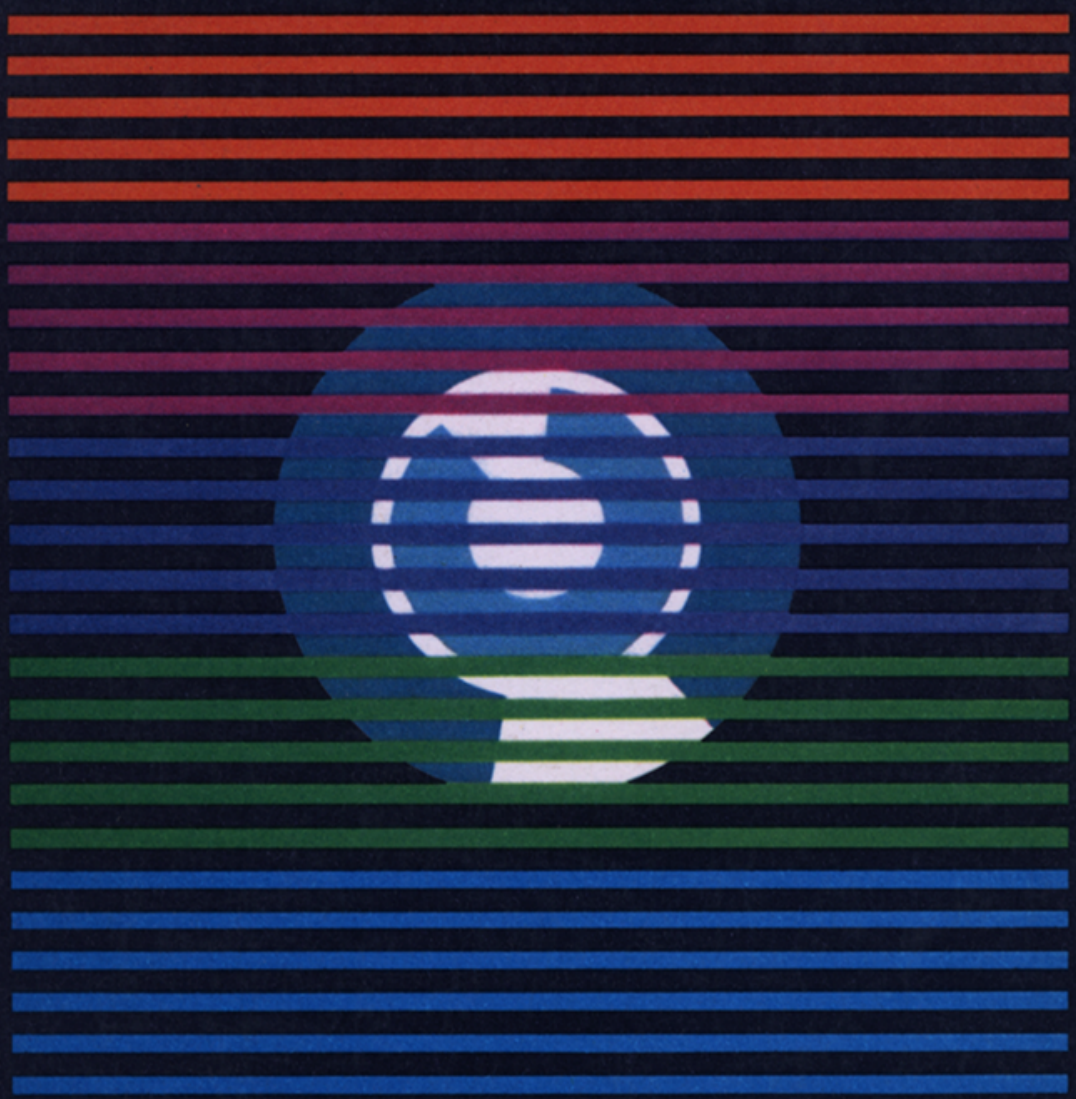
Aplicacoes de Computadores

Programação Estruturada em Cobol

ANIBAL P. CARDOSO



LIVROS TÉCNICOS E CIENTÍFICOS EDITORA

SÉRIE "APLICAÇÕES DE COMPUTADORES"

Coordenação técnica

DONALDO DE SOUZA DIAS

Já lançados:

COMPUTADOR, SOCIEDADE E DESENVOLVIMENTO
– Martin/Norman
INTRODUÇÃO À PROGRAMAÇÃO LINEAR
– Puccini
PROJETO DE SISTEMAS DE PROCESSAMENTO DE DADOS
– Dias/Gazzaneo
PROGRAMAÇÃO MODULAR
– Maynard
PROGRAMAÇÃO COBOL
– Alex Bastos
PROGRAMAÇÃO PL-1
– Sá
REDES DE COMUNICAÇÃO DE DADOS
– Tarouco
A ORGANIZAÇÃO DA FUNÇÃO DE PROCESSAMENTO DE DADOS
– Withington
PROCESSAMENTO INTERATIVO – A LINGUAGEM DE PROGRAMAÇÃO APL
– Zimmermann
PROGRAMAÇÃO ESTRUTURADA EM COBOL
– Cardoso

PROGRAMAÇÃO ESTRUTURADA EM COBOL

PROGRAMAÇÃO ESTRUTURADA EM COBOL

ANÍBAL PEREIRA CARDOSO

Professor da PUC/RS e da UNISINOS
Assessor Técnico da Caixa Econômica Estadual/RS
Mestre em Informática pela PUC/RJ

RIO DE JANEIRO
SÃO PAULO

LIVROS TÉCNICOS E CIENTÍFICOS EDITORA

Para a composição deste livro foi usado o Press Roman.
Marília Martins do Amaral funcionou como revisora do texto.
O *lay-out* e a arte-final da capa são de AG Comunicação Visual Assessoria e Projetos Ltda.
Série **"Aplicações de Computadores"**,
Coordenador da área de Computação na LTC: professor Donaldo de Souza Dias.

CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ.

Cardoso, Aníbal Pereira.
C26p Programação estruturada em COBOL / Aníbal Pereira Cardoso. – Rio de Janeiro: LTC – Livros Técnicos e Científicos, 1981.

Bibliografia.
Apêndices.

1. COBOL (Linguagem de programação para computadores) I. Título.

CDD – 001.6424
81-0545 CDU – 800.92COBOL

ISBN: 85-216-0163-8

Direitos reservados por:
LTC – LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.
Av. Venezuela, 163 – 20220
Rio de Janeiro, RJ
1981
Impresso no Brasil

A Ada e Sabrina

Prefácio

Recordo-me vividamente das discussões e conversas, nas salas e corredores do Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, onde se debatia a então emergente técnica da *Programação Estruturada*. Foi neste ambiente de entusiasmo acadêmico que o Prof. Aníbal Pereira Cardoso escreveu, em 1975, a tese de mestrado que deu origem ao excelente texto que agora está sendo publicado – *Programação Estruturada em COBOL*.

A comunidade brasileira de processamento de dados – alunos e profissionais – precisava deste texto. Inúmeras cópias da tese do Prof. Aníbal Cardoso existem em centro de processamento de dados do país. Era a única bibliografia em português sobre o assunto. Agora, aqui está o livro, revisto e atualizado, que permitirá a assimilação desta moderna técnica por parte de alunos e profissionais brasileiros.

A idéia da *Programação Estruturada* parece ter surgido da famosa carta do Professor E. W. Dijkstra, publicada na *Comunicações da ACM* de março de 1968. Naquela carta, ele alertava que declarações GO TO eram potencialmente perigosas para o estado mental dos programadores, que necessitavam testar códigos complexos e entrelaçados.

A base teórica da *Programação Estruturada* é atribuída ao trabalho de Böhm e Jacopini, datado de 1966, onde eles mostram que é possível escrever qualquer programa usando-se apenas três estruturas básicas: seqüência, seleção e controle de repetição. Usando-se apenas estas três construções, é possível escrever programas que podem ser lidos seqüencialmente, sem necessidade de se voltar freqüentemente a partes anteriores do programa.

O uso da *Programação Estruturada* reduz a complexidade dos programas e aumenta a sua clareza, possibilitando a diminuição do custo.

A complexidade de um programa dificulta não só o seu desenvolvimento, mas também a sua manutenção. Quando temos de modificar um programa, devido à correção de um erro ou à inclusão de uma melhoria, a sua complexidade torna difícil o entendimento.

A clareza de um programa representa a facilidade com que uma pessoa que não o conhece é capaz de determinar o que ele faz e como ele opera. A falta de clareza de um programa também dificulta o seu desenvolvimento e manutenção.

Este livro é uma iniciativa importante. Não só pelo seu valor intrínseco, mas também pela contribuição que poderá dar à melhor utilização dos recursos computacionais brasileiros. Programas simples e claros resultam num aproveitamento mais eficiente de nossos talentos de programação e, conseqüentemente, em menores custos.

Donaldo de Souza Dias

Setembro de 1981

Apresentação

A Programação Estruturada tem despertado grande interesse na comunidade de Processamento de Dados do Brasil, desde o seu surgimento. O ambiente acadêmico foi o palco de suas primeiras manifestações, há cerca de meia década atrás, e logo após, a técnica popularizou-se nos CPD's comerciais. Todavia, o acervo bibliográfico de Computação, em língua portuguesa, não recebeu a dose proporcional de obras sobre a Programação Estruturada, como era de esperar.

Pretendemos, com o lançamento deste livro, preencher, pelo menos em parte, esta lacuna, colocando à disposição dos professores, estudantes e profissionais de programação um texto didático e acessível, capaz de fornecer os fundamentos teóricos da Programação Estruturada e sugerir a forma de utilizá-la adequadamente.

A obra divide-se em duas partes. A primeira apresenta a descrição da Programação Estruturada, constituindo-se na base teórica indispensável a quem pretende utilizar a técnica. É por demais útil ao aprendizado nas Universidades ou no próprio ambiente de programação comercial.

A segunda parte sugere uma forma de utilização da Programação Estruturada em COBOL, orientada pela teoria apresentada na primeira parte, e enriquecida com a descrição, passo-a-passo, de um programa tipicamente comercial.

Com estes conteúdos, o livro atinge sua pretensão. Porém, o leitor interessado no uso da Programação Estruturada "em equipes", encontrará no Apêndice C a descrição das metodologias intituladas, em inglês, *Chief Programmer Teams* e *Structured Walk-throughs.*

Sendo esta obra uma evolução de nossa tese de mestrado intitulada *Programação Estruturada, Equipes de Programação e COBOL*, defendida em 1975 na PUC-Rio de Janeiro, seus principais conteúdos foram escritos naquela oportunidade. Cabe, portanto, lembrar que continuamos gratos a todas as pessoas e instituições que colaboraram, incentivaram ou patrocinaram a elaboração daquele trabalho, destacando o Prof. Eugênio Vilar Pires, os professores, funcionários e colegas do Departamento de Informática da PUC-RJ, a PUC-RS e a Caixa Econômica Estadual do Rio Grande do Sul.

Nossa especial gratidão ao Prof. Donaldo de Souza Dias, a cuja colaboração e incentivo emprestados desde longa data uniu-se a influência decisiva para a edição deste livro.

Aníbal Cardoso

Porto Alegre, abril de 1981

Sumário

Parte I

Programação Estruturada

Capítulo 1

Introdução

O surgimento da Programação Estruturada como uma nova técnica de programação situa-se no período subseqüente ao lançamento dos primeiros computadores de terceira geração.

Nessa época, como conseqüência dos progressos de *hardware*, a atividade de programação sofria algumas mudanças profundas em suas características que precisavam ser assimiladas pelos programadores.

O conflito confiabilidade X extensão é a novidade que merece maior destaque. Equipamentos cada vez mais potentes proporcionavam a oportunidade de se submeter problemas mais extensos à solução via computador. Derivava daí uma dificuldade direta para o programador: escrever programas corretos de grande porte. Tornava-se necessário, então, dotar a atividade de programação de uma sistemática de resolução de problemas complexos, o que não era oferecido pelas técnicas então conhecidas.

A produtividade dos programadores dentro dos CPD's também sofria alterações. Programar já não era apenas desenvolver novos programas, mas também alterar programas já prontos. E o componente *manutenção* contribuía substancialmente para a baixa da produtividade. Com efeito, a produtividade do trabalho de manutenção depende mais de certas qualidades do programa, tais como adaptabilidade e boa documentação, do que do programador que o está alterando. Como estas qualidades devem ser introduzidas durante a criação do programa, a atividade de manutenção é fortemente influenciada pela técnica usada durante o seu desenvolvimento.

De um modo geral, observamos, no período, uma preocupação maior com a própria atividade de programar; da forma como é executada depende a obtenção de um produto útil (correto) e com qualidades.

A Programação Estruturada surgiu da necessidade de oferecer ao programador uma ferramenta capaz de auxiliá-lo a manipular estas novas características de seu trabalho. Os objetivos da técnica, apresentados a seguir, consolidam esta idéia.

1.1 – OBJETIVOS DA TÉCNICA

A Programação Estruturada constitui-se numa técnica que auxilia o programador a construir programas mais confiáveis, mais legíveis, de maneira disciplinada e racional.

São quatro os objetivos específicos que são atingidos pelo programador com o uso da Programação Estruturada:

- obtenção de programas mais corretos;
- domínio da extensão;
- disciplina do raciocínio; e
- incremento de certas qualidades externas dos programas.

1.1.1 – Obtenção de Programas mais Corretos

A correção de um programa é uma condição indispensável para a sua utilidade. Na produção de programas estruturados, o programador é incentivado a certificar-se da correção paralelamente ao desenvolvimento.

1.1.2 – Domínio da Extensão

Os problemas que se apresentam ao programador têm aumentado de extensão. A dificuldade da mente humana para dominar um grande número de variáveis ao mesmo tempo torna-se uma barreira para o desenvolvimento de grandes programas. Uma das maneiras eficientes de remover esta barreira é a divisão do grande problema em problemas menores, de modo que a solução destes leve à solução daquele. O uso da Programação Estruturada facilita este tipo de procedimento através do desenvolvimento por refinamentos sucessivos, como veremos adiante.

1.1.3 – Disciplina do Raciocínio

É reconhecida a influência que a linguagem falada exerce sobre a nossa maneira de pensar. Por outro lado, a atividade de programar exige um trabalho mental bastante grande na escolha das melhores alternativas para a resolução de problemas que utilizam uma linguagem executável. O uso da Programação Estruturada proporciona a organização do raciocínio através de utilização disciplinada de certas construções semânticas padronizadas e de critérios para o desenvolvimento de soluções.

1.1.4 – Incremento de Certa Qualidades Externas dos Programas

Os objetivos específicos anteriores dizem mais respeito à ajuda que a técnica oferece ao programador para melhor encaminhar seu trabalho e para prover seu programa com qualidades internas. Como conseqüência desta prática, certas qualidades externas são incrementadas automaticamente.

Capítulo 2

Estruturação Básica

Chamamos de *Estruturação Básica* ao suporte semântico e matemático sobre o qual se assentam as idéias da Programação Estruturada.

