicionar o caractere **ASCII** HT (valor 0x09) a uma janela, a expansão para brancos é feita, mantendo-se paradas de tabulação a cada oito colunas. O final da linha não é ultrapassado.

7. Um caractere não-visível é convertido em um ou mais caracteres visíveis na janela (ver a rotina unctrl).

8. Tratamento de caracteres acentuados.

   Caracteres acentuados da língua portuguesa podem ser adicionados a uma janela. A representação dos caracteres segue o padrão ABICOMP. Estes caracteres podem ser entregues a addch usando um único caractere com o oitavo bit ligado ou uma seqüência **ASCII**:

SO , caracteres em ASCII , SI

A representação interna dos caracteres numa janela usa 8 bits. Em um determinado momento, uma janela pode estar apta a receber caracteres acentuados ou não (ver a rotina wacent) e pode estar apta a tratar seqüências SO ... SI ou não (ver a rotina wmodoabc). Além do mais, o terminal sendo usado pode representar ou não caracteres acentuados na tela. A ação tomada por **waddch** é resumida no quadro a seguir.

JANELA ACEITA SEQUÊNCIA	JANELA ACEITA ACENTUAÇÃO	TERMINAL TEM ACENTUAÇÃO	REPRESENTAÇÃO ENTREGUE	AÇÃO TOMADA
não	não	não	8 bits	1
não	não	não	seqüência	2
não	não	sim	8 bits	1
não	não	sim	seqüência	2
não	sim	não	8 bits	1
não	sim	não	seqüência	2
não	sim	sim	8 bits	4
não	sim	sim	seqüência	2
sim	não	não	8 bits	1
sim	não	não	seqüência	3
sim	não	sim	8 bits	1
sim	não	sim	seqüência	3
sim	sim	não	8 bits	1
sim	sim	não	seqüência	3
sim	sim	sim	8 bits	4
sim	sim	sim	seqüência	5

As ações são as seguintes:

1. "Desacentua" o caractere (por exemplo, ç se transforma em c) e o adiciona à janela.

2. Exibe cada caractere da seqüência (incluindo SO e SI) normalmente.

3. Converte a seqüência a caractere de 8 bits (liga o oitavo bit), removendo os delimitadores SO e SI. Os caracteres são adicionados à janela, após desacentuação.

4. Adiciona o caractere à janela.

5. Como a ação 3, mas sem desacentuação.

VALOR DE RETORNO

O valor normalmente retornado é **OK.** Em caso de erro, o código de retorno é **ERRO.** As condições de erro são as seguintes:

1. A posição original do cursor está fora da janela.

2. A nova posição do cursor causaria um rolamento da tela, e o rolamento não é permitido.

Página intencionalmente em branco.

WADDSTR, ADDSTR

SINOPSE

```
int
waddstr(jan, ap_cadeia)
JANELA   *jan;
char     *ap_cadeia;

#define addstr(ap_cadeia) waddstr(stdscr, ap_cadeia)
```

DESCRIÇÃO SUCINTA

A rotina **waddstr** adiciona uma cadeia de caracteres à janela **jan**, a partir da posição atual do cursor, sobrepondo os caracteres que estavam nestas posições. **Ap_cadeia** é um apontador para a cadeia a ser adicionada.

CONSIDERAÇÕES ESPECIAIS

1. A rotina **waddstr** adiciona cada caractere na janela, usando a rotina **waddch**.

VALOR DE RETORNO

Normalmente, o valor de retorno é **OK**. Em caso de erro, o código de retorno é **ERRO**. Os casos de erro são fornecidos pela rotina **waddch**.

Página intencionalmente em branco.

WATTROFF, ATTROFF

SINOPSE

```
int
wattroff(jan, atrib)
JANELA   *jan;
int      atrib;

#define attroff(atrib)  wattroff(stdscr, atrib)
```

DESCRIÇÃO SUCINTA

A rotina **wattroff** desliga os atributos de vídeo **atrib** da janela **jan.**

CONSIDERAÇÕES ESPECIAIS

1. **Wattroff** não afeta os outros atributos correntes da janela **jan.**

2. Os atributos de vídeo são apresentados na descrição da rotina **wattron.**

VALOR DE RETORNO

A rotina **wattroff** sempre retorna **OK.**

Página intencionalmente em branco.

WATTRON, ATTRON

SINOPSE

```
int
wattron(jan, atrib)
JANELA   *jan;
int      atrib;

#define attron(atrib)   wattron(stdscr, atrib)
```

DESCRIÇÃO SUCINTA

A rotina **wattron** liga os atributos de vídeo **atrib** para a janela **jan**. Cada caractere adicionado ou inserido na janela recebe os atributos correntes da janela.

CONSIDERAÇÕES ESPECIAIS

1. **Wattron** não afeta os outros atributos correntes da janela **jan**.

2. Os possíveis atributos de vídeo são os seguintes (ver o arquivo **curses.h**):

   A_STANDOUT (o modo de destaque do terminal);
   A_UNDERLINE (sublinhado);
   A_REVERSE (vídeo inverso);
   A_BLINK (piscante);
   A_DIM (atenuado);
   A_BOLD (intensificado);
   A_NORMAL (sem atributos).

3. Os atributos podem ser ligados em combinações, por exemplo:

   attron(A_BOLD | A_BLINK);

4. Atributos aplicam-se a caracteres e não a posições da janela. Isto é, se um caractere for movido na janela, os atributos movem-se com o caractere.

VALOR DE RETORNO

A rotina **wattron** sempre retorna **OK.**

WATTRSET, ATTRSET

SINOPSE

```
int
wattrset( jan, atrib )
JANELA    *jan;
int       atrib;

#define attrset(atrib)    wattrset(stdscr, atrib)
```

DESCRIÇÃO SUCINTA

A rotina **wattrset** estabelece **atrib** como o valor dos atributos da janela **jan**.

CONSIDERAÇÕES ESPECIAIS

1. Os atributos de vídeo estão descritos na rotina **wattron**.

VALOR DE RETORNO

Wattrset sempre retorna **OK**.

Página intencionalmente em branco.

WCLEAR, CLEAR

SINOPSE

```
int
wclear(jan)
JANELA   *jan;

#define clear()     wclear(stdscr)
```

DESCRIÇÃO SUCINTA

A rotina **wclear** preenche com caracteres brancos toda a janela **jan**. Um código de controle para apagar a tela do terminal será gerado na próxima atualização da tela (ver a rotina clearok). O cursor da janela posiciona-se na primeira coluna da primeira linha.

CONSIDERAÇÕES ESPECIAIS

1. **Wclear** é preferível a **werase**, quando se sabe que a janela ocupa toda a tela do terminal.

VALOR DE RETORNO

Wclear sempre retornará **OK**.

Página intencionalmente em branco.

WCLRTOBOT, CLRTOBOT

SINOPSE

```
void
wclrtobot( jan )
JANELA    *jan;

#define clrtobot()      wclrtobot( stdscr )
```

DESCRIÇÃO SUCINTA

A rotina **wclrtobot** preenche de caracteres brancos todas as linhas da janela **jan**, abaixo da posição do cursor. A linha corrente, a partir da posiçao do cursor, inclusive, é apagada.

CONSIDERAÇÕES ESPECIAIS

1. A posição do cursor não é alterada com a chamada de **wclrtobot.**

Página intencionalmente em branco.

WCLRTOEOL, CLRTOEOL

SINOPSE

```
void
wclrtoeol(jan)
JANELA    *jan;

#define clrtoeol()      wclrtoeol(stdscr)
```

DESCRIÇÃO SUCINTA

A rotina **wclrtoeol** preenche com caracteres brancos a linha da janela **jan**, a partir da posição atual do cursor, inclusive, até a última coluna da linha.

CONSIDERAÇÕES ESPECIAIS

1. A posição do cursor não é alterada com a chamada de **wclrtoeol**.

Página intencionalmente em branco.

WDELCH, DELCH

SINOPSE

```
int
wdelch(jan)
JANELA   *jan;

#define delch     wdelch(stdscr)
```

DESCRIÇÃO SUCINTA

A rotina **wdelch** deleta um caractere da janela **jan**, na posição atual do cursor. O resto da linha é movido uma posição à esquerda, e, na última coluna da mesma, é adicionado um caractere em branco.

CONSIDERAÇÕES ESPECIAIS

1. O cursor permanece na posição em que se encontrava antes da chamada de **wdelch**.

VALOR DE RETORNO

Wdelch sempre retorna **OK**.

Página intencionalmente em branco.

WDELETELN, DELETELN

SINOPSE

```
void
wdeleteln(jan)
JANELA   *jan;

#define deleteln()       wdeleteln(stdscr)
```

DESCRIÇAO SUCINTA

A rotina **wdeleteln** deleta da janela **jan** a linha onde se encontra o cursor. As linhas abaixo da linha onde se encontra o cursor são movidas uma posição acima. A última linha da janela é preenchida com caracteres brancos.

CONSIDERAÇOES ESPECIAIS

1. A posição do cursor não é alterada com a chamada da rotina **wdeleteln**.

2. Se a linha onde se encontra o cursor for a última, **wdeleteln** simplesmente preenche esta linha com brancos.

3. O movimento das linhas para cima é realizado mesmo em janelas onde não é permitido fazer rolamento (ver a rotina scrollok).

Página intencionalmente em branco.

WERASE, ERASE

SINOPSE

```
void
werase(jan)
JANELA   *jan;

#define erase()              werase(stdscr)
```

DESCRIÇAO SUCINTA

A rotina **werase** preenche com caracteres brancos toda a janela **jan**. O cursor da janela passa para a primeira coluna da primeira linha.

CONSIDERAÇOES ESPECIAIS

1. **Werase** é preferível a **erase** para janelas que ocupam apenas parte da tela do terminal.

2. **Werase** não gera um código de controle para apagar a tela (ver clearok e wclear), Portanto, é mais lenta do que wclear para apagar janelas que ocupam a tela inteira.

Página intencionalmente em branco.

WGETCH, GETCH

SINOPSE

```
int
wgetch(jan)
JANELA   *jan;

#define getch()        wgetch(stdscr)
```

DESCRIÇÃO SUCINTA

A rotina **wgetch** lê um caractere do terminal, associando-o à janela **jan**. Se a janela estiver em modo nodelay (ver a rotina **nodelay**), wgetch funciona de forma não-bloqueada. Caso contrário, wgetch espera que um caractere seja teclado. Se o terminal estiver em modo nocbreak (ver a rotina **nocbreak**) o caractere teclado só é passado após o recebimento de um caractere "**nova_linha**" ou "**retorno_de_carro**". Os caracteres são adicionados automaticamente na janela jan com a rotina **waddch**, se o modo eco estiver ligado (ver a rotina **echo**).

CONSIDERAÇÕES ESPECIAIS

1. Se o terminal pode entregar caracteres acentuados, eles terão representação em 8 bits, padrão ABICOMP, quando retornados por wgetch. Se o terminal enviar seqüências de caracteres para representar caracteres acentuados, a combinação das seqüências

em caracteres acentuados só será feita se o modo ABICOMP estiver ligado (ver a rotina **wmodoabc**).

2. Os caracteres são "desacentuados" automaticamente, a não ser que o modo de acentuação da janela esteja ligado (ver a rotina **wacent**).

VALOR DE RETORNO

Wgetch normalmente retorna o caractere lido do terminal, de acordo com as considerações acima. Em caso de erro, o valor retornado é **ERRO**. As condições de erro são as seguintes:

1. O modo eco está ligado, e a janela não tem permissão para fazer rolamento, e o cursor encontra-se na última posição da região de rolamento (ver a rotina **scrollok**).

2. O terminal está no modo **nodelay** (ver a rotina **nodelay**), e nenhum caractere foi digitado.

WGETSTR, GETSTR

SINOPSE

```
int
wgetstr(jan, ap_cadeia)
JANELA   *jan;
char     *ap_cadeia;

#define getstr(ap_cadeia) wgetstr(stdscr, ap_cadeia)
```

DESCRIÇÃO SUCINTA

A rotina **wgetstr** lê uma cadeia de caracteres do teclado e a armazena na cadeia **ap_cadeia**.

CONSIDERAÇÕES ESPECIAIS

1. A rotina **wgetstr** lê caractere a caractere, usando a rotina **getch**, terminando ao encontrar um caractere "nova_linha" ou "retorno_de_carro".

2. Os caracteres de controle para apagamento de caractere e de linha são interpretados, como também o são as teclas-função KEY_LEFT e KEY_BACKSPACE.

3. No final da cadeia, é adicionado o caractere '\0'.

VALOR DE RETORNO

Normalmente, o valor retornado é **OK.** Em caso de erro, o valor de retorno é **ERRO.** Os casos de erro são os mesmos apresentados na rotina **wgetch** (ver a rotina **wgetch**).

WINCH, INCH

SINOPSE

```
#define  winch(jan)...
JANELA    *jan;

#define inch()     winch(stdscr)
```

DESCRIÇÃO SUCINTA

Winch é uma macro (definida no arquivo **curses.h**) que retorna o caractere que está localizado na posição corrente do cursor na janela **jan**, juntamente com os atributos de vídeo deste caractere.

CONSIDERAÇÕES ESPECIAIS

1. **Winch** não altera o conteúdo da janela **jan**.

2. As máscaras **A_TEXTO** e **A_ATRIBUTOS**, definidas no arquivo **curses.h** podem ser usadas para extrair, respectivamente, o caractere ou os atributos.

3. O tipo do valor da macro é caractere, definido no arquivo **curses.h**.

Página intencionalmente em branco.

WINSCH, INSCH

SINOPSE

```
int
winsch(jan, c)
JANELA   *jan;
caractere c;

#define insch(c)      winsch(stdscr, c)
```

DESCRIÇÃO SUCINTA

A rotina **winsch** insere o caractere **c** na posição atual do cursor da janela **jan**. O caractere que estava localizado nesta posição, como o restante da linha, são movidos uma posição à direita, e o último caractere da linha é perdido. O cursor não muda de posição com a chamada de **winsch**.

CONSIDERAÇÕES ESPECIAIS

1. O tratamento de caracteres acentuados segue o tratamento dado na rotina **waddch**.

2. O tratamento de caracteres especiais (tabulação, nova-linha, etc.) segue o tratamento dado na rotina waddch.

VALOR DE RETORNO

Normalmente, o valor de retorno é **OK.** Em caso de erro, o valor de retorno é **ERRO.** A condição de erro é a seguinte:

1. A inserção de um caractere de nova-linha, quando o cursor da janela está na última linha da região de rolamento, e o rolamento não é permitido (ver a rotina scrollok).

WINSERTLN, INSERTLN

SINOPSE

```
int
winsertln(jan)
JANELA   *jan;

#define insertln()       winsertln(stdscr)
```

DESCRIÇÃO SUCINTA

A rotina **winsertln** adiciona à janela **jan** uma linha em branco na posição atual do cursor. Todas as linhas a partir da atual são movidas uma posição abaixo. A última linha da janela é perdida.

CONSIDERAÇÕES ESPECIAIS

1. A posição do cursor não é alterada na chamada de **winsertln**.

VALOR DE RETORNO

Winsertln sempre retorna **OK**.

Página intencionalmente em branco.

WMODOABC, MODOABC

SINOPSE

```
int
wmodoabc(jan, indicador)
JANELA   *jan;
bool     indicador;

#define modoabc(indicador) wmodoabc(stdscr,indicador)
```

DESCRIÇÃO SUCINTA

A rotina **wmodoabc** habilita (se o valor de **indicador** for igual a **TRUE**) ou desabilita (se o valor de **indicador** for igual a **FALSE**) o uso de seqüências ABICOMP para representar caracteres acentuados na janela **jan** (ver a rotina **waddch**).

CONSIDERAÇÕES ESPECIAIS

1. Internamente, CURSES usa o conjunto de caracteres ABICOMP, onde um único octeto de 8 bits representa cada caractere.

VALOR DE RETORNO

Wmodoabc sempre retorna o valor OK.

Página intencionalmente em branco.

WMOVE, MOVE

SINOPSE

```
int
wmove(jan, y, x)
JANELA   *jan;
int y, x;

#define move(y, x)        wmove(stdscr, y, x)
```

DESCRIÇAO SUCINTA

A rotina **wmove** move o cursor para a posição (y, x) da janela **jan**. A posição (y, x) é relativa à posição (0, 0), vértice superior esquerdo da janela.

CONSIDERAÇOES ESPECIAIS

1. O movimento só ocorre se a posição (y, x) estiver dentro do limite da janela.

2. O cursor do terminal corresponde à posição do cursor da última janela sujeita à rotina **wrefresh** (ver também a rotina leaveok).

VALOR DE RETORNO

O valor normalmente retornado é **OK.** Em caso de erro, o valor de retorno é **ERRO.** A condição de erro é a seguinte:

1. Tentativa de mover o cursor fora dos limites da janela.

WNOUTREFRESH

SINOPSE

```
int
wnoutrefresh(jan)
JANELA   *jan;
```

DESCRIÇÃO SUCINTA

A rotina **wnoutrefresh** copia os trechos alterados da janela **jan** para a tela virtual **curscr** (ver as rotinas wrefresh e doupdate).

VALOR DE RETORNO

Wnoutrefresh sempre retorna **OK.**

Página intencionalmente em branco.

WNSCROLL,NSCROLL,SCROLL

SINOPSE

```
void
wnscroll(jan, n)
JANELA   *jan;
int      n;

#define nscroll(n)       wnscroll(stdscr, n)

void
scroll(jan)
JANELA   *jan;
```

DESCRIÇÃO SUCINTA

A rotina **wnscroll** realiza um rolamento lógico de **n** linhas na janela **jan.** O rolamento é para cima, se n for positivo, e para baixo, se n for negativo.

CONSIDERAÇOES ESPECIAIS

1. O efeito de fazer um rolamento lógico de n linhas para cima na janela é o seguinte: as primeiras n linhas da região de rolamento da janela (ver a rotina wsetscrreg) são removidas, e todas as linhas abaixo delas são movimentadas n posições para cima, e n linhas em branco são inseridas no final da região de rolamento da janela. O rolamento para baixo (n negativo) é simétrico, com remoções no fim e inserções no início da região de rolamento.