Este componente se subdivide, por sua vez, em três partes: a primeira faz um estudo isolado de seis estruturas de controle, descrevendo sua semântica; a segunda mostra como estas estruturas podem ser integradas na composição de algoritmos; a terceira, fundamento matemático da Programação Estruturada, não será estudada aqui por fugir ao escopo deste livro.

2.1 – ESTRUTURAS DE CONTROLE

Apresentamos a seguir as principais estruturas de controle com as quais podemos escrever quaisquer programas. São subdivididas, segundo o tipo de fluxo de controle, em seqüência, seleção e repetição.

2.1.1 – Seqüência

A seqüência é a estrutura de controle mais simples. Ela representa uma computação decomposta em um número finito de ações seqüenciais. Seu fluxograma é dado na Fig. 2.1, onde $a_i, i = 1, 2, \ldots, n$ são ações ou processos.

Exemplo.

```
a := x + y;
v := y;
print r, y;
```

2.1.2 – Seleção

As estruturas de controle de seleção traduzem a escolha entre caminhos alternativos. Esta escolha é feita baseada no teste de uma condição que define o caminho a ser seguido pelo fluxo de controle.

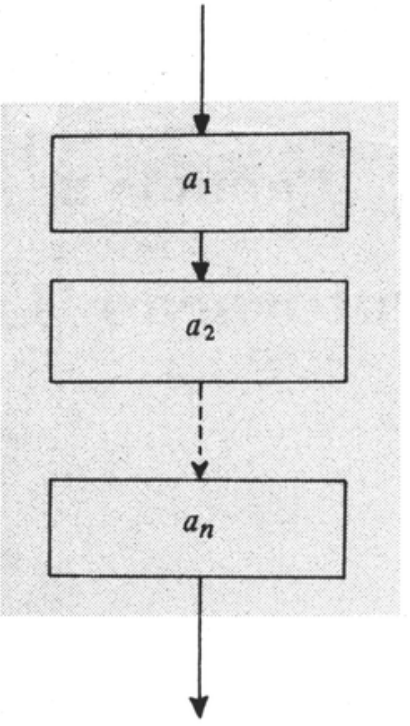


Fig. 2.1

Existem dois tipos de estruturas de seleção, quanto ao número de alternativas.

Seleção com duas alternativas. Esta estrutura se caracteriza por um comando de transferência que avalia uma expressão lógica e cujo resultado, portanto, pode ser VERDADEIRO ou FALSO. Dependendo do resultado da avaliação, uma, e somente uma, das alternativas será executada. Seu fluxograma é dado na Fig. 2.2 e sua forma escrita é

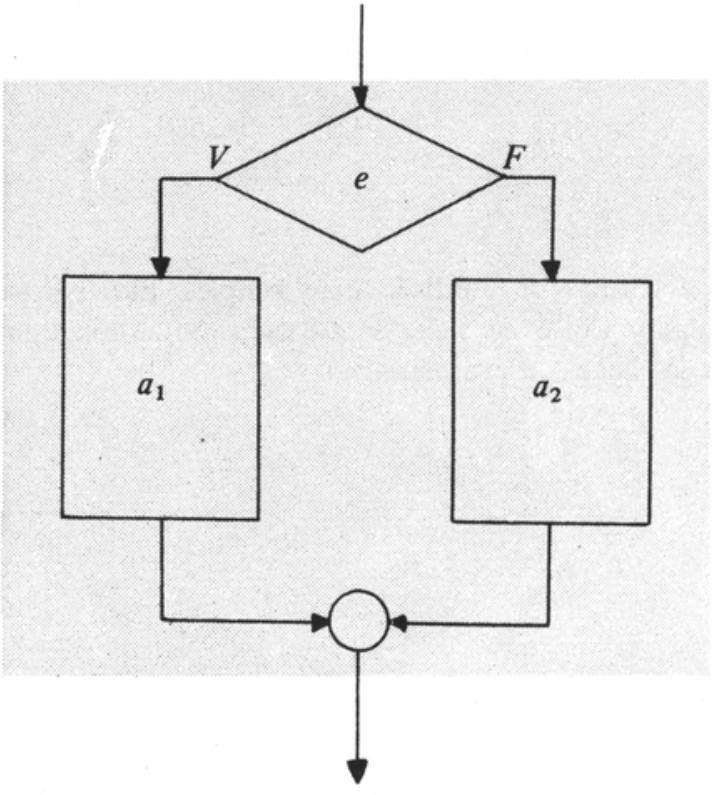


Fig. 2.2

```
if e
   then
      a1
   else
      a2
fi
```

onde e é uma expressão lógica, a_1 e a_2 são ações ou processos.

Esta estrutura recebe o nome de IF-THEN-ELSE.

Um tipo particular de IF-THEN-ELSE é o IF-THEN. Neste caso, embora a escolha se faça entre dois caminhos alternativos, o caminho gerado pelo resultado FALSO da expressão lógica não possui nenhuma ação a ser executada, mas é apenas um desvio para a saída da estrutura (Fig. 2.3).

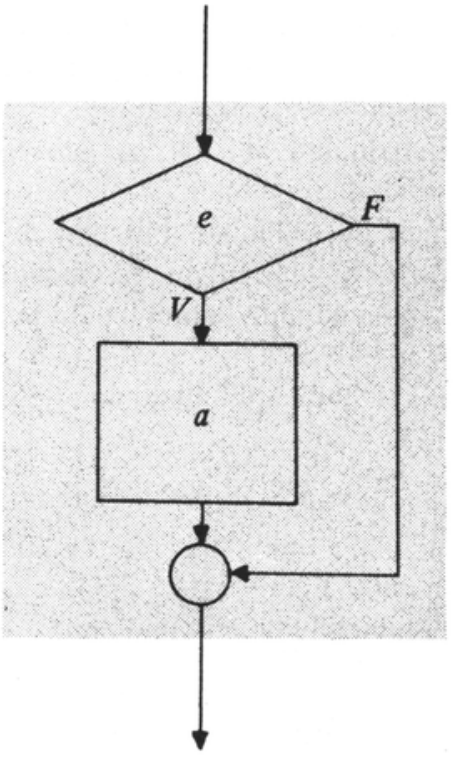


Fig. 2.3

```
if e
   then
      a
fi
```

Exemplos. a)

```
if a > b
   then
      a := a - b;
   else
      b := b - a;
fi;
```

```
b) if x ≠ y
      then
         x: = z +2;
         y: = z ** 2;
         print x, y;
      else
         print z;
   fi;

c) if k = 0
      then
         call rotina;
   fi;
```

O exemplo (*a*) apresenta um caso de IF-THEN-ELSE, onde cada alternativa executa apenas uma ação, isto é, uma instrução executável na linguagem usada.

No exemplo (*b*), a alternativa THEN executa um processo composto de três instruções, enquanto a alternativa ELSE executa uma ação apenas.

No exemplo (*c*), temos um caso de IF-THEN com uma única instrução.

Seleção com três ou mais alternativas. Quando a escolha do caminho a ser seguido pelo fluxo de controle envolve mais do que duas alternativas, é conveniente o uso da estrutura CASE. A escolha é feita dependendo do valor de uma expressão do programa e apenas uma das alternativas será executada.

O fluxograma da estrutura CASE é apresentado na Fig. 2.4.

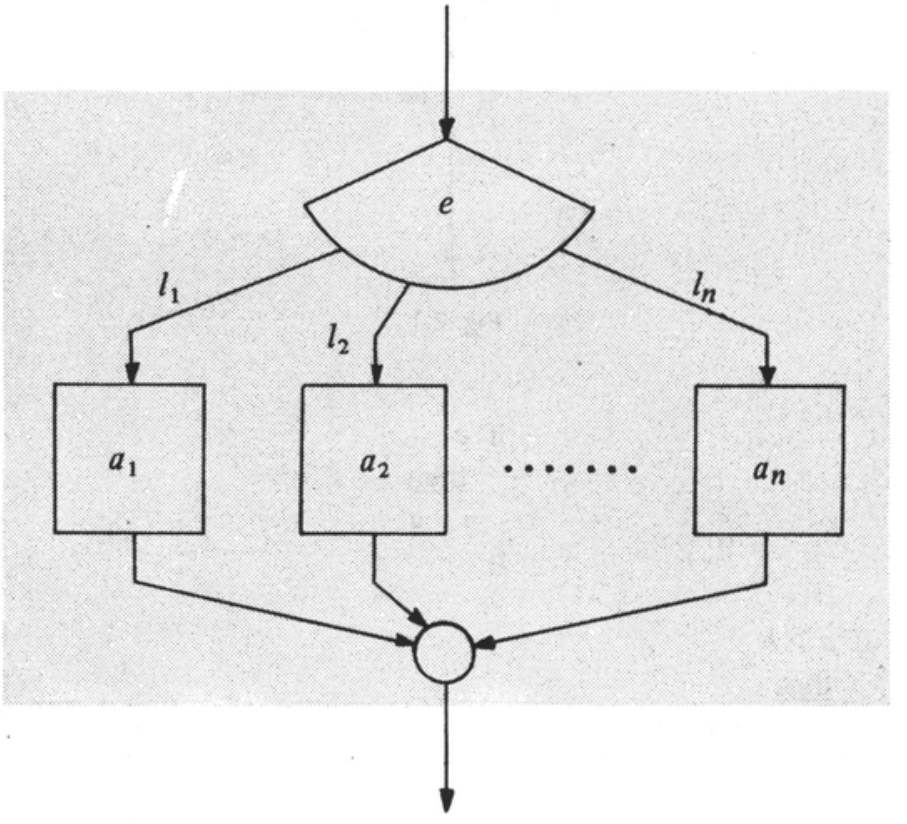


Fig. 2.4

```
case e
   l1:a1
   l2:a2
    .
    .
    .
   ln:an
esac
```

onde e é uma expressão aritmética, l_i, $i = 1, 2, 3, \ldots, n$, são rótulos (*labels*) das diversas alternativas e a_i, $i = 1, 2, 3, \ldots, n$ são ações ou processos. Quando a expressão e tem valor k, então a ação ou processo rotulado por l_k é selecionado e executado e, depois, o fluxo de controle vai para a saída da estrutura.

Quando o mesmo processo deve ser executado para mais de um valor da expressão e, isto é, $a_i = a_j = \ldots = a_k$, então a notação pode ser abreviada para

$$l_i{:}l_j{:}\ldots{:}l_k{:}a_i.$$

Exemplo.

```
case j;
   L1: a:=b + c;
   L2: read m, n;
       i:=i + 1;
   L3: L4: a:=a + 1;
           print c;
   L5: print m, n;
esac;
```

Observação. Existem vários outros tipos de CASE's. Por exemplo, o CASE sugerido por Wirth e Hoare, em 1966, para modificações na linguagem ALGOL tem a seguinte forma:

```
case e of
  begin
    S1;
    S2;
     .
     .
     .
    Sn;
  end
```

que causa a execução do comando S_e, sendo e uma expressão aritmética cujo resultado de sua avaliação é um inteiro entre 1 e n. Neste tipo de CASE, o comando executado é escolhido por sua posição ordinal. A introdução de *labels* para evitar esta forma de escolha resultou num novo tipo:

```
case e
  of
     1:      S1;
     2:      S2;
     3:4:5:  S3;
     9:      S4;
     6:7:8:  S5;
  end
```

2.1.3 – Repetição

As estruturas de repetição são compostas de dois elementos: um *processo* ou *ação* e um *teste*. Processo ou ação é um grupo formado por um ou mais comandos cuja execução deve ser repetida; teste é o elemento controlador desta repetição. O teste consiste na avaliação de uma expressão lógica. Dependendo do resultado desta avaliação, realiza-se ou encerra-se a execução do processo ou ação e o fluxo vai para a saída da estrutura.

As estruturas de repetição têm duas versões principais: DO-WHILE e REPEAT-UNTIL:*

DO-WHILE

Esta versão se caracteriza principalmente pela colocação do teste antes do processo ou ação (Fig. 2.5).

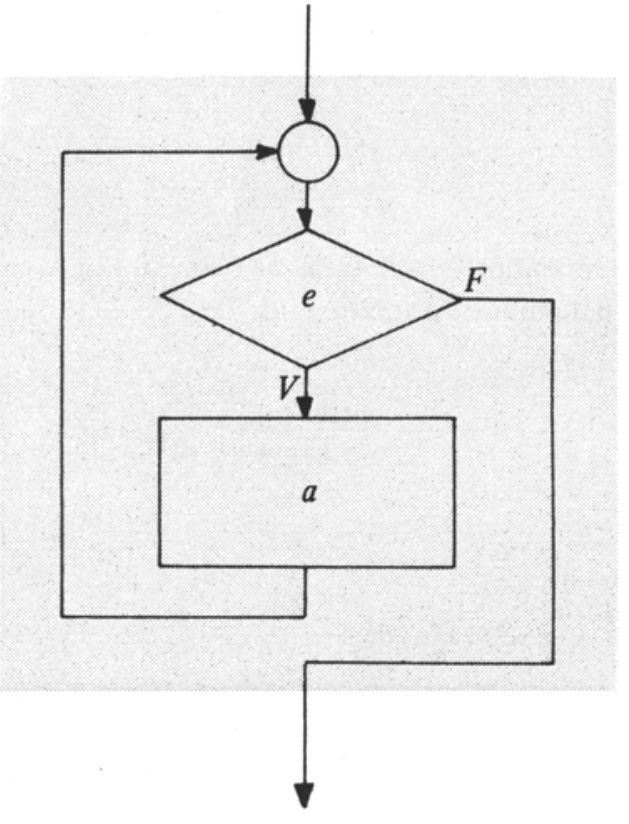


Fig. 2.5

* *While* = enquanto. *Until* = até que.

```
do while e
    a
od
```

onde *e* é uma expressão lógica e *a* é uma ação ou processo. Enquanto a expressão *e* é VERDADEIRA, o processo *a* é executado.

Exemplos. *a)*

```
i: = 1;
do while i < 101;
   a: = a + b;
   i: = i + 1;
od;
```

b)

```
j: = false;
do while j = true;
   print m;
   call rotina;
   j: = false;
od;
```

Enquanto no exemplo (*a*) temos a execução de um processo cem vezes, no exemplo (*b*), o processo não será executado. Esta é uma característica importante do DO-WHILE, devido ao fato de o teste estar disposto antes do processo.

REPEAT-UNTIL

Esta versão difere do DO-WHILE pela disposição do teste após o processo e pela saída da estrutura, que ocorre quando a condição é VERDADEIRA. O processo sempre será executado, pelo menos uma vez (Fig. 2.6).