2. O cursor move com o texto, mas não sai da região de rolamento.

3. A rotina **scroll** não é uma macro por questão de compatibilidade com aplicações já existentes. É equivalente a:

   wnscroll(jan, 1)

WPRINTW, PRINTW,MVWPRINTW,MVPRINTW

SINOPSE

```
int
wprintw( jan, form, arg1, arg2, ... )
JANELA     *jan;
char       *form;
int        arg1, arg2, ...;

printw( form, arg1, arg2, ... )
char       *form;
int        arg1, arg2, ...;

mvwprint( jan, y, x, form, arg1, arg2, ...)
JANELA * jan;
int y, x;
char * form;
int arg1, arg2, ...;

mvprintw( y, x, form, arg1, arg2, ...)
int y, x;
char * form;
int arg1, arg2, ...;
```

DESCRIÇÃO SUCINTA

A rotina **wprintw** realiza na janela **jan** uma impressão no formato form dos argumentos arg1, arg2, ... A interpretação do formato e argumentos segue a mesma semântica da rotina padrão de entrada/saída **printf.**

CONSIDERAÇÕES ESPECIAIS

1. Os argumentos no formato form são adicionados à janela **jan** através da rotina **waddstr.**

2. **Printw** é equivalente à rotina **wprintw,** mas age sobre a janela stdscr.

3. **mvwprintw** e **mvprintw** movem o cursor utilizando o rotina **wmove** e realizam a ação a elas correspondentes.

4. Observe que **printw, mvwprintw** e **mvprintw** não são macros.

VALOR DE RETORNO

Wprintw retorna o valor retornado pela rotina **waddstr.**

WREFRESH, REFRESH

SINOPSE

```
int
wrefresh( jan )
JANELA    *jan;

#define refresh()        wrefresh( stdscr )
```

DESCRIÇAO SUCINTA

A rotina **wrefresh** atualiza a tela, de forma a exibir a janela **jan**.

CONSIDERAÇOES ESPECIAIS

1. A janela **jan** deve estar localizada dentro dos limites da tela do terminal.

2. **Wrefresh** pode apagar a tela do terminal e redesenhá-la por inteiro, caso a rotina **clearok** tenha sido chamada desde o último **wrefresh** dado (ver a rotina **clearok**). O mesmo ocorre se **wrefresh** for chamada com parâmetro **curscr**.

3. A posição do cursor após a atualização da tela depende da última chamada à rotina **leaveok** (ver a rotina **leaveok**). Normalmente, o cursor da tela seguirá a posição do cursor da última janela sujeita a **wrefresh**.

4. A rotina **wrefresh** chama a rotina **wnoutrefresh** e, em seguida, **doupdate.**

VALOR DE RETORNO

Wrefresh sempre retorna **OK.**

WSCANW, SCANW,MVWSCANW,MVSCANW

SINOPSE

```
int
wscanw( jan, form, arg1, arg2, ... )
JANELA    *jan;
char      *form;
int       arg1, arg2, ...;

int
scanw( form, arg1, arg2, ... )
char      *form;
int       arg1, arg2, ...;

mvwscanw( jan, y, x, form, arg1, arg2, ...)
JANELA * jan;
int y, x;
char * form;
int arg1, arg2, ...;

mvscanw( y, x, form, arg1, arg2, ...)
int, y, x;
char * form;
int arg1, arg2, ...;
```

DESCRIÇÃO SUCINTA

A rotina **wscanw** lê os argumentos arg1, arg2, ... do teclado do terminal no formato form, seguindo a mesma semântica da rotina padrão de entrada/saída **scanf.**

CONSIDERAÇÕES ESPECIAIS

1. Os argumentos no formato form são lidos do teclado do terminal através da rotina **wgetstr.**

2. A rotina **scanw** é equivalente à rotina **wscanw**, mas age sobre a janela **stdscr.**

3. **mvwscanw** e **mvscanw** movem o cursor utilizando a rotina **wmove** e realizam a ação a elas correspondentes.

4. Observe que **scanw**, **mvwscanw** e **mwscanw** não são macros.

VALOR DE RETORNO

Se a rotina **wgetstr** retornar **ERRO**, **wscanw** também retornará este valor. Caso contrário, o valor retornado tem a mesma semântica do valor retornado pela rotina **scanf.**

WSETSCRREG, SETSCRREG

SINOPSE

```
void
wsetscrreg( jan, marg_inicial, marg_final )
JANELA    *jan;
int       marg_inicial, marg_final;

#define setscrreg(marg_inicial, marg_final)
        wsetscrreg( stdscr, marg_inicial, marg_final )
```

DESCRIÇAO SUCINTA

A rotina **wsetscrreg** permite determinar a região da janela **jan** afetada por um rolamento lógico (ver a rotina wnscroll). A região é composta das linhas entre **marg_inicial** e **marg_final**, inclusive (a linha 0 é a linha inicial da janela).

CONSIDERAÇOES ESPECIAIS

1. Inicialmente, a região de rolamento é a janela inteira.

VALOR DE RETORNO

Normalmente, **wsetscrreg** retorna **OK**. Caso a região de rolamento especificada não esteja dentro dos limites da janela, o valor **ERRO** é retornado.

Página intencionalmente em branco.

WSTANDEND, STANDEND

SINOPSE

```
int
wstandend( jan )
JANELA    *jan;

#define standend()      wstandend( stdscr )
```

DESCRIÇÃO SUCINTA

A rotina **wstandend** retira o atributo de vídeo A_STANDOUT da janela **jan.**

CONSIDERAÇÕES ESPECIAIS

1. O efeito de **wstandend(jan)** é o mesmo de

   wattroff(jan, A_STANDOUT)

VALOR DE RETORNO

Wstandend sempre retorna **OK.**

Página intencionalmente em branco.

WSTANDOUT, STANDOUT

SINOPSE

```
int
wstandout( jan )
JANELA    *jan;

#define standout()       wstandout( stdscr )
```

DESCRIÇÃO SUCINTA

A rotina **wstandout** estabelece o atributo de vídeo **A_STANDOUT** para a janela **jan**.

CONSIDERAÇÕES ESPECIAIS

1. O efeito de **wstandout(jan)** é o mesmo de

   wattron(jan, A_STANDOUT)

VALOR DE RETORNO

Wstandout sempre retorna **OK**.

Página intencionalmente em branco.

XON, NOXON

SINOPSE

```
void
xon()

void
noxon()
```

DESCRIÇÃO SUCINTA

A rotina **xon** habilita o tratamento de caracteres de controle de fluxo no terminal (teclas CONTROLE S e CONTROLE Q).

A rotina **noxon** desabilita o tratamento de caracteres de controle de fluxo no terminal.

CONSIDERAÇÕES ESPECIAIS

1. O valor default para este tratamento é obtido do controlador de terminal do sistema operacional, quando da chamada a **initscr**.

3.2 ROTINAS DA BIBLIOTECA TERMINFO

TERMINFO é um conjunto de sub-rotinas e funções para a manipulação de terminais. A biblioteca CURSES usa as rotinas do TERMINFO para atingir a independência de terminal. Em alguns casos raros, um programa que use CURSES também terá que usar rotinas do TERMINFO diretamente. A descrição das características do terminal é fornecida no Módulo 2 deste manual. Para usar a biblioteca TERMINFO, o arquivo **term.h** deve ser incluído no programa. As rotinas desta biblioteca são descritas a seguir.

MVCUR

SINOPSE

```
void
mvcur( antiga_linha, antiga_coluna, nova_linha, nova_
       coluna)
int    antiga_linha, antiga_coluna, nova_linha, nova_
       coluna;
```

DESCRIÇÃO SUCINTA

A rotina **mvcur** movimenta o cursor na tela, de forma otimizada e eficiente, da posição (antiga_linha, antiga_coluna) para a posição (nova_linha, nova_coluna).

Página intencionalmente em branco.

PUTP

SINOPSE

```
void
putp( cadeia)
char *cadeia;
```

DESCRIÇÃO SUCINTA

A rotina **putp** tem o mesmo efeito da chamada

tputs(cadeia, 1, putchar)

(ver a rotina **tputs**).

Página intencionalmente em branco.

SETTERM

SINOPSE

```
void
setterm( term)
char *term
```

DESCRIÇÃO SUCINTA

A rotina **setterm** tem o mesmo efeito da chamada

setupterm(term, 1, (int *) 0)

Página intencionalmente em branco.

SETUPTERM

SINOPSE

```
int
setupterm( term, fd_saida, stat_erro)
char *term;
int  fd_saida;
int  *stat_erro;
```

DESCRIÇÃO SUCINTA

A rotina **setupterm** lê a descrição do terminal **term** no TERMINFO e inicializa as estruturas de controle do terminal. Qualquer saída usa o descritor de arquivo **fd_saída**. Se o argumento **stat_erro** for diferente de **NULL**, **setupterm** armazena um valor em ***stat_erro**, indicando o resultado da operação.

CONSIDERAÇÕES ESPECIAIS

1. Se o argumento **stat_erro** é **NULL**, **setupterm** imprime uma mensagem de erro e encerra o programa, em caso de erro.

2. Se o argumento **term** é NULL, a variável do ambiente "**TERM**" é usada.

3. **Setupterm** não inicializa as estruturas usadas por CURSES (isto é, stdscr, curscr, ...).
4. **Setupterm** inicializa as variáveis LINHAS e COLUNAS do TERMINFO, da seguinte maneira: se as variáveis do ambiente LINES e/ou COLUMNS existem, elas são usadas para a inicialização. Caso contrário, os valores presentes na descrição do terminal são usados.

VALOR DE RETORNO

Setupterm normalmente retorna **OK.** Em caso de erro, e se **stat_erro** for diferente de **NULL**, o valor retornado é **ERRO**, e um código de erro é armazenado em ***stat_erro.** Em caso de erro, e se **stat_erro** for igual a **NULL**, **setupterm** não retorna. Os casos de erro são os seguintes:

1. A descrição do terminal **term** não foi encontrada, ou não pode ser lida.

2. Não há memória para alocar as estruturas de controle do terminal.

3. A descrição do terminal não possui o formato correto.

TIC

NOME

tic - compilador TERMINFO

SINOPSE

tic [-v[n]] arquivo

DESCRIÇÃO SUCINTA

Tic tranforma um arquivo contendo especificações de terminais para um formato reconhecido pelas rotinas das bibliotecas CURSES e TERMINFO. O formato do arquivo-fonte está descrito no Módulo 2 deste manual. Este arquivo pode conter a especificação de um ou mais terminais. Para cada terminal, o resultado da compilação é deixado nos arquivos

```
/usr/lib/terminfo/x1/nome1
/usr/lib/terminfo/x2/nome2 ...
```

onde nome1, ... são os nomes deste terminal dados na especificação. x1 é a primeira letra de nome1, x2 é a primeira letra de nome2, etc. Se a variável do ambiente TERMINFO existe, são usados os arquivos criados neste diretório, ao invés de **/usr/lib/terminfo.** A opção -v pode ser usada para obter detalhes das operações feitas por **tic.** Se um número entre 1 e 10 seguir a opção -v, é tomado como o nível de detalhes desejado. Um número maior fornece mais detalhes. O valor default de n é 1.

Quando **tic** encontra a característica **use=nome**, a descrição do terminal chamado **nome** é procurada no arquivo corrente. Caso não exista, o arquivo **terminfo.src** é examinado.

O tamanho final da descrição de um terminal não pode ser maior que 4.096 octetos. O campo especificando o nome extenso do terminal não pode ultrapassar 128 octetos.

TPARM

SINOPSE

```
char *
tparm( cadeia, p1, p2, ..., p9 )
char *cadeia;
int  p1, p2, ..., p9;
```

DESCRIÇÃO SUCINTA

A rotina **tparm** realiza a substituição dos parâmetros da cadeia de caracteres **cadeia** pelos parâmetros p1, p2, ... ,p9 e retorna um apontador para o resultado da cadeia de caracteres com os parâmetros p1, p2, ... ,p9, substituídos (ver cadeias de caracteres parametrizadas no TERMINFO).

VALOR DE RETORNO

Normalmente, **tparm** retorna um apontador para uma cadeia de caracteres. Em caso de erro, o valor retornado é **NULL.** As condições de erro são as seguintes:

1. O argumento cadeia é igual a **NULL.**

2. Número de parâmetros errado.

3. Sintaxe da cadeia errada.

Página intencionalmente em branco.

TPUTS

SINOPSE

```
void
tputs(cadeia, num_linhas, saida)
char *cadeia;
int  num_linhas;
int  (*saida)();
```

DESCRIÇÃO SUCINTA

A rotina **tputs** imprime a cadeia, inserindo caracteres de preenchimento onde necessário. Cadeia deve ser uma variável do tipo cadeia de caracteres ou o valor de retorno da rotina **tparm, tgoto ou tgetstr. Num_linhas** é o número de linhas afetadas na tela pela impressão da cadeia. **Saida** é uma rotina semelhante a **putchar**, que imprime os caracteres, um de cada vez.

CONSIDERAÇÕES ESPECIAIS

1. Se a aplicação de **num_linhas** não for necessária, seu valor deve ser um 1.

Página intencionalmente em branco.

VIDATTR

SINOPSE

```
void
vidattr( atributos)
int  atributos;
```

DESCRIÇÃO SUCINTA

A rotina **vidattr** é semelhante à rotina **vidputs**, mas imprime com a rotina **putchar**.

Página intencionalmente em branco.

VIDPUTS

SINOPSE

```
void
vidputs ( atributos, saida )
int  atributos;
int  (*saida)();
```

DESCRIÇÃO SUCINTA

A rotina **vidputs** imprime a cadeia de caracteres necessária para colocar o terminal no modo de atributos de vídeo. Estes atributos podem ser qualquer combinação dos atributos de vídeo existente no seu terminal. Saída é uma rotina semelhante à **putchar** que imprime os caracteres na tela, um de cada vez.

CONSIDERAÇÕES ESPECIAIS

1. Os atributos de vídeo do terminal estão descritos no TERMINFO.

Página intencionalmente em branco.

3.3 ROTINAS DA BIBLIOTECA TERMCAP

O TERMINFO é uma evolução da biblioteca TERMCAP, mais antiga. Entretanto a biblioteca TERMINFO inclui rotinas de compatibilização com TERMCAP, para programas baseados nesta biblioteca. Tais rotinas são descritas a seguir.

Página intencionalmente em branco.

TGETENT

SINOPSE

```
int
tgetent( buffer, nome)
char *buffer;
char *nome;
```

DESCRIÇAO SUCINTA

A rotina **tgetent** lê a descrição do terminal do tipo nome e inicializa as estruturas do TERMINFO.

CONSIDERAÇOES ESPECIAIS

1. O argumento buffer é ignorado.

VALOR DE RETORNO

Normalmente **tgetent** retorna 1. Em caso de erro, retorna os seguintes valores.

- 0, se o terminal não estiver cadastrado no TERMINFO.

- -1, se a descrição do terminal não pôde ser localizada ou lida.

Página intencionalmente em branco.

TGETFLAG

SINOPSE

```
int
tgetflag( var_bool)
char var_bool 2 ;
```

DESCRIÇÃO SUCINTA

A rotina **tgetflag** obtêm a entrada da variável booleana **var_bool** descrita no TERMINFO (ver variáveis booleanas no TERMINFO).

CONSIDERAÇÕES ESPECIAIS

1. A variável **var_bool** é referente à última descrição de terminal lida pela rotina **tgetent** ou **setupterm.**

VALOR DE RETORNO

Tgetflag retorna **TRUE**, se a variável booleana **var_bool** estiver cadastrada para o terminal, e **FALSE**, caso contrário.

Página intencionalmente em branco.

TGETNUM

SINOPSE

```
int
tgetnum( var_num )
char var_num[2] ;
```

DESCRIÇÃO SUCINTA

A rotina **tgetnum** retorna o valor da variável numérica **var_num** descrita no TERMINFO (ver variáveis numéricas no TERMINFO).

CONSIDERAÇÕES ESPECIAIS

1. A variável **var_num** é referente à última descrição de terminal lida pela rotina **tgetent ou setupterm.**

VALOR DE RETORNO

Se a variável numérica **var_num** existe na descrição do terminal, **tgetnum** retorna seu valor (um número não-negativo), caso contrário, retorna -1.

Página intencionalmente em branco.

TGETSTR

SINOPSE

```
char *
tgetstr( var_cad, buffer )
char var_cad[2] ;
char **buffer;
```

DESCRIÇÃO SUCINTA

A rotina **tgetstr** retorna um apontador para uma cadeia de caracteres, inicializada com o valor da variável do tipo cadeia **var_cad**, descrita no TERMINFO (ver variáveis do tipo cadeia no TERMINFO).

CONSIDERAÇÕES ESPECIAIS

1. A variável **var_cad** é referente à última descrição de terminal lida pela rotina **tgetent** ou **setupterm**.

2. O argumento buffer é ignorado.

VALOR DE RETORNO

Se a variável do tipo cadeia **var_cad** existir na descrição do terminal, **tgetstr** retorna um apontador para uma área estática contendo seu valor. Caso contrário, retorna **NULL**.

Página intencionalmente em branco.

TGOTO

SINOPSE

```
char *
tgoto( var_cad, coluna, linha )
char *var_cad;
int linha, coluna;
```

DESCRIÇÃO SUCINTA

A rotina **tgoto** realiza a substituição dos parâmetros da cadeia de caracteres **var_cad** pelos parâmetros **coluna** e linha e retorna um apontador para o resultado da cadeia de caracteres **var_cad** com os parâmetros coluna e linha substituídos.

CONSIDERAÇÕES ESPECIAIS

1. **Var_cad** deve ser uma variável do tipo cadeia de caracteres, correspondendo ao endereçamento do cursor (ver cadeia de caracteres de endereçamento do cursor no TERMINFO).

2. Se o valor de retorno de **tgoto** for impresso através da rotina **tputs**, o cursor será movido para a linha **linha** e coluna **coluna** da tela.

VALOR DE RETORNO

Tgoto retorna o valor retornado pela rotina **tparm.**

APÊNDICE A

TABELAS ASCII E ABICOMP

Este apêndice contém uma descrição das tabelas ASCII e ABICOMP, usadas pelo CURSES.