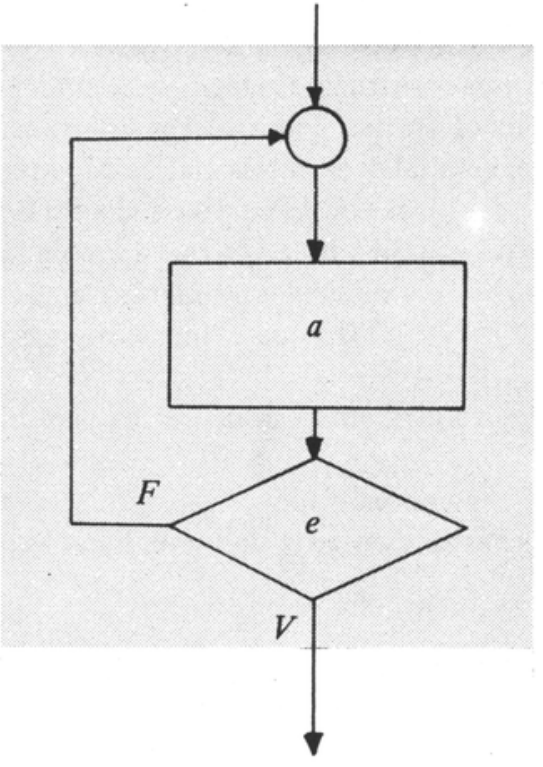


Fig. 2.6

```
repeat
  a
until e.
```

Exemplos. *a)*

```
j: = 10;
repeat
   j: = 0;
   p: = p + q;
until j = 0;
```

b)

```
i: = 0;
repeat
   i: = i + 2;
   j: = 5 * m;
   print j;
until i = 100;
```

No exemplo (*a*), a execução é feita apenas uma vez, pois a condição $j = 0$ é satisfeita logo na primeira passagem, uma vez que a instrução $j: = 0$; dentro do processo da estrutura torna verdadeira a expressão lógica da cláusula UNTIL. No exemplo (*b*), a execução do mesmo processo é feita 50 vezes.

2.2 – CARACTERÍSTICAS DAS ESTRUTURAS DE CONTROLE

Foi provado matematicamente que podemos escrever qualquer programa usando unicamente três estruturas de controle: SEQUÊNCIA, DO-WHILE e IF-THEN-ELSE [4]. Todavia, nem sempre o uso restrito destas três construções é fácil e eficiente. Novas estruturas têm sido acrescentadas ao rol das construções aceitas para uso em Programação Estruturada, com o intuito de implementar com maior naturalidade certas classes de problemas de programação. É o que ocorre, por exemplo, com a estrutura CASE, cuja ausência em linguagem de programação provoca o uso de ninhos de IF-THEN-ELSE's toda vez que a escolha pode ser feita entre mais do que duas alternativas. O excesso de IF's aninhados diminui a clareza do programa.

Um outro exemplo é o REPEAT-UNTIL, que é uma forma de repetição equivalente ao DO-WHILE.

Outras estruturas têm sido sugeridas, mas seu uso ainda não foi totalmente aceito [16] [27].

Em vista disto, consideramos como estruturas de controle *válidas* para uso em Programação Estruturada não apenas aquelas apresentadas aqui, mas todas as que possuam as seguintes características principais:

a) fluxo de controle com uma única entrada e uma única saída; e
b) comportamento padronizado do fluxo de controle dentro da estrutura, de tal forma que seu emprego possibilite a resolução de alguma classe de problemas de programação.

2.3 – COMPOSIÇÃO DE PROGRAMAS

Após o estudo isolado das estruturas de controle, feito na Seç. 2.1, convém agora ampliarmos nossa visão de Programação Estruturada, fazendo a integração desses esquemas em algoritmos mais complexos e comentar algumas vantagens que aparecem desde já.

Para a formação de programas, podemos integrar as estruturas de controle válidas de duas maneiras: *concatenação* e *aninhamento*.

Na *concatenação* de duas estruturas de controle A e B, nesta ordem, temos a formação de um fluxograma, onde a saída do fluxo de controle de A é a entrada no fluxo de controle de B (Fig. 2.7*a*).

No *aninhamento* de uma estrutura de controle A numa estrutura de controle B, obtemos um fluxograma em que a estrutura A é uma parte do fluxograma da estrutura B. Isto significa que o aninhamento só é possível se a estrutura B puder ser decomposta em funções mais detalhadas, isto é, refinada em termos da linguagem de programação usada (Fig. 2.7*b*).

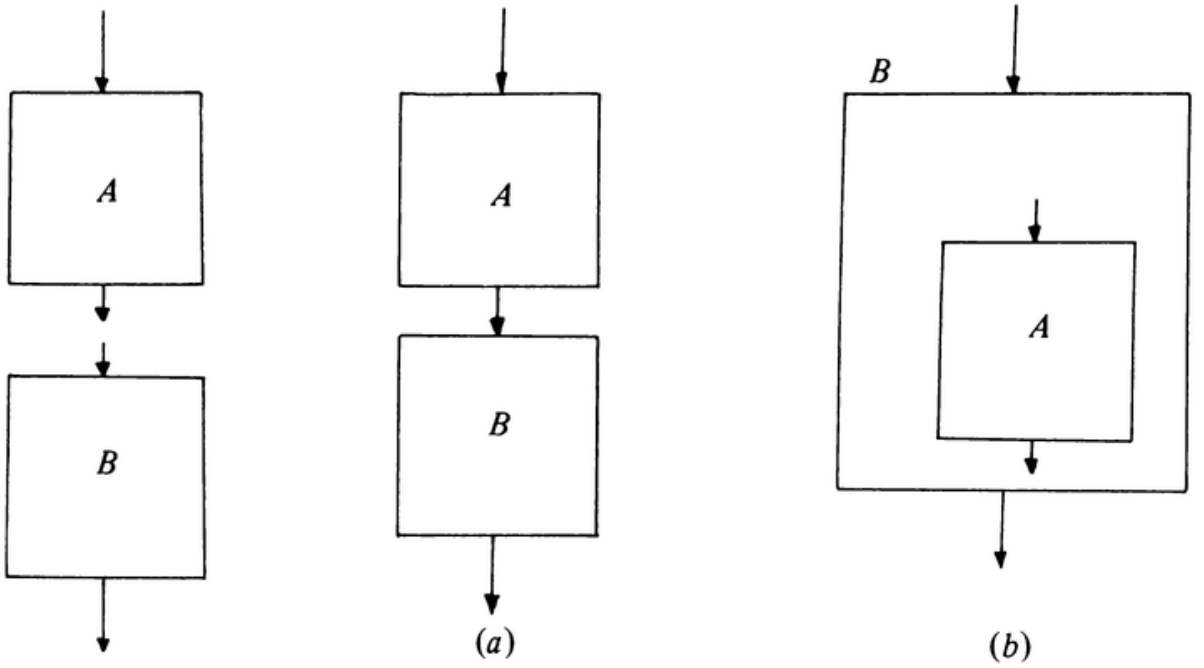
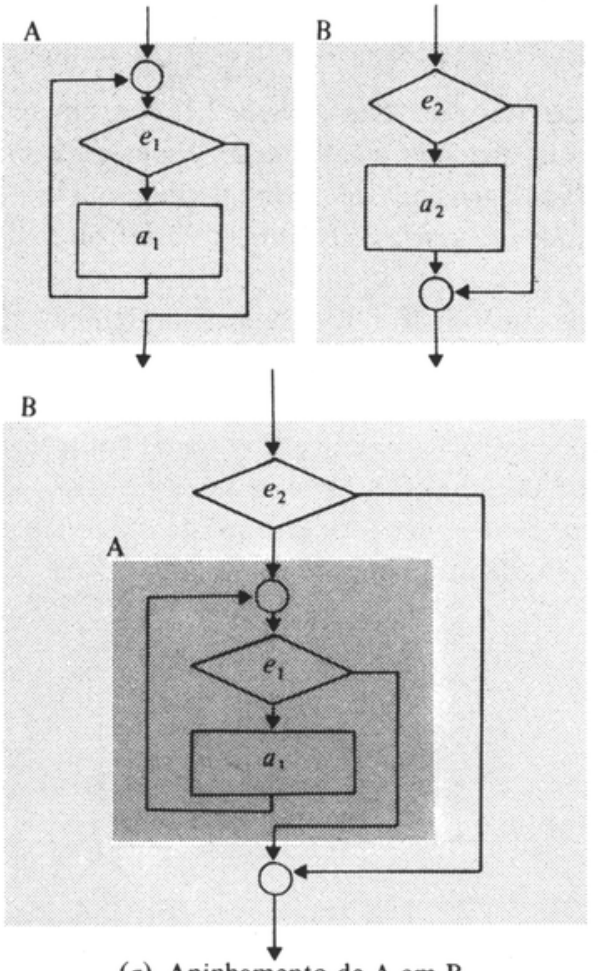


Fig. 2.7

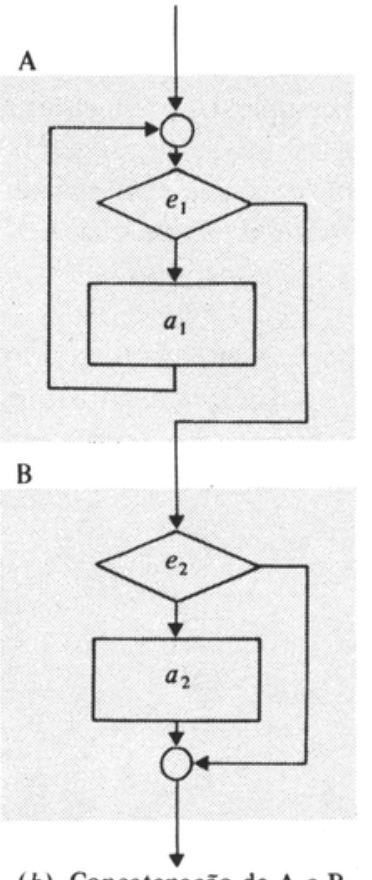
Exemplo. Sejam A uma estrutura de repetição DO-WHILE e B uma estrutura de seleção IF-THEN. A Fig. 2.8 apresenta um exemplo de concatenação de A e B e de aninhamento de A em B.

Os textos dos fluxogramas (*a*) e (*b*) da Fig. 2.8 são, respectivamente, os seguintes:

```
a) if e2
       then
          do while e1
             a1
          od
   fi
```



(*a*) Aninhamento de A em B



(*b*) Concatenação de A e B

Fig. 2.8

b) do while e_1
a_1
od
if e_2
then
a_2
fi

Usando estas duas formas de integração de estruturas de controle, podemos obter algoritmos mais complexos, compostos exclusivamente de seqüências, seleções e repetições.

2.3.1 – Um Algoritmo Estruturado*

Na Fig. 2.9, apresentamos um algoritmo abstrato em forma de fluxograma, o que, à primeira vista, nos parece altamente incompreensível. Todavia, ele é composto unicamente de concatenações e aninhamentos de estruturas de controle dos tipos mencionados anteriormente.

---

* Usaremos a seguinte notação ao longo desta seção:

- letras minúsculas representam instruções numa certa linguagem de programação;
- letras maiúsculas representam unidades funcionais refináveis em termos desta linguagem.

As figuras seguintes mostrarão estas estruturas com maior clareza.

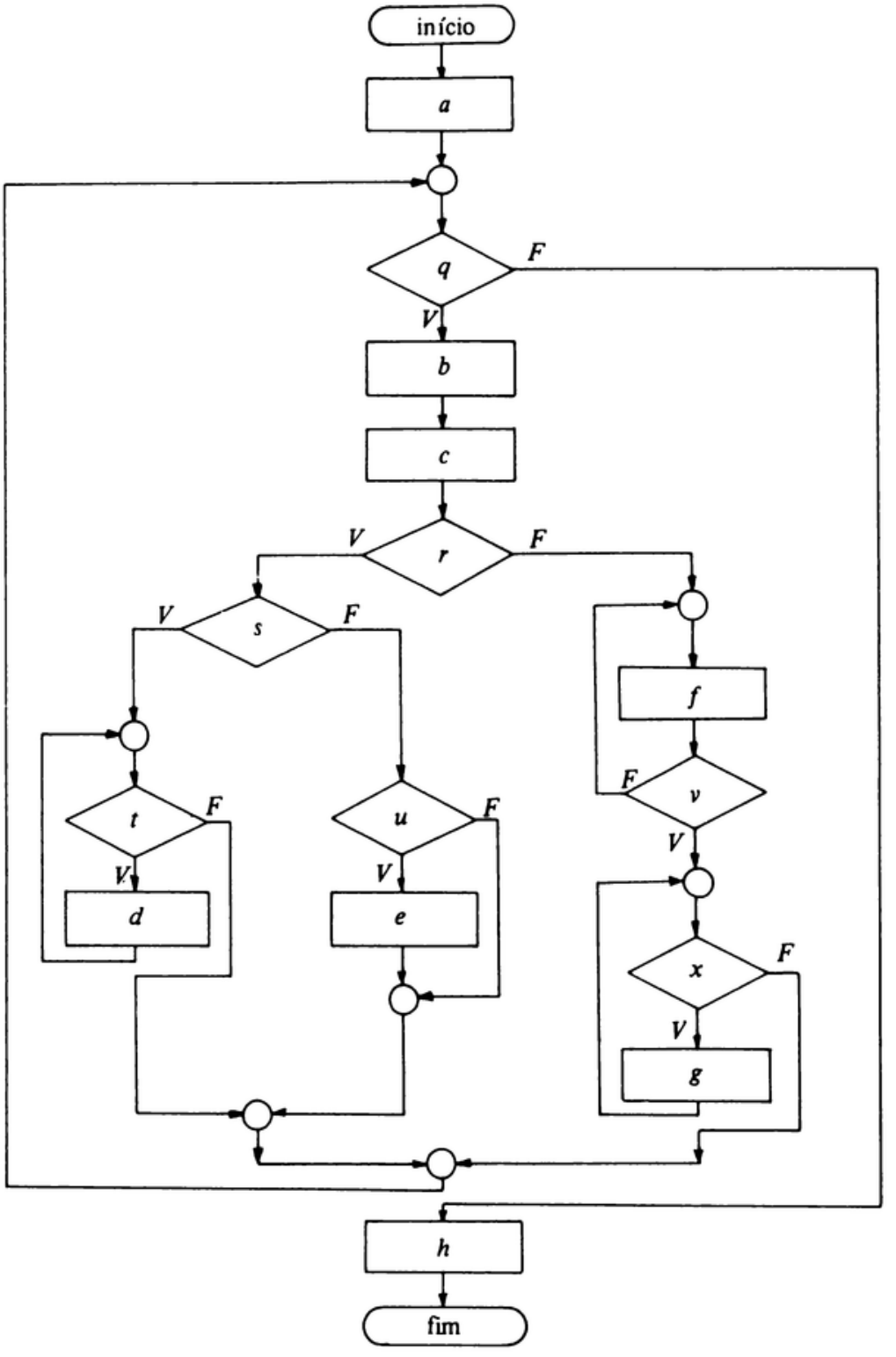


Fig. 2.9

Na Fig. 2.10, cinco estruturas são destacadas:

A: SEQÜÊNCIA
B: DO-WHILE

C: IF-THEN
D: REPEAT-UNTIL
E: DO-WHILE.

Destacamos no fluxograma as cinco estruturas de controle salientadas na Fig. 2.10.