TABELA ASCII

DEC	OCT	EX	CARAC	DEC	OCT	EX	CARAC
000	000	00	NUL	020	024	14	DC4
001	001	01	SOH	021	025	15	NAK
002	002	02	STX	022	026	16	SYN
003	003	03	ETX	023	027	17	ETB
004	004	04	EOT	024	030	18	CAN
005	005	05	ENQ	025	031	19	EM
006	006	06	ACK	026	032	1A	SUB
007	007	07	BEL	027	033	1B	ESC
008	010	08	BS	028	034	1C	FS
009	011	09	HT	029	035	1D	GS
010	012	0A	LF	030	036	1E	RS
011	013	0B	VT	031	037	1F	US
012	014	0C	FF	032	040	20	SP
013	015	0D	CR	033	041	21	!
014	016	0E	SO	034	042	22	"
015	017	0F	SI	035	043	23	#
016	020	10	DLE	036	044	24	$
017	021	11	DC1	037	045	25	%
018	022	12	DC2	038	046	26	&
019	023	13	DC3	039	047	27	'

040	050	28	(
041	051	29	)
042	052	2A	*
043	053	2B	+
044	054	2C	,
045	055	2D	-
046	056	2E	.
047	057	2F	/
048	060	30	0
049	061	31	1
050	062	32	2
051	063	33	3
052	064	34	4
053	065	35	5
054	066	36	6
055	067	37	7
056	070	38	8
057	071	39	9
058	072	3A	:
059	073	3B	;
060	074	3C	<
061	075	3D	=
062	076	3E	>
063	077	3F	?

064	100	40	@
065	101	41	A
066	102	42	B
067	103	43	C
068	104	44	D
069	105	45	E
070	106	46	F
071	107	47	G
072	110	48	H
073	111	49	I
074	112	4A	J
075	113	4B	K
076	114	4C	L
077	115	4D	M
078	116	4E	N
079	117	4F	O
080	120	50	P
081	121	51	Q
082	122	52	R
083	123	53	S
084	124	54	T
085	125	55	U
086	126	56	V
087	127	57	W

088	130	58	X
089	131	59	Y
090	132	5A	Z
091	133	5B	[
092	134	5C	\
093	135	5D	]
094	136	5E	^
095	137	5F	_
096	140	60	`
097	141	61	a
098	142	62	b
099	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k

108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

TABELA ABICOMP

128	200	80	NUL	152	230	98	CAN
129	201	81	SOH	153	231	99	EM
130	202	82	STX	154	232	9A	SUB
131	203	83	ETX	155	233	9B	ESC
132	204	84	EOT	156	234	9C	FS
133	205	85	ENQ	157	235	9D	GS
134	206	86	ACK	158	236	9E	RS
135	207	87	BEL	159	237	9F	US
136	210	88	BS	160	240	A0	SP
137	211	89	HT	161	241	A1	À
138	212	8A	LF	162	242	A2	Á
139	213	8B	VT	163	243	A3	Â
140	214	8C	FF	164	244	A4	Ã
141	215	8D	CR	165	245	A5	Ä
142	216	8E	SO	166	246	A6	Ç
143	217	8F	SI	167	247	A7	È
144	220	90	DLE	168	250	A8	É
145	221	91	DC1	169	251	A9	Ê
146	222	92	DC2	170	252	AA	Ë
147	223	93	DC3	171	253	AB	Ì
148	224	94	DC4	172	254	AC	Í
149	225	95	NAK	173	255	AD	Î
150	226	96	SYN	174	256	AE	Ï
151	227	97	ETB	175	257	AF	Ñ

176	260	B0	O
177	261	B1	Ó
178	262	B2	Ô
179	263	B3	Õ
180	264	B4	O
181	265	B5	OE
182	266	B6	U
183	267	B7	Ú
184	270	B8	U
185	271	B9	U
186	272	BA	Y
187	273	BB	"
188	274	BC	
189	275	BD	'
190	276	BE	
191	277	BF	
192	300	C0	
193	301	C1	à
194	302	C2	á
195	303	C3	â
196	304	C4	ã
197	305	C5	a
198	306	C6	ç
199	307	C7	e

200	310	C8	é
201	311	C9	ê
202	312	CA	e
203	313	CB	ì
204	314	CC	í
205	315	CD	î
206	316	CE	i
207	317	CF	n
208	320	D0	o
209	321	D1	ó
210	322	D2	ô
211	323	D3	õ
212	324	D4	o
213	325	D5	oe
214	326	D6	u
215	327	D7	ú
216	330	D8	u
217	331	D9	u
218	332	DA	y
219	333	DB	
220	334	DC	
221	335	DD	
222	336	DE	
223	337	DF	±

Página intencionalmente em branco.

APÊNDICE B

ARQUIVOS DE INCLUSÃO

A seguir são descritos os conteúdos dos arquivos /usr/include/curses.h e /usr/include/term.h. Estes arquivos podem ser incluídos no programa-fonte, com a utilização do comando #include do pré-processador c.

Arquivo /usr/include/curses.h

```
/*
 * Descricao   - Estruturas de dados, variaveis e macros do CURSES.
 *
 * /usr/include/curses.h                        #include <curses.h>
 *
 */

#ifndef _JAN

#define _JAN    1

/*
 *      Este arquivo define a estrutura de dados de uma janela (estrutura
 *      basica do curses), variaveis globais que podem ser usadas pelo
 *      usuario e pseudas funcoes do curses
 */

#include        <stdio.h>
#include        <stdlib.h>

#ifndef CUR_MACROS

#define _TEXTO( janela, y, x )  ( (janela) -> texto[ (y) ][ (x) ] )
#define INIC_ROL( janela )      ( (janela) -> inic_rol )
#define FIM_ROL( janela )       ( (janela) -> fim_rol )
#define LINHA_CUR( janela )     ( (janela) -> cursory )
#define COLUNA_CUR( janela )    ( (janela) -> cursorx )

#endif  CUR_MACROS

/*
 *      Definicao de um caractere para o curses:
 *        - byte menos significativo contem o caractere.
 *        - byte mais significativo contem os atributos (de video) deste
 *          caractere
 */
```

```
typedef unsigned short  caractere;

/*
 *      Estrutura de trecho de uma linha indica o trecho da linha (coluna
 *      inicial e final) que ainda nao foi atualizado. Se inic_trecho for
 *      igual a -1 esta linha ja foi atualizada.
 */

struct  t_trecho{
        short inic_trecho;
        short fim_trecho;
        };

/*
 *      Estrutura de uma janela (area retangular contendo texto).
 *
 *
 *      cursory e cursorx : indica a linha e a coluna onde se encontra o cursor
 *                          da janela

 *
 *      maximoy e maximox : indica o numero de linhas e de colunas desta janela
 *
 *      origemy e origemx : indica a linha e a coluna da tela (video) onde
 *                          comeca a janela (posicao 0,0 da janela)
 *
 *      inic_rol e fim_rol: indica a linha inicial e final da regiao de
 *                          rolamento
 *
 *      lin_mudada        : indica que so existe um caractere na janela a ser
 *                          atualizado. O seu valor e igual ao numero da linha
 *                          onde se encontra este caractere, Se for igual a -1
 *                          entao nao tem so um caractere a ser atualizado
 *
 *      lin_inserida      : semelhante a lin_mudada porem se refere a um
 *                          caractere inserido na janela
 *
 *      trecho            : aponta para um array que indica o inicio e o fim do
 *                          trecho de cada linha (ver a estrutura t_trecho)
```

```
 *
 *      desloca          : aponta para um array que indica o off_set de cada
 *                         linha em relacao a sua posicao anterior na janela
 *                         (este array e usado na otimizacao do scroll e/ou
 *                         delecao/insercao de linha por hardware)
 *
 *      texto            : area retangular contendo o texto da janela
 *
 *      atributos        : indica os atributos da de video janela (ver
 *                         mascaras de atributos de video)
 *
 *      janela_suja      : indica se a janela precisa ser atualizada ou nao
 *
 *      pad_sub_janela   : indica se esta estrutura e referente a uma janela
 *                         a uma sub_janela ou a um pad
 *
 *      abicomp_acent    : indica se os caracteres colocados na janela vao ser
 *                         acentuados ou nao e se as sequencias abnt7 serao
 *                         tratadas ou nao quando lidas atraves da janela
 *
 *      flags            : contem atributos da janela (ver mascaras para flags)
 *
 */

struct  janela{

        short   cursory, cursorx;
        short   maximoy, maximox;
        short   origemy, origemx;
        short   inic_rol, fim_rol;
        short   lin_mudada;
        short   lin_inserida;
        struct  t_;
        short   *desloca;
        caractere **texto;
        caractere atributos;
        short   janela_suja;
```

```
        short   pad_sub_janela; /*      = 0  para janela
                                 *      = 1  para sub_janela
                                 *      = 2  para pad
                                 */
        short   abicomp_acent:  /*
                                 *      1 bit para abicomp
                                 *      2 bit para acentuacao
                                 *      3 bit para SO
                                 */
        short   flags;

        };

typedef struct  janela  JANELA;

/*      MASCARAS PARA FLAGS      */

#define B_ROLAMENTO             01      /*indica se rolamento e permitido*/

#define B_APAGAMENTO            02      /*indica se a janela esta vazia*/

#define B_INS_DEL               04      /*indica se deve ser usado ins./del*/
#define B_CHAVE                 010     /*indica se devem ser usadas as t_ch */
#define B_POSICIONAMENTO        020     /*indica se o cursor deve ser mostr. */
#define B_LE_BLOQUEADA          040     /*indica se a leitura e bloqueada*/

 /*     MASCARAS PARA ABICOMP_ACENT      */

#define ABICOMP                 01 /*indica se deve tratar as sequencias */
#define ACENTUACAO              02 /*indica se deve ser acentuado os car.*/
#define B_SO                    04 /*indica se foi lido um shift_out*/

 /*     MASCARAS PARA PAD_SUB_JANELA     */

#define JAN     0               /*indica se e uma janela*/
#define SUBJAN  1               /*indica se e uma subjanela*/
#define PAD     2               /*indica se e um pad*/
```

```
/*      ATRIBUTOS DE VIDEO DA JANELA      */

#define A_STANDOUT      0000400
#define A_UNDERLINE     0001000
#define A_REVERSE       0002000
#define A_BLINK         0004000
#define A_DIM           0010000
#define A_BOLD          0020000
#define A_NORMAL        0000000

/*      ATRIBUTOS DE CARACTERE      */

#define A_ATRIBUTOS     0037400         /*mascara para pegar os atributos*/
#define A_TEXTO         0000377         /*mascara para pegar o texto*/
#define A_ACENTO        0000200         /*mascara para acentuacao*/
#define A_SEM_ACENTO    0177            /*mascara para pegar so 7 bits*/

/*      VARIAVEL PARA TRATAMENTO DE MAGIC_COOKIE        */

#define BRAN_MCG            (' ' | 0000200)

/*      VARIAVEIS GLOBAIS       */

extern  short   LINHAS, COLUNAS;        /*numero de linhas e colunas do
                                          terminal*/
extern  short   c_pendente;             /*indica se existe um caractere
                                          pendente na tela (ultima linha,
                                          ultima coluna da tela)*/
extern  JANELA  *stdscr, *curscr;       /*janelas padroes*/

extern  char    _cod_controle[][3];     /*tabela de codigo de controle para
                                          uso da rotina unctrl()*/
#define ERRO    (0)
#define OK      (1)
#define TRUE    (1)
#define FALSE   (0)
#define bool    char
```

```
/*      DECLARACAO DAS FUNCOES        */
JANELA  *newwin(), *subwin(), *newpad(), *initscr();
char    erasechar(), killchar(), *longname();
char    *_versao();
int     wgetch();
struct  _tab_ch *_adicione(),
        *_inic_tab_chave();
JANELA  *_cria_janela();

/*      PSEUDO FUNCOES PARA TELA PADRAO (stdscr)        */

# define        acent(flag)      wacent(stdscr, flag)
# define        addch(carac)     waddch(stdscr, carac)
# define        addstr(cadeia)   waddstr(stdscr, cadeia)
# define        attroff(atrib)   wattroff(stdscr,atrib)
# define        attron(atrib)    wattron(stdscr,atrib)
# define        attrset(atrib)   wattrset(stdscr,atrib)
# define        clear()          wclear(stdscr)
# define        clrtobot()       wclrtobot(stdscr)
# define        clrtoeol()       wclrtoeol(stdscr)
# define        delch()          wdelch(stdscr)
# define        deleteln()       wdeleteln(stdscr)
# define        erase()          werase(stdscr)
# define        getch()          wgetch(stdscr)
# define        getstr(cadeia)   wgetstr(stdscr, cadeia)
# define        inch()           winch(stdscr)
# define        insch(carac)     winsch(stdscr,carac)
# define        insertln()       winsertln(stdscr)
# define        modoabc(flag)    wmodoabc(stdscr,flag)
# define        move(y, x)       wmove(stdscr, y, x)
# define        nscroll( n )     wnscroll(stdscr, n)
# define        refresh()        wrefresh(stdscr)
# define        standout()       wstandout(stdscr)
# define        standend()       wstandend(stdscr)
```

```
# define   setscrreg(marg_sup,marg_inf)   wsetscrreg(stdscr, marg_sup, marg_inf)

/*      PSEUDO FUNCOES MV      */

#define mvwaddch(janela,y,x,carac)        (wmove(janela,y,x)==ERRO?ERRO: \
                                           waddch(janela,carac))
#define mvwaddstr(janela,y,x,cadeia)      (wmove(janela,y,x)==ERRO?ERRO: \
                                           waddstr(janela,cadeia))
#define mvwdelch(janela,y,x)              (wmove(janela,y,x)==ERRO?ERRO: \
                                           wdelch(janela))
#define mvwgetch(janela,y,x)              (wmove(janela,y,x)==ERRO?ERRO: \
                                           wgetch(janela))
#define mvwgetstr(janela,y,x,cadeia)      (wmove(janela,y,x)==ERRO?ERRO: \
                                           wgetstr(janela,cadeia))
#define mvwinch(janela,y,x)               (wmove(janela,y,x)==ERRO?ERRO: \
                                           winch(janela))
#define mvwinsch(janela,y,x,carac)        (wmove(janela,y,x)==ERRO?ERRO: \
                                           winsch(janela,carac))

/*      PSEUDO FUNCOES MV PARA TELA PADRAO      */

#define mvaddch(y,x,carac)      mvwaddch(stdscr,y,x,carac)
#define mvaddstr(y,x,cadeia)    mvwaddstr(stdscr,y,x,cadeia)
#define mvdelch(y,x)            mvwdelch(stdscr,y,x)
#define mvgetch(y,x)            mvwgetch(stdscr,y,x)
#define mvgetstr(y,x,cadeia)    mvwgetstr(stdscr,y,x,cadeia)
#define mvinch(y,x)             mvwinch(stdscr,y,x)
#define mvinsch(y,x,c)          mvwinsch(stdscr,y,x,c)

/*      PSEUDO FUNCOES      */

#define getyx(janela, y, x)     ( (y) = LINHA_CUR ( (janela) ), \
                                  (x) = COLUNA_CUR( (janela) ) )
#define winch(janela)           ( _TEXTO( (janela), LINHA_CUR ( (janela) ), \
                                          COLUNA_CUR( (janela) ) ) )
#define unctrl(carac)           (&_cod_controle[(unsigned)(carac)][0])
```

```
/*      definicoes das chaves        */
#define KEY_BREAK       0401         /* tecla break (nao confiavel) */
#define KEY_DOWN        0402         /* As quatro teclas com setas */
#define KEY_UP          0403
#define KEY_LEFT        0404
#define KEY_RIGHT       0405         /* ... */
#define KEY_HOME        0406         /* tecla HOME (seta para cima e esquerda)
                                      */
#define KEY_BACKSPACE   0407         /* retrocesso (nao confiavel) */
#define KEY_F0          0410         /* teclas de funcao, espaco para 64 */
#define KEY_F(n)        (KEY_F0+(n)) /* funcoes foi reservado */
#define KEY_DL          0510         /* deleta linha */
#define KEY_IL          0511         /* insere linha */
#define KEY_DC          0512         /* deleta caractere */
#define KEY_IC          0513         /* insere car. ou entra em modo de
                                        insercao */
#define KEY_EIC         0514         /* sai do modo de insercao de car. */
#define KEY_CLEAR       0515         /* limpa tela */
#define KEY_EOS         0516         /* limpa ate fim da tela */
#define KEY_EOL         0517         /* limpa ate fim da linha */
#define KEY_SF          0520         /* scroll de 1 linha para frente */
#define KEY_SR          0521         /* scroll de 1 linha para tras */
#define KEY_NPAGE       0522         /* proxima pagina */
#define KEY_PPAGE       0523         /* pagina anterior */
#define KEY_STAB        0524         /* seta tab */
#define KEY_CTAB        0525         /* zera tab */
#define KEY_CATAB       0526         /* zera todos os tabs */
#define KEY_ENTER       0527         /* Enter ou Send (nao confiavel) */
#define KEY_SRESET      0530         /* reset parcial (soft) (nao confiavel)
                                      */
#define KEY_RESET       0531         /* reset total (hard) (nao confiavel) */
#define KEY_PRINT       0532         /* imprime ou copia */
#define KEY_LL          0533         /* home em baixo (esq. em baixo) */
```

```
                                        /* configuracao da ilha */
                                        /*   a1   cima   a3   */
                                        /*   esq   b2   dir   */
                                        /*   c1  baixo   c3   */
#define KEY_A1          0534
#define KEY_A3          0535
#define KEY_B2          0536
#define KEY_C1          0537
#define KEY_C3          0540
#define MAX_CHAVES      40

/* compatibilizacao com libcurses */
#define WINDOW  JANELA
#define COLS    COLUNAS
#define LINES   LINHAS
#define ERR     ERRO
#define chtype  caractere
#define A_CHARTEXT      A_TEXTO
#define A_ATTRIBUTES    A_ATRIBUTOS


#endif _JAN
```