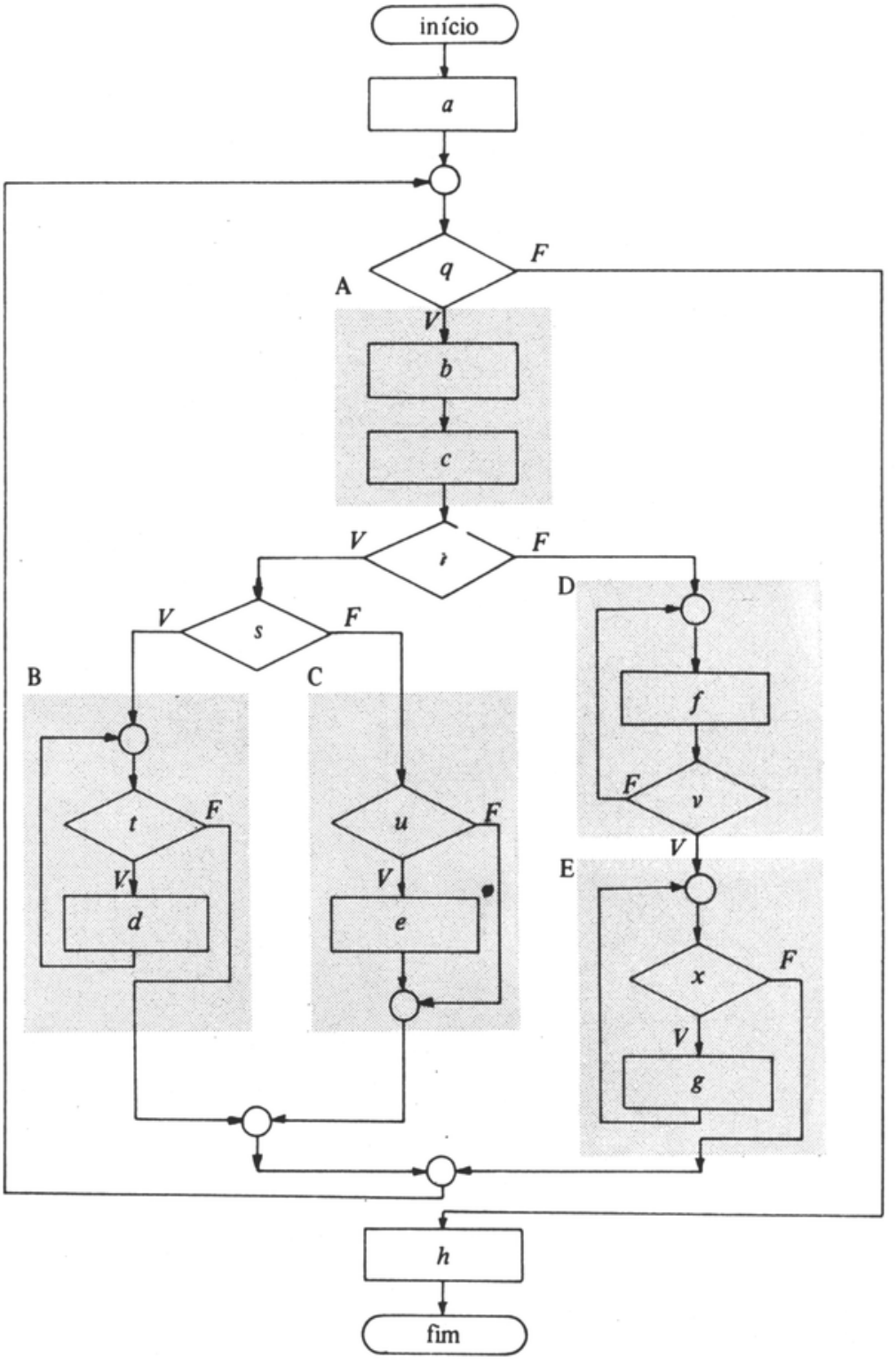


Fig. 2.10

Agora que estas estruturas já estão mais visíveis dentro do fluxograma e como conhecemos o fluxo padronizado de cada uma delas, podemos simplificar o algoritmo considerando-as simplesmente como unidades funcionais (caixas-pretas) com o seu nome dentro.

A Fig. 2.11 mostra um fluxograma equivalente ao anterior, porém já mais simplificado e distinguindo novas estruturas:

F: IF-THEN-ELSE
G: SEQÜÊNCIA.

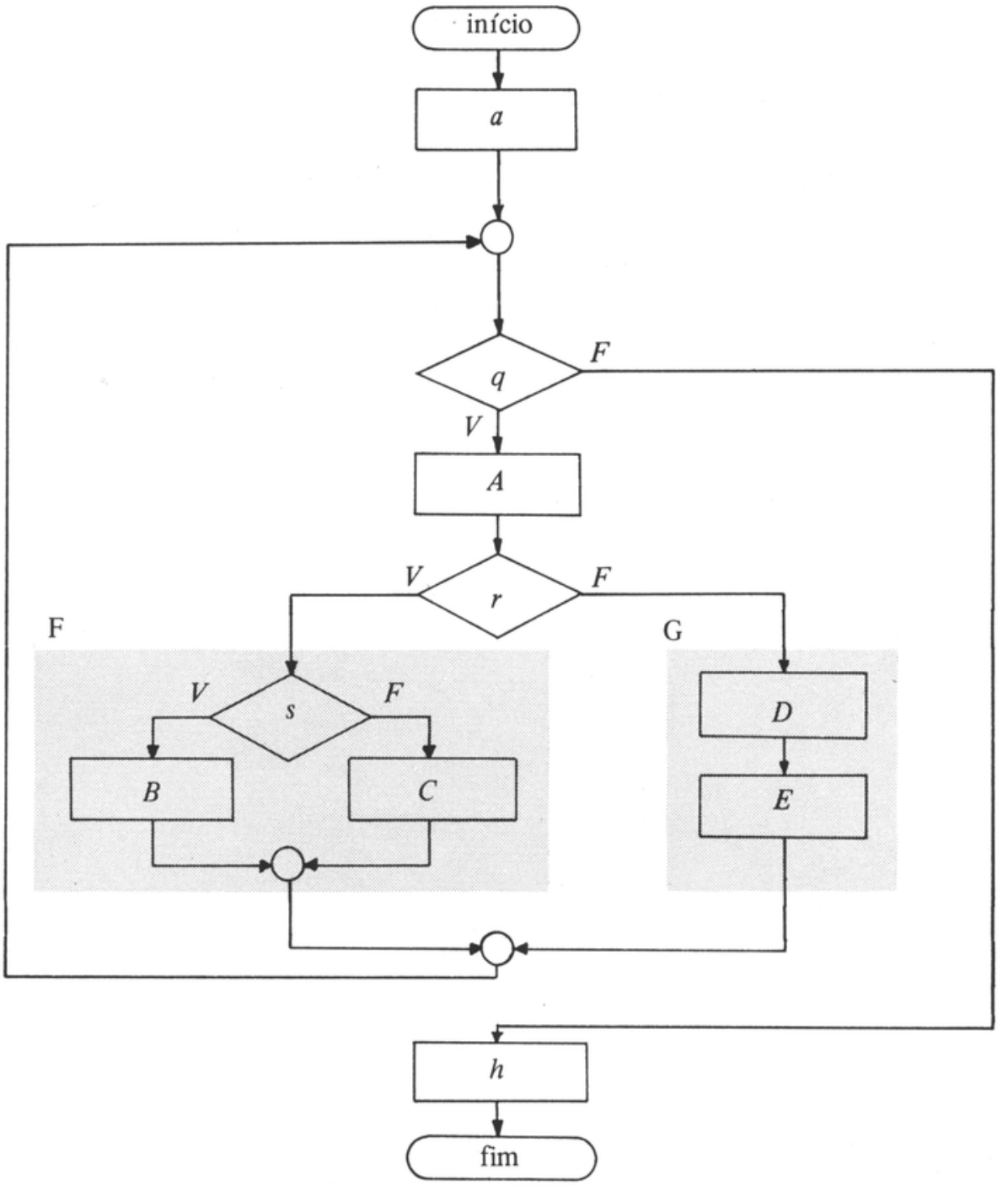


Fig. 2.11

Considerando agora *F* e *G* como *caixas-pretas*, podemos distinguir no fluxograma equivalente da Fig. 2.12 uma nova estrutura:

H: IF-THEN-ELSE.

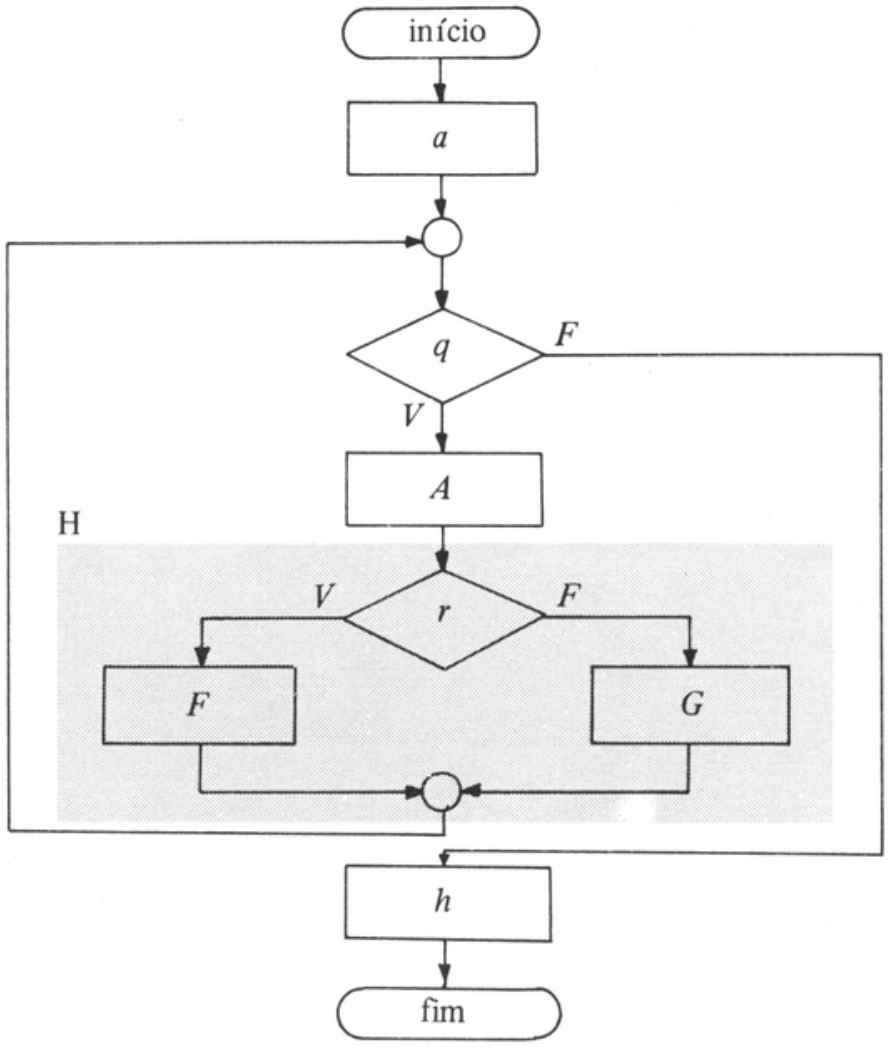


Fig. 2.12

Procedendo de forma análoga, obteremos, sucessivamente:

Fig. 2.13 – I: SEQÜÊNCIA
Fig. 2.14 – J: DO-WHILE
Fig. 2.15 – K: SEQÜÊNCIA.

Este exemplo mostra a grande flexibilidade das estruturas de controle padronizadas pela Programação Estruturada. Elas são usadas em todos os níveis de detalhamento do programa. A Fig. 2.15 mostra, por exemplo, que o programa como um todo é uma seqüência composta de

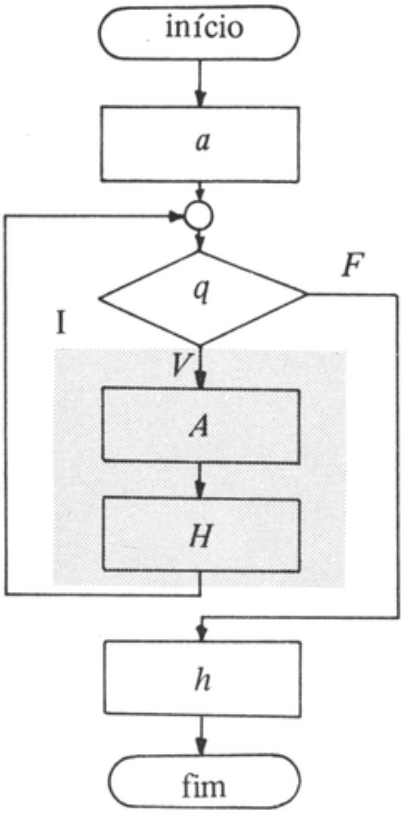


Fig. 2.13

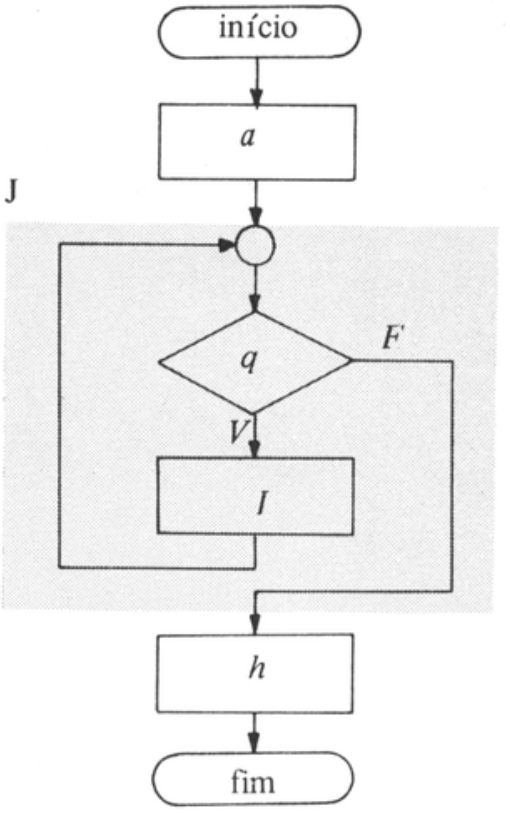


Fig. 2.14

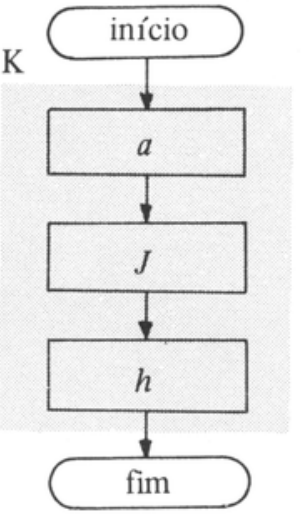


Fig. 2.15

uma instrução (*a*), um grande DO-WHILE (*J*) e outra instrução (*h*). Esta característica é de grande importância para o desenvolvimento de um programa, conforme veremos no Cap. 3.

2.3.2 – A Apresentação do Código-Fonte

A maneira como apresentamos o código-fonte é fundamental para a melhor compreensão do programa. O uso de margens verticais para determinar os níveis de detalhamento é a forma mais usada em linguagens como PL/I, ALGOL e PASCAL.

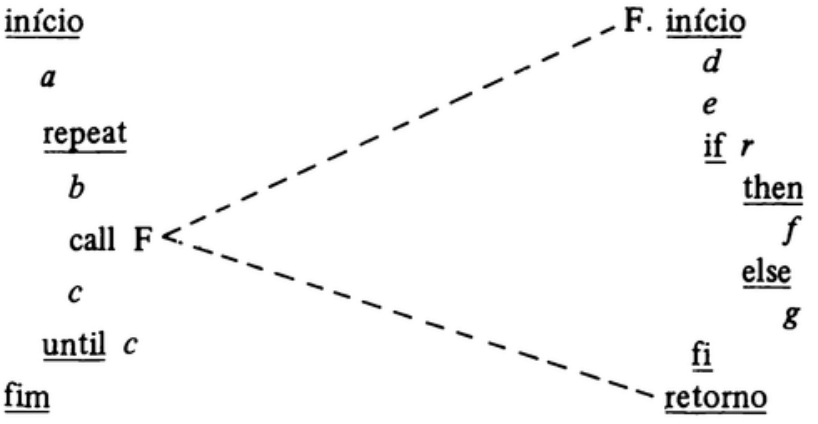
Como ilustração, apresentamos a seguir o código fonte do algoritmo estruturado da Seç. 2.3.1 e destacamos nele as cinco estruturas de controle salientadas na Fig. 2.10.

```
início
  a
  do while q
    b                                   A
    c
    if r
      then
        if s
          then
            do while t
              d                         B
            od
          else
            if u
              then
                e                       C
            fi
        fi
      else
        repeat
          f                             D
        until v
        do while x
          g                             E
        od
    fi
  od
  h
fim
```

2.3.3 – O Uso de Subprogramas

A utilização de subprogramas se adapta perfeitamente às idéias aqui expostas. Um subprograma deve ter uma única entrada e uma única saída e ser escrito apenas com as estruturas de controle válidas para a Programação Estruturada. Desta forma, sua chamada pelo programa principal é considerada como a inserção de seu texto no ponto de chamada.

Exemplo.


início
a
repeat
b
call F
c
until c
fim
F. início
d
e
if r
then
f
else
g
fi
retorno

Capítulo 3

Desenvolvimento de Programas por Refinamentos Sucessivos

Após o estudo da estruturação básica, estamos em condições de examinar o segundo componente da técnica: o desenvolvimento de programas por refinamentos sucessivos. Ele se constitui numa sistemática de resolução de problemas complexos que permite transformar a programação numa atividade altamente disciplinada.

3.1 – LINGUAGEM NATURAL X LINGUAGEM DE PROGRAMAÇÃO

O desenvolvimento de um programa é um processo delimitado por dois eventos bem definidos: o evento que determina o início do desenvolvimento é o instante em que o problema a ser resolvido está perfeitamente especificado (evento 1); o evento que encerra o desenvolvimento é o instante em que o problema já tem uma solução codificada em alguma forma executável pelo computador (evento *n*) (Fig. 3.1).

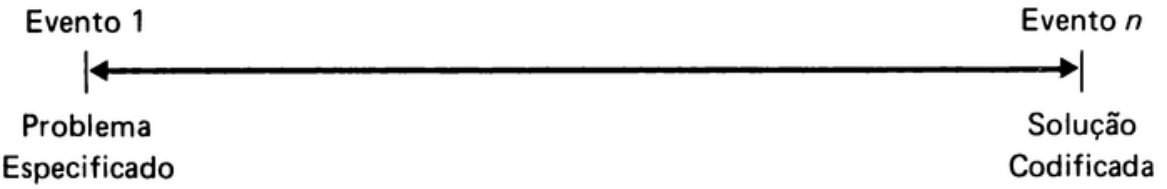


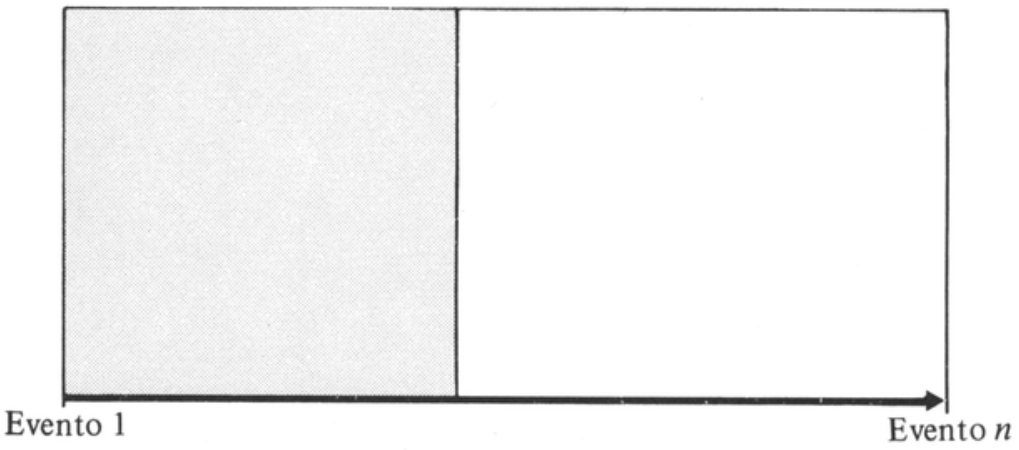
Fig. 3.1

Notamos que a grande diferença que existe entre estes dois pólos é a linguagem. No evento 1, temos o problema geralmente especificado em linguagem natural, ao passo que no evento *n*, a solução do problema deve estar totalmente implementada em linguagem executável pela máquina. Cabe ao programador fazer a passagem de uma linguagem para a outra. Esta passagem tem-se constituído numa grande dificuldade para muitos programadores e a explicação para este fato nos parece bastante clara. Sabemos que a nossa maneira de pensar é influenciada pela linguagem que usamos. O tipo de raciocínio utilizado no início do desenvolvimento recebe influência da nossa linguagem natural. Já no outro pólo, devemos estar raciocinando em termos de uma outra linguagem, a de programação, que não é nossa maneira usual de pensar.

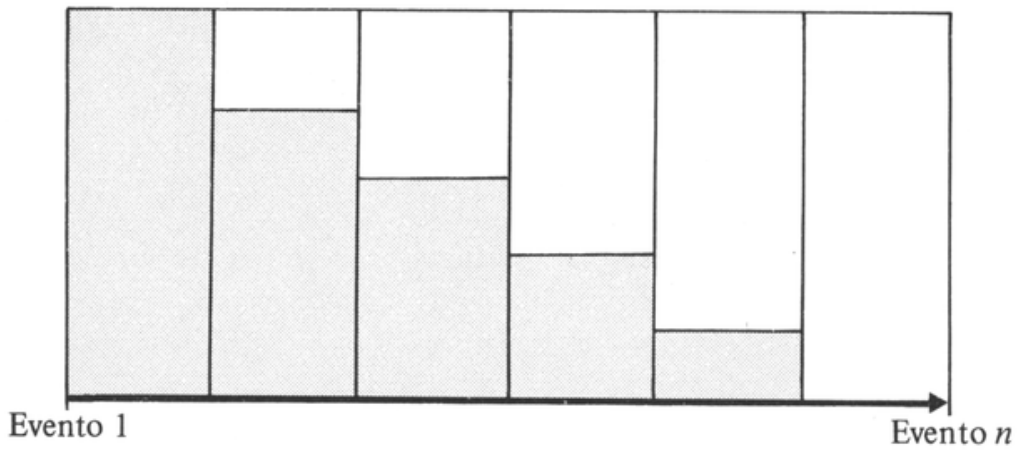
Esta mudança de tipo de raciocínio durante o desenvolvimento de um programa constitui-se na grande barreira da arte da programação.

A forma de passagem que tem sido tentada, em geral, é a mudança *brusca* de um tipo de raciocínio para o outro, que é feita em algum ponto da fase de desenvolvimento. Esta forma de passagem apresenta a deficiência de exigir grande concentração do esforço mental no momento da mudança de uma linguagem para outra. A Fig. 3.2*a* é uma maneira de representar este tipo de transição.

A Programação Estruturada sugere uma nova forma de mudança no comportamento mental durante o desenvolvimento: a passagem *suave* de um tipo de raciocínio para outro, através de *uso paralelo de uma linguagem de programação com a linguagem natural* (Fig. 3.2*b*).



(*a*) desenvolvimento tradicional



(*b*) desenvolvimento por refinamentos sucessivos

Raciocínio em linguagem natural
Raciocínio em linguagem de programação

Fig. 3.2

3.2 – A HIERARQUIA NA ESTRUTURA DE UM PROGRAMA

É conveniente a introdução, neste ponto, da idéia de hierarquia na estrutura de um programa, por ser de bastante utilidade na fase de desenvolvimento por refinamentos sucessivos.

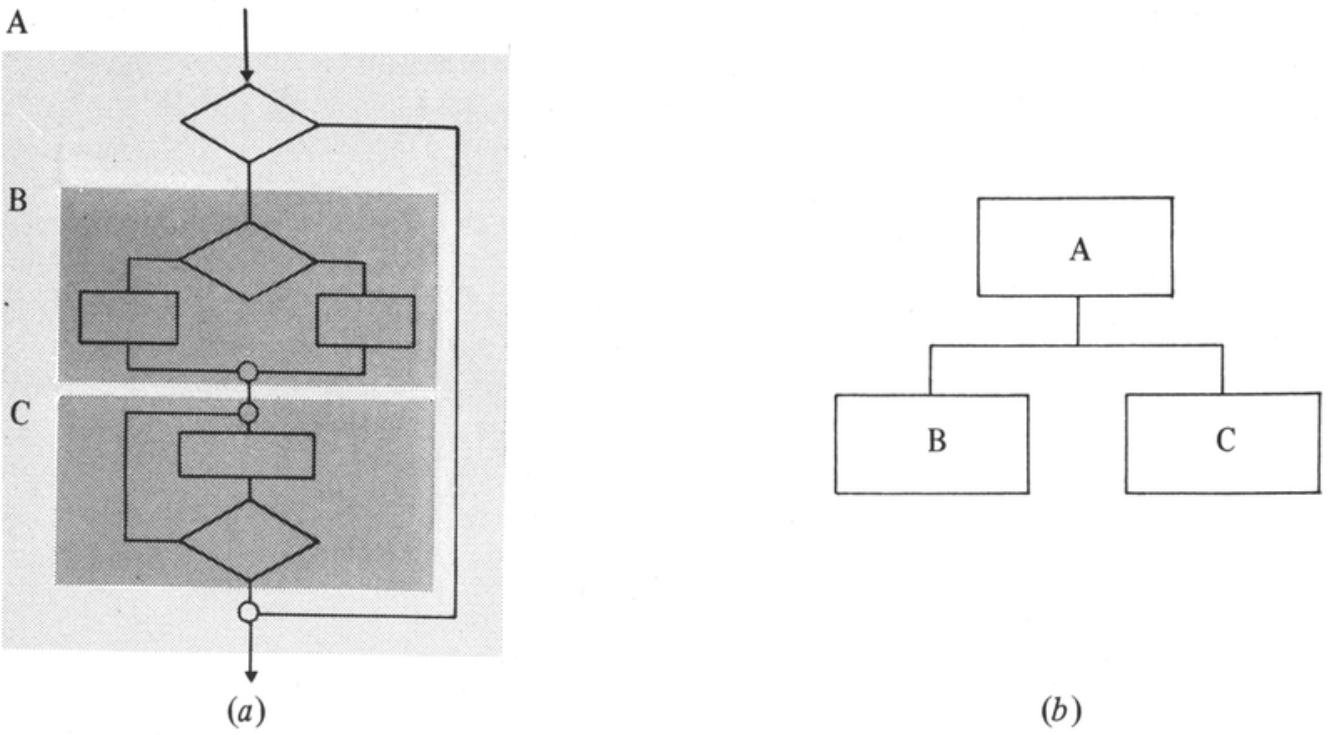


Fig. 3.3

A hierarquia é estabelecida em termos da linguagem de programação em uso e o critério a ser utilizado é o de níveis de detalhamento dos processos. Por exemplo, na Fig. 3.2, a *árvore* do item (*b*) representa a hierarquia do fluxograma do item (*a*).

No desenvolvimento de programas estruturados, o algoritmo vai sendo construído a partir de seus níveis mais altos para os mais detalhados.

3.3 – OS PASSOS DO DESENVOLVIMENTO

No desenvolvimento de programas por refinamentos sucessivos, usaremos uma importante ferramenta mental de que dispomos: a *abstração*. Através dela somos capazes de dar nome a uma determinada operação e levar em conta apenas "o que ela faz" sem considerar "como trabalha".

Para desenvolver um programa, devemos escrever várias versões do mesmo. A primeira delas deve ser uma *instrução abstrata* capaz de resumir, em linguagem natural, a essência da especificação do programa. A seguir, inserindo gradativamente instruções da linguagem de programação, escrevemos novas versões, passando por diversos níveis de abstração. Para cada um destes níveis, consideramos a existência de uma *máquina abstrata* capaz de executar nossas abstrações expressas pelo uso paralelo das linguagens natural e de programação. O processo chega ao fim quando obtemos uma versão totalmente executável pela máquina real.

O procedimento pode ser resumido na seguinte regra:

"Escrever versões equivalentes e completas do programa onde:

a) a primeira versão é apenas uma especificação geral e sucinta da função do programa;
b) a última versão é o programa inteiramente codificado na linguagem de programação;

c) para passar da primeira versão para a última devem ser escritas tantas versões intermediárias quantas forem necessárias;

d) uma versão intermediária *i* é igual à sua versão antecessora *i*-1, com o refinamento de alguma função da versão *i*-1, isto é, maior detalhamento de alguma função de *i*-1;

e) tanto a primeira versão como as intermediárias podem fazer uso paralelo de comandos da linguagem natural e da linguagem de programação; e

f) só podemos escrever qualquer versão usando exclusivamente as estruturas de controle da Programação Estruturada."

Por exemplo,* se um problema a ser resolvido estiver perfeitamente especificado, poderemos escrever a primeira versão do programa, o que não nos exigirá grande esforço mental, pois basta dizer resumidamente em linguagem natural qual a sua função.

```
PASSO 1. start;
            "resolver problema x"
         stop;
```

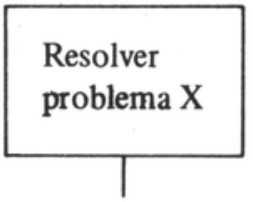


Fig. 3.4

A representação gráfica em forma de árvore é muito útil para a visualização do desenvolvimento do programa, bem como para a sua documentação final. A construção da árvore, paralelamente à escrita das versões, pode auxiliar bastante na atividade desta fase. A versão 1 do nosso exemplo é o topo da árvore.