Arquivo /usr/include/term.h

```
/*
 * Descricao   - Estruturas de dados e macros do TERMINFO.
 *
 * /usr/include/term.h                    #include <term.h>
 *
 */

#ifndef auto_left_margin

#define auto_left_margin            CUR Auto_left_margin
#define auto_right_margin           CUR Auto_right_margin
#define beehive_glitch              CUR Beehive_glitch
#define ceol_standout_glitch        CUR Ceol_standout_glitch
#define eat_newline_glitch          CUR Eat_newline_glitch
#define erase_overstrike            CUR Erase_overstrike
#define generic_type                CUR Generic_type
#define hard_copy                   CUR Hard_copy
#define has_meta_key                CUR Has_meta_key
#define has_status_line             CUR Has_status_line
#define insert_null_glitch          CUR Insert_null_glitch
#define memory_above                CUR Memory_above
#define memory_below                CUR Memory_below
#define move_insert_mode            CUR Move_insert_mode
#define move_standout_mode          CUR Move_standout_mode
#define over_strike                 CUR Over_strike
#define status_line_esc_ok          CUR Status_line_esc_ok
#define teleray_glitch              CUR Teleray_glitch
#define tilde_glitch                CUR Tilde_glitch
#define transparent_underline       CUR Transparent_underline
#define xon_xoff                    CUR Xon_xoff
#define abicomp                     CUR Abicomp
#define abicomp_glitch              CUR Abicomp_glitch
#define columns                     CUR Columns
#define init_tabs                   CUR Init_tabs
#define lines                       CUR Lines
#define lines_of_memory             CUR Lines_of_memory
```

```
#define cursor_normal              CUR strs.Cursor_normal
#define cursor_right               CUR strs.Cursor_right
#define cursor_to_ll               CUR strs.Cursor_to_ll
#define cursor_up                  CUR strs.Cursor_up
#define cursor_visible             CUR strs.Cursor_visible
#define delete_character           CUR strs.Delete_character
#define delete_line                CUR strs.Delete_line
#define dis_status_line            CUR strs.Dis_status_line
#define down_half_line             CUR strs.Down_half_line
#define enter_alt_charset_mode     CUR strs.Enter_alt_charset_mode
#define enter_blink_mode           CUR strs.Enter_blink_mode
#define enter_bold_mode            CUR strs.Enter_bold_mode
#define enter_ca_mode              CUR strs.Enter_ca_mode
#define enter_delete_mode          CUR strs.Enter_delete_mode
#define enter_dim_mode             CUR strs.Enter_dim_mode
#define enter_insert_mode          CUR strs.Enter_insert_mode
#define enter_secure_mode          CUR strs.Enter_secure_mode
#define enter_protected_mode       CUR strs.Enter_protected_mode
#define magic_cookie_glitch        CUR Magic_cookie_glitch
#define padding_baud_rate          CUR Padding_baud_rate
#define virtual_terminal           CUR Virtual_terminal
#define width_status_line          CUR Width_status_line
#define back_tab                   CUR strs.Back_tab
#define bell                       CUR strs.Bell
#define carriage_return            CUR strs.Carriage_return
#define change_scroll_region       CUR strs.Change_scroll_region
#define clear_all_tabs             CUR strs.Clear_all_tabs
#define clear_screen               CUR strs.Clear_screen
#define clr_eol                    CUR strs.Clr_eol
#define clr_eos                    CUR strs.Clr_eos
#define column_address             CUR strs.Column_address
#define command_character          CUR strs.Command_character
#define cursor_address             CUR strs.Cursor_address
#define cursor_down                CUR strs.Cursor_down
#define cursor_home                CUR strs.Cursor_home
#define cursor_invisible           CUR strs.Cursor_invisible
#define cursor_left                CUR strs.Cursor_left
#define cursor_mem_address         CUR strs.Cursor_mem_address
```

```
#define enter_reverse_mode          CUR strs.Enter_reverse_mode
#define enter_standout_mode         CUR strs.Enter_standout_mode
#define enter_underline_mode        CUR strs.Enter_underline_mode
#define erase_chars                 CUR strs.Erase_chars
#define exit_alt_charset_mode       CUR strs.Exit_alt_charset_mode
#define exit_attribute_mode         CUR strs.Exit_attribute_mode
#define exit_ca_mode                CUR strs.Exit_ca_mode
#define exit_delete_mode            CUR strs.Exit_delete_mode
#define exit_insert_mode            CUR strs.Exit_insert_mode
#define exit_standout_mode          CUR strs.Exit_standout_mode
#define exit_underline_mode         CUR strs.Exit_underline_mode
#define flash_screen                CUR strs.Flash_screen
#define form_feed                   CUR strs.Form_feed
#define from_status_line            CUR strs.From_status_line
#define init_1string                CUR strs.Init_1string
#define init_2string                CUR strs.Init_2string
#define init_3string                CUR strs.Init_3string
#define init_file                   CUR strs.Init_file
#define insert_character            CUR strs.Insert_character
#define insert_line                 CUR strs.Insert_line
#define insert_padding              CUR strs.Insert_padding
#define key_backspace               CUR strs.Key_backspace
#define key_catab                   CUR strs.Key_catab
#define key_clear                   CUR strs.Key_clear
#define key_ctab                    CUR strs.Key_ctab
#define key_dc                      CUR strs.Key_dc
#define key_dl                      CUR strs.Key_dl
#define key_down                    CUR strs.Key_down
#define key_eic                     CUR strs.Key_eic
#define key_eol                     CUR strs.Key_eol
#define key_eos                     CUR strs.Key_eos
#define key_f0                      CUR strs.Key_f0
#define key_f1                      CUR strs.Key_f1
#define key_f10                     CUR strs.Key_f10
#define key_f2                      CUR strs.Key_f2
#define key_f3                      CUR strs.Key_f3
#define key_f4                      CUR strs.Key_f4
#define key_f5                      CUR strs.Key_f5
```

```
#define key_f6                  CUR strs.Key_f6
#define key_f7                  CUR strs.Key_f7
#define key_f8                  CUR strs.Key_f8
#define key_f9                  CUR strs.Key_f9
#define key_home                CUR strs.Key_home
#define key_ic                  CUR strs.Key_ic
#define key_il                  CUR strs.Key_il
#define key_left                CUR strs.Key_left
#define key_ll                  CUR strs.Key_ll
#define key_npage               CUR strs.Key_npage
#define key_ppage               CUR strs.Key_ppage
#define key_right               CUR strs.Key_right
#define key_sf                  CUR strs.Key_sf
#define key_sr                  CUR strs.Key_sr
#define key_stab                CUR strs.Key_stab
#define key_up                  CUR strs.Key_up
#define keypad_local            CUR strs.Keypad_local
#define keypad_xmit             CUR strs.Keypad_xmit
#define lab_f0                  CUR strs.Lab_f0
#define lab_f1                  CUR strs.Lab_f1
#define lab_f10                 CUR strs.Lab_f10
#define lab_f2                  CUR strs.Lab_f2
#define lab_f3                  CUR strs.Lab_f3
#define lab_f4                  CUR strs.Lab_f4
#define lab_f5                  CUR strs.Lab_f5
#define lab_f6                  CUR strs.Lab_f6
#define lab_f7                  CUR strs.Lab_f7
#define lab_f8                  CUR strs.Lab_f8
#define lab_f9                  CUR strs2.Lab_f9
#define meta_off                CUR strs2.Meta_off
#define meta_on                 CUR strs2.Meta_on
#define newline                 CUR strs2.Newline
#define pad_char                CUR strs2.Pad_char
#define parm_dch                CUR strs2.Parm_dch
#define parm_delete_line        CUR strs2.Parm_delete_line
#define parm_down_cursor        CUR strs2.Parm_down_cursor
#define parm_ich                CUR strs2.Parm_ich
#define parm_index              CUR strs2.Parm_index
#define parm_insert_line        CUR strs2.Parm_insert_line
```

```
#define save_cursor          CUR strs2.Save_cursor
#define scroll_forward       CUR strs2.Scroll_forward
#define scroll_reverse       CUR strs2.Scroll_reverse
#define set_attributes       CUR strs2.Set_attributes
#define set_tab              CUR strs2.Set_tab
#define set_window           CUR strs2.Set_window
#define tab                  CUR strs2.Tab
#define to_status_line       CUR strs2.To_status_line
#define underline_char       CUR strs2.Underline_char
#define up_half_line         CUR strs2.Up_half_line
#define init_prog            CUR strs2.Init_prog
#define key_a1               CUR strs2.Key_a1
#define key_a3               CUR strs2.Key_a3
#define key_b2               CUR strs2.Key_b2
#define key_c1               CUR strs2.Key_c1
#define key_c3               CUR strs2.Key_c3
#define prtr_non             CUR strs2.Prtr_non
#define e_e_alt              CUR strs2.E_e_alt
#define e_s_alt              CUR strs2.E_s_alt
#define parm_left_cursor     CUR strs2.Parm_left_cursor
#define parm_right_cursor    CUR strs2.Parm_right_cursor
#define parm_rindex          CUR strs2.Parm_rindex
#define parm_up_cursor       CUR strs2.Parm_up_cursor
#define pkey_key             CUR strs2.Pkey_key
#define pkey_local           CUR strs2.Pkey_local
#define pkey_xmit            CUR strs2.Pkey_xmit
#define print_screen         CUR strs2.Print_screen
#define prtr_off             CUR strs2.Prtr_off
#define prtr_on              CUR strs2.Prtr_on
#define repeat_char          CUR strs2.Repeat_char
#define reset_1string        CUR strs2.Reset_1string
#define reset_2string        CUR strs2.Reset_2string
#define reset_3string        CUR strs2.Reset_3string
#define reset_file           CUR strs2.Reset_file
#define restore_cursor       CUR strs2.Restore_cursor
#define row_address          CUR strs2.Row_address
```

```
#define s_e_alt              CUR strs2.S_e_alt
#define s_s_alt              CUR strs2.S_s_alt
#define map_ent              CUR strs2.Map_Ent
#define map_saida            CUR strs2.Map_Saida
#define vers_i_t             CUR strs2.Vers_i_t
typedef char *charptr;

struct strs {
   charptr
        Back_tab,
        Bell,
        Carriage_return,
        Change_scroll_region,
        Clear_all_tabs,
        Clear_screen,
        Clr_eol,
        Clr_eos,
        Column_address,
        Command_character,
        Cursor_address,
        Cursor_down,
        Cursor_home,
        Cursor_invisible,
        Cursor_left,
        Cursor_mem_address,
        Cursor_normal,
        Cursor_right,
        Cursor_to_ll,
        Cursor_up,
        Cursor_visible,
        Delete_character,
        Delete_line,
        Dis_status_line,
        Down_half_line,
        Enter_alt_charset_mode,
        Enter_blink_mode,
        Enter_bold_mode,
```