No passo 2, teremos um refinamento da versão 1. Devemos fazer este refinamento obedecendo às restrições no uso de estruturas de controle válidas. Em nosso exemplo, apresentamos a decomposição de "resolver problema x" em uma seqüência de três funções:

---

* A notação usada nos exemplos desta seção é a seguinte:

– textos entre aspas significam funções expressas em linguagem natural (abstrações);

– textos que não estão entre aspas significam instruções da linguagem de programação fictícia, que é descritano Apêndice A;

– textos entre chaves são comentários.

```
PASSO 2.  start;
             "ler dados. (1)"
             "processar. (2)"
             "editar relatório. (3)"
          stop;
```

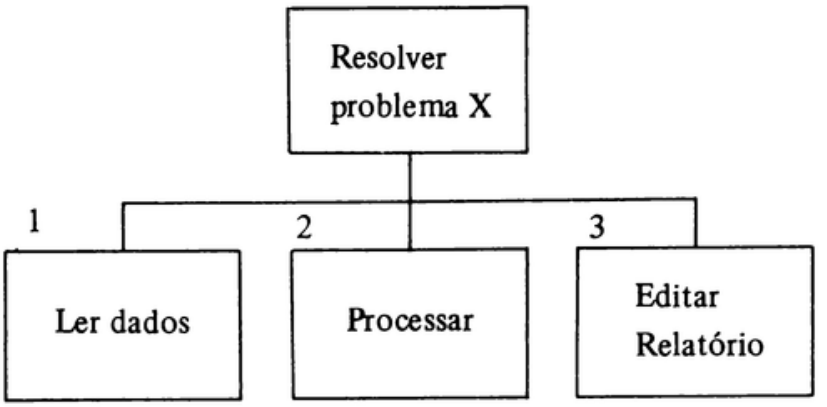


Fig. 3.5

Em algum passo posterior, deveremos refinar a função "processar". Supondo que ela seja uma repetição, poderemos ter o seguinte refinamento:

```
PASSO n.  {2 – PROCESSAR}

          chave: = 1;
          do while  chave = 1;
             "efetuar alterações. (2.1)"
             "calcular vencimentos. (2.2)"
             "processar ordem. (2.3)"
             "emitir aviso. (2.4)"
          od;
```

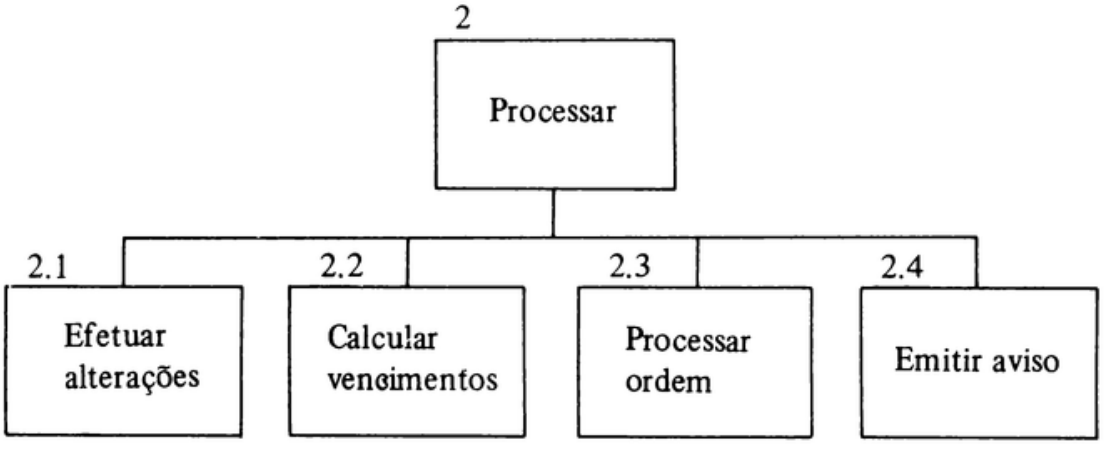


Fig. 3.6

É interessante notar que a escrita completa do programa em cada passo do desenvolvimento pode tornar-se uma tarefa monótona e demorada, principalmente quando o processo está avançado. Neste caso, podemos escrever apenas o refinamento correspondente sem repetir a parte do texto que fica inalterada.

3.4 – UM EXEMPLO COMPLETO

O objetivo do exemplo a seguir é mostrar as características do desenvolvimento de um programa completo real.

Nossa preocupação ao escolhermos um problema bastante simples foi a de evitar que a complexidade viesse a desviar a atenção do leitor para detalhes de lógica ou que a grande extensão do desenvolvimento tornasse enfadonha a nossa apresentação.

Problema. "Ordenar uma tabela unidimensional de *n* posições em ordem crescente de valores. O tamanho da tabela e os seus elementos são lidos em cartões. Imprimir a tabela original e a tabela ordenada."

Desenvolvimento. A primeira versão de nosso programa é o enunciado do problema. Usaremos a instrução (abstrata) "ordenar" como abreviatura dessa especificação.

Vamos construir a árvore do programa paralelamente ao desenvolvimento.

PASSO 1.
```
start;
   "ordenar"
stop;
```

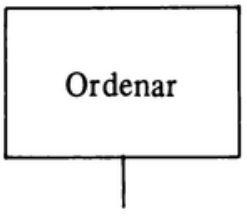


Fig. 3.7

Já temos aqui uma primeira aproximação do nosso programa, onde aparece o uso paralelo de linguagens. As palavras *start* e *stop* são instruções de nossa linguagem de programação que indicam que o programa terá uma única entrada em cima e uma única saída embaixo.

Vamos fazer um primeiro refinamento, subdividindo a função "ordenar" em subfunções usando exclusivamente estruturas de controle válidas. Raciocinando em termos da especificação do problema, sentimos que existe uma seqüência bem definida de processos: primeiramente, o programa deve obter a tabela a ser ordenada e imprimi-la; a seguir, esta tabela deve ser ordenada em ordem crescente dos valores de seus elementos; finalmente, a tabela ordenada deve ser impressa.

Temos, então, a segunda versão do programa "ordenar".

PASSO 2. {ORDENAR}

start;
"obter tabela original e imprimir. (1)"
"ordenar tabela. (2)"
"imprimir tabela ordenada. (3)"
stop;

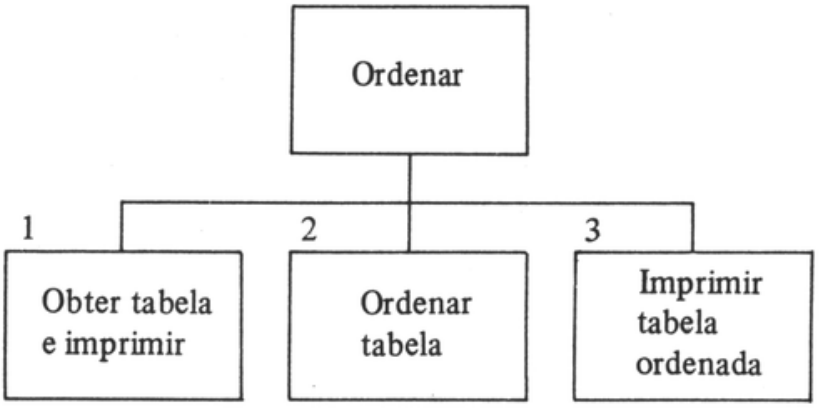


Fig. 3.8

Antes de passarmos para o passo 3, no qual vamos refinar uma das três funções do passo 2, convém tentarmos ter certeza da correção do programa até esse ponto. A dificuldade na análise da correção de um texto de programa é proporcional à sua extensão. O fato de cada refinamento caber em poucas linhas de código, seja ele abstrato ou concreto, é fundamental para a análise da correção do programa paralelamente ao seu desenvolvimento. Por "análise de correção", neste contexto, queremos dizer que o programador, por simples inspeção, adquire confiança de que o programa executará corretamente na máquina abstrata definida neste nível de abstração. Assim, podemos estabelecer facilmente a correção do passo 2.

O passo 3 será um refinamento da função "obter tabela original e imprimir". Teremos uma nova seqüência.

PASSO 3. {1 – OBTER TABELA E IMPRIMIR}

read n;
"ler e imprimir o i-ésimo elemento da tabela
para $i = 1, 2, \ldots, n$. (1.1)"

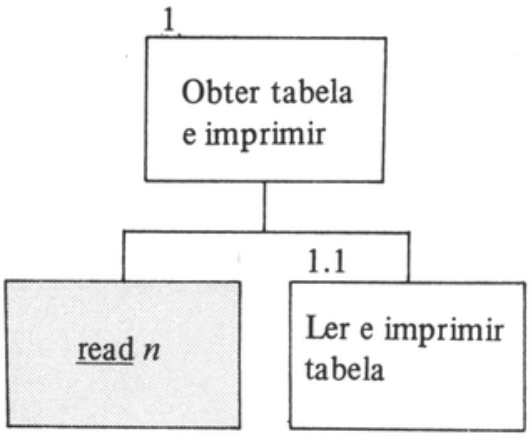


Fig. 3.9

A função "ler e imprimir . . ." especifica uma repetição controlada por uma variável auxiliar i. A escolha deve ser feita entre o REPEAT-UNTIL e o DO-WHILE. Como nada foi dito a respeito do valor de n, é possível que ele seja zero. Se usarmos o REPEAT-UNTIL e n for zero, ocorrerá um erro, pois esta estrutura provoca a execução de seu corpo pelo menos uma vez. Escolhemos, então, o DO-WHILE.

PASSO 4. {1.1 – LER E IMPRIMIR A TABELA}

```
i:= 1;
do while i ⩽ n;
    read tab (i);
    print tab (i);
    i:= i + 1;
od;
```

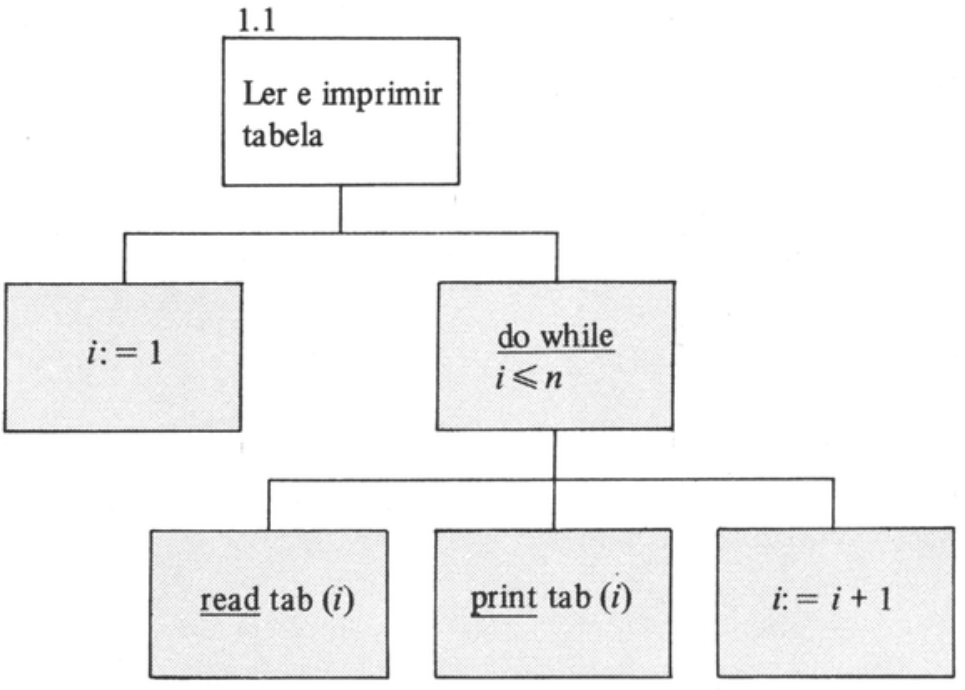


Fig. 3.10

Passemos agora ao refinamento da função "ordenar tabela". Existem muitos métodos disponíveis para programar a ordenação de uma tabela. Nossa escolha deveria basear-se numa série de critérios como o número médio de comparações do método, o gasto de memória extra, vantagens que o método pode tirar de uma ordenação prévia dos elementos etc. Não é objetivo deste texto analisar cada um desses métodos, por isso, vamos escolher a "seleção linear" pela simplicidade de seu algoritmo. Este método consiste em trocar o i-ésimo elemento com o menor elemento da tabela formada pelas posições de i a n, para $i = 1, 2, 3, \ldots, n - 1$. Temos, portanto, um novo caso de repetição.

Pelo mesmo motivo que no passo 4 usamos o DO-WHILE, repetimos aqui sua escolha.

PASSO 5. {2 – ORDENAR TABELA}

```
i: = 1;
do while i ⩽ n – 1;
   "procurar o menor elemento de tab (i) até
   tab (n) e trocar com tab (i). (2.1)"
   i: = i + 1;
od;
```

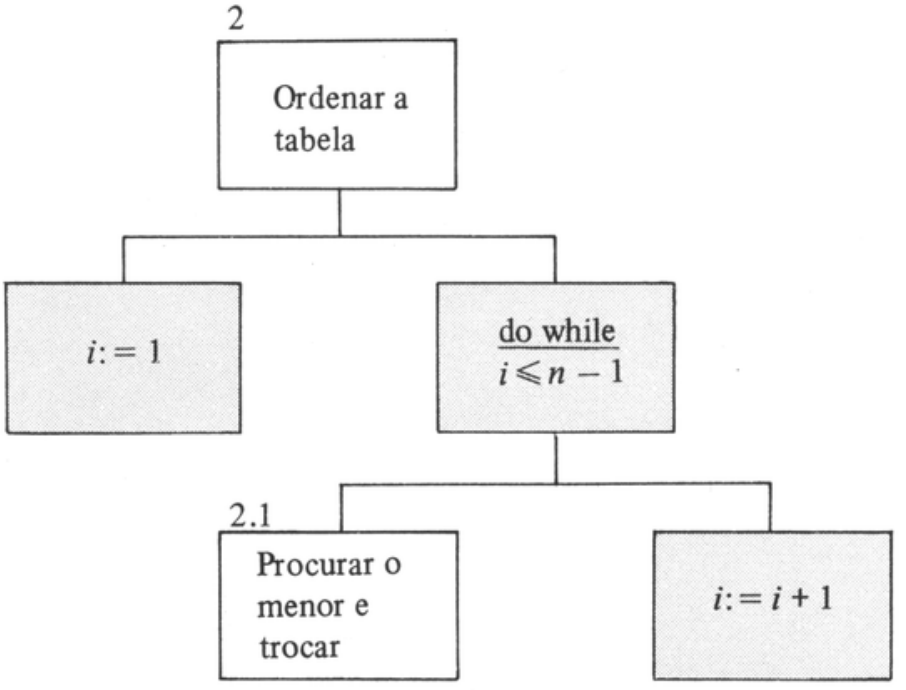


Fig. 3.11

A função 2.1 ("procurar o menor . . .") também é uma repetição, que deve ser controlada por uma nova variável j que varia de $i + 1$ até n. Desta vez, podemos optar pela estrutura REPEAT-UNTIL.

PASSO 6. {2.1 – PROCURAR O MENOR E TROCAR}

```
j: = i + 1;
repeat
   if tab (j) < tab (i)
      then
         "trocar tab (j) com tab (i). (2.1.1)"
   fi;
   j: = j + 1;
until j > n;
```

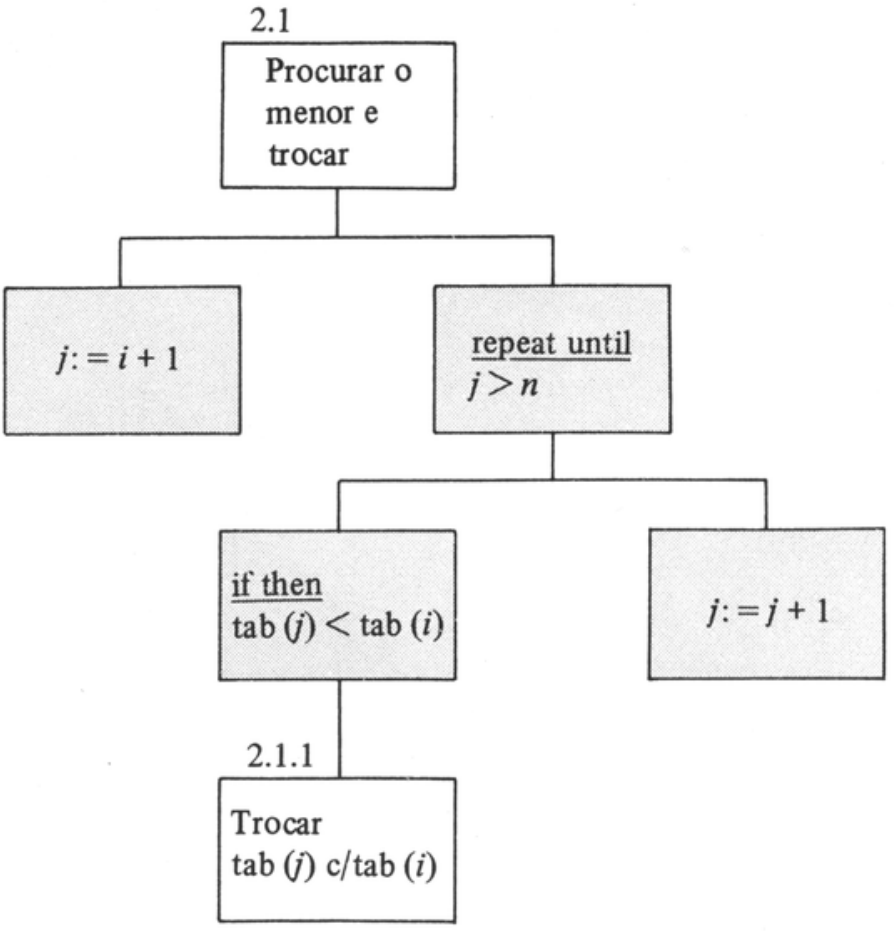


Fig. 3.12

O passo 7 é o refinamento de 2.1.1 ("trocar . . ."). É uma seqüência de comandos executáveis da linguagem de programação.

PASSO 7. {2.1.1 – TROCAR TAB (*I*) COM TAB (*J*)}

```
temp: = tab (i);
tab (i): = tab (j);
tab (j): = temp;
```

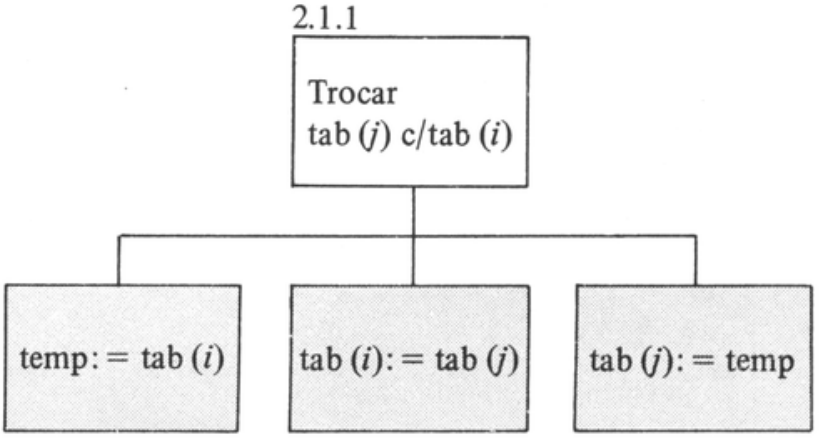


Fig. 3.13

Se juntarmos os pedaços da árvore em uma única figura (3.14), teremos uma idéia melhor da situação do desenvolvimento.

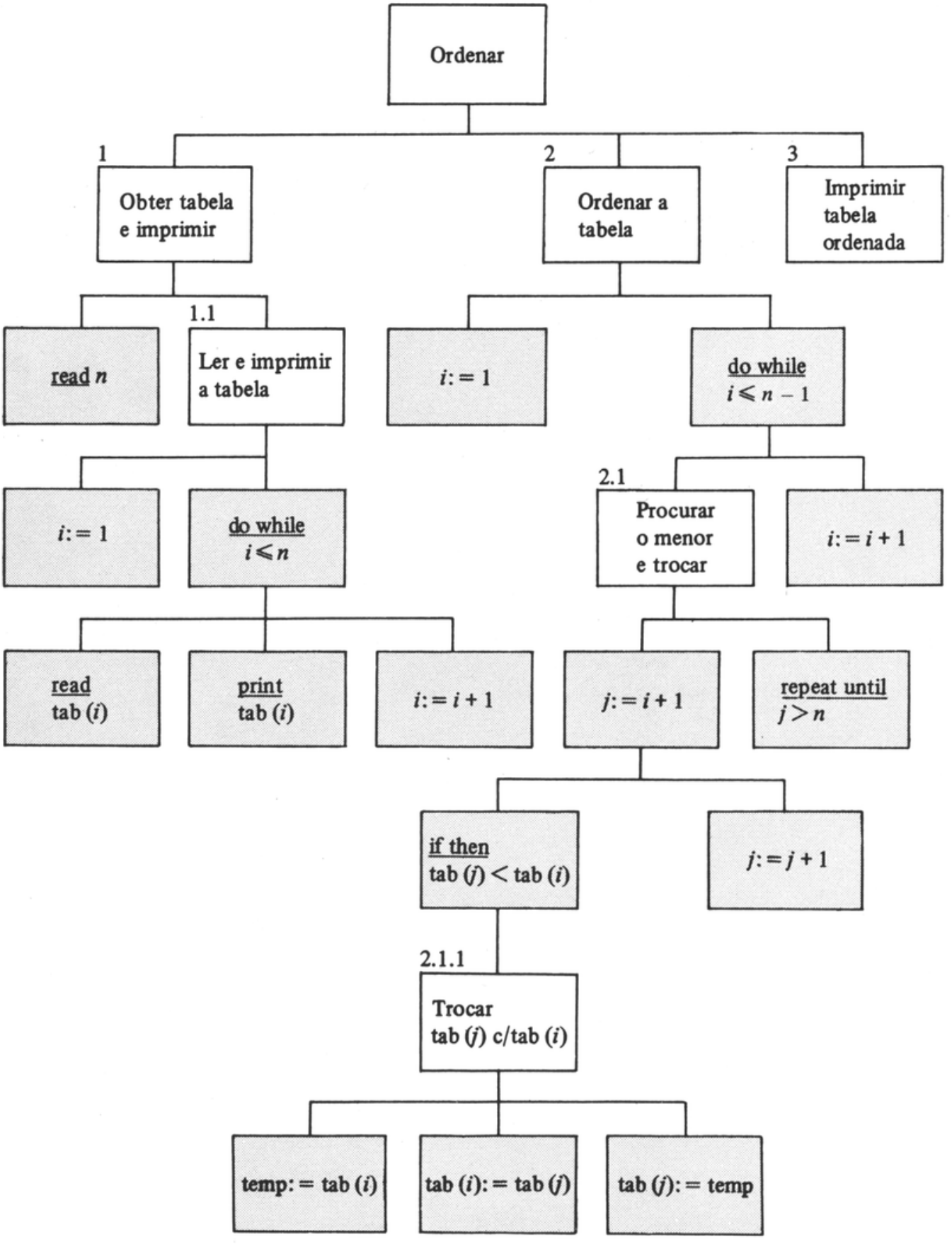


Fig. 3.14

Nota-se facilmente pela observação da árvore que só falta o refinamento de 3 ("imprimir abela ordenada").

PASSO 8. {3 – IMPRIMIR TABELA ORDENADA}

```
i:= 1;
do while i ⩽ n;
   print tab (i);
   i:= i + 1;
od;
```

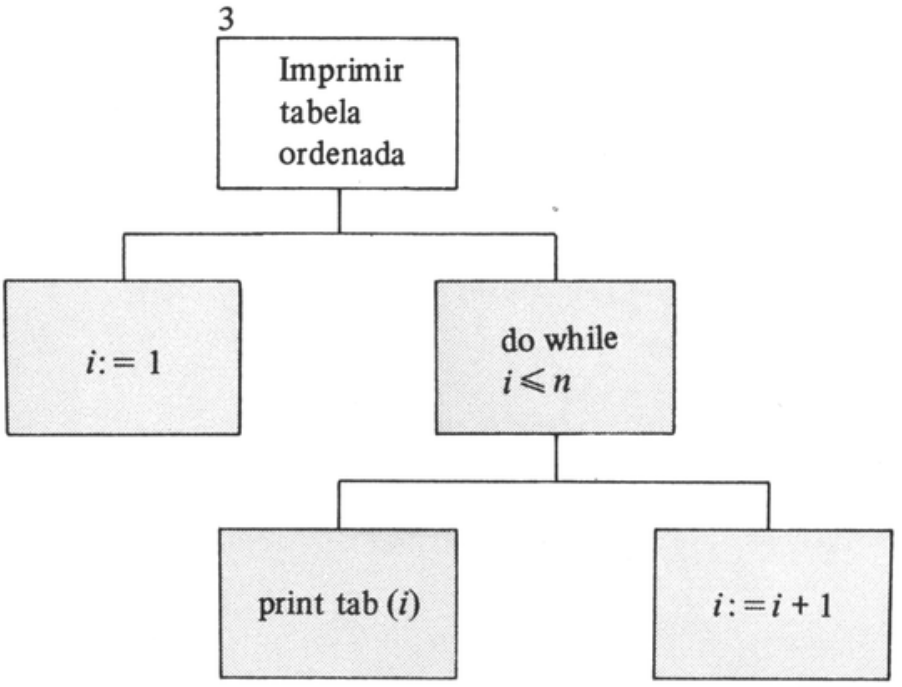


Fig. 3.15

Ainda com a ajuda da árvore, ou simplesmente usando a seqüência numérica dos refinamentos, podemos escrever a versão final do nosso programa.

PASSO 9. {ORDENAR}

```
start;
   {1 – OBTER TABELA E IMPRIMIR}
   read n;
   {1.1 – LER E IMPRIMIR A TABELA}
   i:= 1;
   do while i ≤ n;
      read tab (i);
      print tab (i);
      i:= i + 1;
   od;
   {2 – ORDENAR TABELA}
   i:= 1;
   do while i ≤ n – 1;
      j:= i + 1;
      {2.1 – PROCURAR O MENOR E TROCAR}
      repeat
         if tab (j) < tab (i)
            then
               {2.1.1 – TROCAR TAB (I) COM TAB (J)}
               temp:= tab (i);
               tab (i):= tab (j);
               tab (j):= temp;
         fi;
         j:= j + 1;
      until j > n;
      i:= i + 1;
   od;
   {3 – IMPRIMIR TABELA ORDENADA}
   i:= 1;
   do while i ≤ n;
      print tab (i); i:= i + 1;
   od;
stop;
```

3.5 – A DOCUMENTAÇÃO DO DESENVOLVIMENTO

A documentação criada pelo programador na fase de desenvolvimento deve ter dois objetivos:

a) funcionar como recurso visual que auxilie na manutenção da disciplina do desenvolvimento; e
b) servir como componente da documentação final do programa.

Este assunto será abordado aqui apenas sob alguns aspectos mais gerais.
As principais considerações dizem respeito a:

- forma do texto-fonte;
- uso de árvores para representar a hierarquia; e
- ausência de fluxogramas.

3.5.1 – A Forma do Texto-Fonte

A linguagem natural, se bem usada, é bastante clara. Por outro lado, a experiência tem mostrado que, com o uso da Programação Estruturada, instruções de linguagens de programação como DO-WHILE, IF-THEN-ELSE, CASE, REPEAT-UNTIL e outras têm-se tornado termos comuns e de significado perfeitamente conhecido dentro de ambientes de programação. Como resultado, o uso concomitante dessas linguagens nas versões de um programa tornam-no bastante claro. Somando a isto o uso de certas regras de formatação do texto e a inserção criteriosa de comentários, o código fonte torna-se um documento bastante claro do programa.

Exemplo.

```
{2 – ORDENAR A TABELA}
i: = 1;
do while i ⩽ n – 1;
    "procurar o menor elemento de tab (i) até
    tab (n) e trocar com tab (i). (2.1)"
    i: = i + 1;
od;
```

3.5.2 – O Uso de Árvores para Representar a Hierarquia

Sem dúvida, a melhor representação gráfica de hierarquia é feita através da *árvore*. O desenho da árvore do programa não só é uma ferramenta na fase de desenvolvimento (como vimos na Seç. 3.4) como é um elemento importante na documentação final por dar uma visão funcional do programa em seus vários níveis de detalhamento. A leitura do texto-fonte orientada pela visualização de sua hierarquia facilita a compreensão.

A construção da árvore pode tornar-se difícil se descermos até o nível de instrução da linguagem de programação. É recomendável especificar apenas as funções em linguagem natural. A árvore que desenhamos no exemplo da Seç. 3.4 poderia ser simplificada (Fig. 3.16).

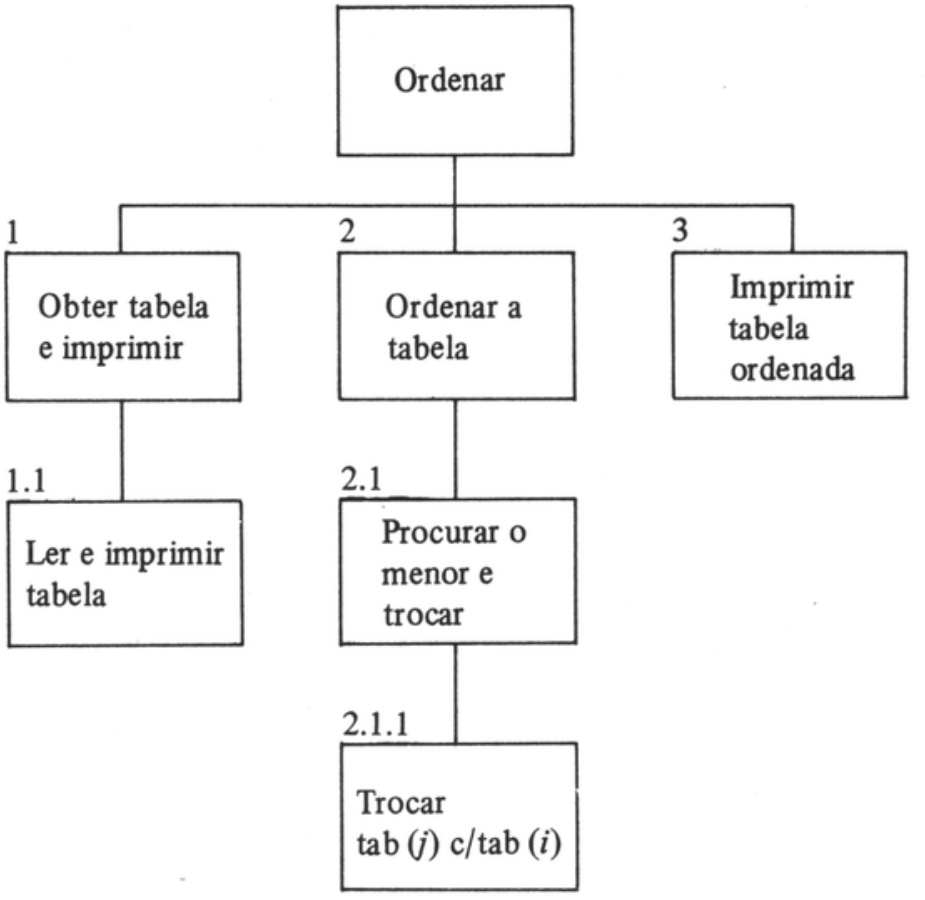


Fig. 3.16

3.5.3 – A Ausência de Fluxogramas

A principal função do fluxograma é proporcionar o acompanhamento visual do fluxo de controle do programa. Com o uso da Programação Estruturada, o fluxograma torna-se dispensável, pois o fluxo de controle tem padrões definidos.