Enter_ca_mode,
Enter_delete_mode,
Enter_dim_mode,
Enter_insert_mode,
Enter_secure_mode,
Enter_protected_mode,
Enter_reverse_mode,
Enter_standout_mode,
Enter_underline_mode,
Erase_chars,
Exit_alt_charset_mode,
Exit_attribute_mode,
Exit_ca_mode,
Exit_delete_mode,
Exit_insert_mode,
Exit_standout_mode,
Exit_underline_mode,
Flash_screen,
Form_feed,
From_status_line,
Init_1string,
Init_2string,
Init_3string,
Init_file,
Insert_character,
Insert_line,
Insert_padding,
Key_backspace,
Key_catab,
Key_clear,
Key_ctab,
Key_dc,
Key_dl,
Key_down,
Key_eic,
Key_eol,
Key_eos,
Key_f0,
Key_f1,

```
Key_sf,
Key_sr,
Key_stab,
Key_up,
Keypad_local,
Keypad_xmit,
Lab_f0,
Lab_f1,
Lab_f10,
Lab_f2,
Lab_f3,
Lab_f4,
Lab_f5,
Lab_f6,
Lab_f7,
Lab_f8;
Key_f10,
Key_f2,
Key_f3,
Key_f4,
Key_f5,
Key_f6,
Key_f7,
Key_f8,
Key_f9,
Key_home,
Key_ic,
Key_il,
Key_left,
Key_ll,
Key_npage,
Key_ppage,
Key_right,
```

```
};
struct strs2 {
    charptr
        Lab_f9,
        Meta_off,
        Meta_on,
        Newline,
        Pad_char,
        Parm_dch,
        Parm_delete_line,
        Parm_down_cursor,
        Parm_ich,
        Parm_index,
        Parm_insert_line,
        Parm_left_cursor,
        Parm_right_cursor,
        Parm_rindex,
        Parm_up_cursor,
        Pkey_key,
        Pkey_local,
        Pkey_xmit,
        Print_screen,
        Prtr_off,
        Prtr_on,
        Repeat_char,
        Reset_1string,
        Reset_2string,
        Reset_3string,
        Reset_file,
        Restore_cursor,
        Row_address,
        Save_cursor,
        Scroll_forward,
        Scroll_reverse,
        Set_attributes,
        Set_tab,
        Set_window,
        Tab,
        To_status_line,
        Underline_char,
```

```
        Up_half_line,
        Init_prog,
        Key_a1,
        Key_a3,
        Key_b2,
        Key_c1,
        Key_c3,
        Prtr_non,
        E_e_alt,
        E_s_alt,
        S_e_alt,
        S_s_alt,
        Map_Ent,
        Map_Saida,
        Vers_i_t;
};

struct term {
    char
        Auto_left_margin,
        Auto_right_margin,
        Beehive_glitch,
        Ceol_standout_glitch,
        Eat_newline_glitch,
        Erase_overstrike,
        Generic_type,
        Hard_copy,
        Has_meta_key,
        Has_status_line,
        Insert_null_glitch,
        Memory_above,
        Memory_below,
        Move_insert_mode,
        Move_standout_mode,
        Over_strike,
        Status_line_esc_ok,
        Teleray_glitch,
        Tilde_glitch,
        Transparent_underline,
```

```
        Xon_xoff,
        Abicomp,
        Abicomp_glitch;
    short
        Columns,
        Init_tabs,
        Lines,
        Lines_of_memory,
        Magic_cookie_glitch,
        Padding_baud_rate,
        Virtual_terminal,
        Width_status_line;
        struct strs strs;
        struct strs2 strs2;
        short DescrArq;

};



#endif auto_left_margin

# define CUR    term_corrente->infoterm->

#ifndef TELA

#ifdef EIX
#include        <sys/types.h>
#endif
#include        <termio.h>
#ifndef _JAN
#include        <curses.h>
#endif  _JAN

/*      ESTRUTURA PARA OTIMIZACAO DE BRANCOS NA TELA      */
```

```
struct  _otim_brancos{

        short nbrancos,
            resto_brancos;

        } ;

/*      ESTRUTURA DA TELA VIRTUAL       */

struct  _virtual{

        JANELA **id_jan;

        struct  _otim_brancos   *otim_brancos;
        char    *bran_ant;
        short   *_ha_mud_atrib;

        short   *del_custos;
        short   *ins_custos;

        JANELA  *jan_virtual;

 } ;

/*      ESTRUTURA TABELA DE KEYPAD      */

struct  _tab_ch{

        char    cadeia[16];

        int     numero;

        } ;

/*      ESTRUTURA DO TERMINAL       */

struct  _tela{

        FILE    *fp_entrada,
```

```
          *fp_saida;

        int     fd_typeahead;

        short   cur_y_fisico,
                cur_x_fisico;

        JANELA  *tela_stdscr;

        struct  _virtual        *tela_virtual;

        struct  termio  prog_mode,
                        shell_mode,
                        buffer_tty;

        struct  term    *infoterm;

        struct  _tab_ch *ap_keypad;

        char    fila_entrada[ 30 ];

        short   cur_invisivel,
                estou_prog,
                estou_echo,
                estou_cbreak,
                estou_delay,
                estou_keypad,
                fiz_savetty;
        caractere _atrib_term;

        };

typedef struct  _tela TELA;

#define MODO_PROG(x)          (term_corrente->prog_mode.x)
#define MODO_SH(x)            (term_corrente->shell_mode.x)

/*      FUNCOES DO CURSES QUE RETORNAM UM APONTADOR PARA ESTRUTURA _tela  */

extern  TELA    *term_corrente;
TELA    *newterm(), *set_term();
#endif  TELA
```

IMPRESSO NA GRÁFICA DA COBRA COMPUTADORES

COMPUTADORES E SISTEMAS BRASILEIROS S.A.

CARTÃO DE CADASTRO E COMENTÁRIOS DO USUÁRIO

OPERAÇÃO CURSES-TERMINFO/SOX

UX0189-02.0
abril/1989

Prezado leitor:

Suas críticas, comentários e sugestões são valiosos instrumentos para que a COBRA possa, cada vez mais, aprimorar a documentação de seus produtos. Solicitamos, portanto, sua colaboração em preencher este cartão, colocando-o em qualquer caixa de correio (porte pago). Obrigado.

AVALIAÇÃO DO MANUAL

Nos itens que se seguem, indique o conceito aplicável, comentando se assim o desejar:

CONCEITOS: • Excelente (E) • Bom (B) • Regular (R) • Precisa melhorar (P) • Insuficiente/Inadequado (I)

ITEM	CONCEITO
APRESENTAÇÃO DO MANUAL (Encadernação, acabamento, formato)	
QUALIDADE GRÁFICA (Tipo de impressão, diagramação, papel utilizado)	
CLAREZA DAS INFORMAÇÕES (Linguagem empregada, facilidade de compreensão do texto)	
ABORDAGEM DOS ASSUNTOS (Organização, precisão e objetividade)	

ITEM	CONCEITO
DESENHOS, ILUSTRAÇÕES E EXEMPLOS (Suficiência, adequabilidade e clareza)	
UTILIDADE (Adequação à utilização pretendida, facilidade de consulta)	
EDITORAÇÃO E REVISÃO (Existência de falhas de impressão, paginação, erros ortográficos ou gramaticais)	
QUALIDADE GERAL DO MANUAL (Avaliação global do usuário)	

COMENTÁRIOS E SUGESTÕES

IDENTIFICAÇÃO DO USUÁRIO

Nome:	Profissão:
Empresa:	Cargo:
Endereço p/correspondência:	CEP:
	Tel.:

Av. Comandante Guaranys, 447 - 22785 - Rio de Janeiro - RJ
(021) 342-9393 - Telex (021) 22420

COLE ESTA ABA NO VERSO DO CARTÃO

ISR - 52 - 850/87
UP APT PRES. VARGAS
DR-RJ

CARTA RESPOSTA COMERCIAL

NÃO É NECESSÁRIO SELAR

O SELO SERÁ PAGO POR

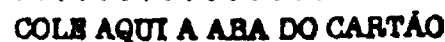
COBRA COMPUTADORES E SISTEMAS BRASILEIROS S/A
(DIV. ASSIST. SOFTWARE - DAS)

20299 RIO DE JANEIRO - RJ

CEP

Endereço:

Remetente:



cobra
Computadores e Sistemas Brasileiros S.A.