O uso deste recurso também passa a ser desaconselhável na documentação final, pois o fluxograma de um programa estruturado é tão complicado quanto o de um não-estruturado. Uma outra desvantagem é que o nível de detalhamento de um fluxograma é mais baixo que o das estruturas de controle válidas.

Capítulo 4

Os Benefícios da Técnica

Durante a descrição da Programação Estruturada, feita nas seções anteriores, o leitor certamente já pôde sentir alguns efeitos benéficos que o seu emprego proporciona. Visando a consolidar essas idéias e destacar alguns benefícios menos tangíveis, vamos estudar as vantagens da nova técnica, segundo dois aspectos importantes da programação:

- correção; e
- qualidades externas dos programas.

4.1 – CORREÇÃO

Um programa só terá utilidade se estiver correto, isto é, se ele fornecer os resultados esperados de sua execução.

Os programador dispõe de alguns métodos (mais ou menos formais) que o auxiliam na obtenção da certeza de que seu programa está correto. De um modo geral, esses métodos dividem-se em três grupos: prova de correção, teste de programas e obtenção intuitiva da correção. Analisaremos aqui apenas as vantagens da Programação Estruturada frente aos dois últimos tipos, visto que os métodos que envolvem a prova de correção de programas não são práticos a ponto de se tornarem acessíveis aos programadores em geral.*

4.1.1 – Teste de Programas

As principais vantagens que o uso da Programação Estruturada oferece no teste e na depuração de programas podem ser resumidas em três títulos:

---

* Não podemos deixar de destacar a importância que a pesquisa na área de prova de correção de programas tem desempenhado no desenvolvimento das linguagens de programação. Podemos dizer, inclusive, que a Programação Estruturada emergiu naturalmente da necessidade de formalizar a semântica de linguagens de programação, com vistas a facilitar a prova de correção. O leitor interessado em ter uma visão inicial desta área deve ler o artigo de London [17]. As influências da Programação Estruturada na prova de correção de programas podem ser sentidas em muitos trabalhos como os de Hoare [14], de Wirth [24] e de Cardoso [6].

– o uso do computador mais cedo;
– maior número de testes para os módulos mais críticos; e
– maior facilidade no teste de programas desenvolvidos em equipe.

O desenvolvimento tradicional de programas tem sido feito, geralmente, de forma que o uso do computador para os testes é retardado para uma fase chamada *teste e depuração* (Fig. 4.1*a*).

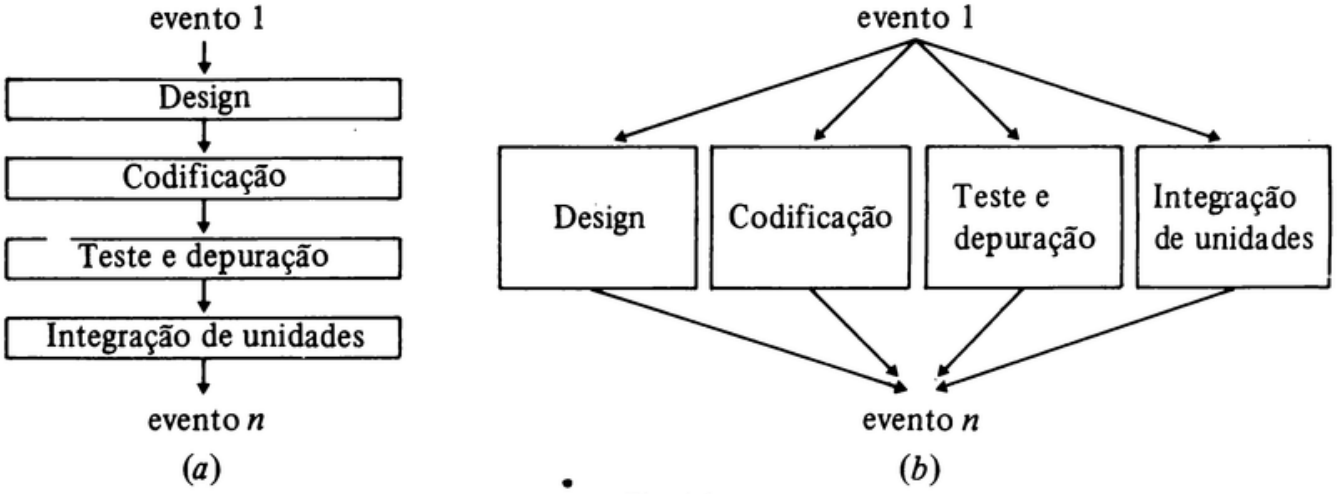


Fig. 4.1

Com a Programação Estruturada, podemos antecipar, o quanto quisermos, a entrada do programa em máquina. Como vimos, o desenvolvimento de um programa estruturado é feito por meio de escrita de várias versões do mesmo. Com exceção da última, as demais versões não são executáveis, pois suas funções são expressas em linguagem natural. Todavia, isto não impede que tais versões sejam testadas no computador. Com o uso de algum artifício, isto é sempre possível. Por exemplo, consideremos um programa em que uma versão genérica (que não seja a última) tem o seguinte texto:

```
start;
   i:= 1;
   do while i ⩽ 100;
      read tipo, nome, idade;
      case tipo
         1: call rot;
         2: "incluir registro"
         3: "excluir registro"
         4: if idade > 30
                then
                   print tipo, nome, idade;
                else
                   "modificar registro"
            fi;
      esac;
      i:= i + 1;
   od;
stop;
```

Esta versão não é executável, pois a rotina *rot* ainda não foi definida e as especificações *incluir registro, excluir registro* e *modificar registro* estão em linguagem natural. Para tornar esta versão executável, basta definir uma *rotina fantasma* que simule a *rot* e substituir as especificações em linguagem natural por comandos provisórios que imprimam mensagens indicativas da passagem do fluxo de controle por aqueles pontos. Teremos, então,

```
start;
   i: = 1;
   do while i ⩽ 100;
      read tipo, nome, idade;
      case tipo
         1: call rot;
         2: {PROVISORIO} print 'registro incluído';
         3: {PROVISORIO} print 'registro excluído';
         4: if idade > 30
               then
                  print tipo, nome, idade;
               else
                  {PROVISORIO} print 'registro modificado';
            fi;
      esac;
      i: = i + 1;
   od;
stop;
procedure: rot;
      start;
         {PROVISORIO} print 'rot chamada';
      return;
```

Desta forma, qualquer versão pode ser testada em máquina.

Os conceitos apresentados demonstram que as atividades de desenvolvimento de programas podem ser desdobradas de forma paralela (Fig. 4.1*b*) com o uso da Programação Estruturada.

Com este procedimento, alguns módulos do programa são mais testados que outros. A Fig. 4.2 apresenta um exemplo ilustrativo. Podemos notar que os módulos $A11$ e $A12$ participam de quatro testes, enquanto o módulo $A32$ participa apenas do último. Para tirar maior proveito dos testes, o programador deve desenvolver primeiro os módulos mais críticos para que os mesmos sejam testados mais vezes.

Na programação em equipe, o paralelismo das atividades (Fig. 4.1*b*) apresenta vantagens de ordem gerencial. A atribuição de uma tarefa (um subprograma, por exemplo) a um programador não implica na interrupção dos testes das versões onde esta unidade é chamada, pois elas poderão ser rodadas chamando aquela como um *subprograma-fantasma*. A integração do novo código, todavia, deve ser feita após a depuração em separado desta unidade e da versão na qual ela será inserida. Não é conveniente a integração de mais de uma unidade entre dois testes.

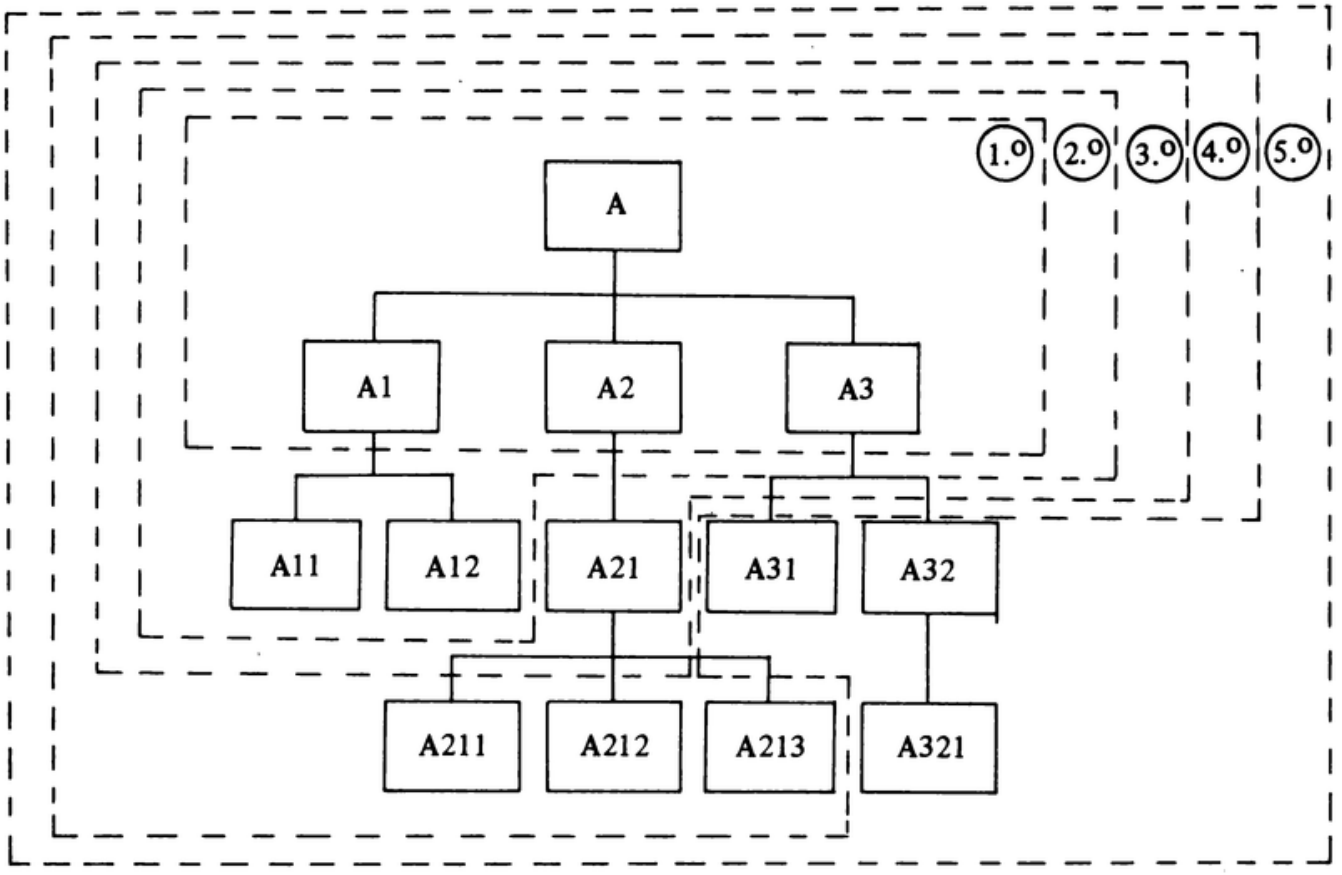


Fig. 4.2

4.1.2 – A Obtenção Intuitiva da Correção

Em termos práticos, a influência da nova técnica no que concerne à correção, constitui-se em tornar a atividade de programar *menos propensa a erros*.

O desenvolvimento por refinamentos sucessivos nos permite, inicialmente, subdividir um problema em subproblemas, a fim de superarmos a nossa incapacidade intelectual para dominar um grande número de variáveis ao mesmo tempo. Depois, utilizando a abstração, numa política de adiamento judicioso de decisões sobre o *design*, especificamos, primeiro, "*o que* uma operação deve fazer", para, mais tarde, refiná-la, dizendo "*como* irá operar". Isto caracteriza a escrita das várias versões de um programa estruturado. Após a criação de cada versão, é possível obtermos maior certeza da correção do programa até aquele momento, seja de maneira completamente informal e intuitiva ou pelo próprio teste do programa.

O uso restrito das estruturas de controle válidas evita naturalmente a inclusão indiscriminada de comandos de transferência de controle incondicional (GO TO). O principal prejuízo que o uso arbitrário deste tipo de comando pode apresentar está na liberdade que ele dá ao programador para a criação de *novas* estruturas de controle, cujo funcionamento, não-padronizado, deve ser investigado com muita atenção. Tal investigação pode tornar-se uma tarefa árdua para a mente humana devido ao grande número de caminhos de controle que o uso indiscriminado do GO TO pode inserir.

A experiência tem mostrado que os programas desenvolvidos sob esta disciplina apresentam versões finais com muito menos erros que os programas feitos de uma forma tradicional.

4.2 – QUALIDADES EXTERNAS DOS PROGRAMAS

Embora as vantagens apresentadas sobre a correção de programas por si sós justifiquem a aceitação que a nova técnica vem tendo, só em alguns dos fatores que afetam a qualidade dos programas é que os benefícios da Programação Estruturada são mais visíveis para a grande maioria das pessoas. Vejamos como a nova técnica se comporta perante cada um desses fatores.

4.2.1 – Adaptabilidade

A *adaptabilidade* de um programa é medida pelo esforço requerido para alterá-lo, a fim de satisfazer as mudanças em suas especificações. Este esforço depende de duas características do programa:

1.ª) *clareza*: de modo que a pessoa encarregada de alterá-lo possa entendê-lo com maior facilidade;

2.ª) *modularização*: que é a divisão do programa em módulos com a máxima independência uns dos outros.

Com respeito à primeira característica, a Programação Estruturada é vantajosa, como será visto no item referente à clareza.

A modularidade de um programa depende, inicialmente, da decisão sobre os critérios que devem orientar o desenvolvimento, constituindo-se em saber, *a priori*, o que pode ser mudado no programa ao longo de seu ciclo de vida. Com base nesses critérios, o desenvolvimento deve ser feito de modo a isolar as funções passíveis de modificações futuras em módulos independentes. A independência consiste em minimizar a conexão entre os módulos para evitar que qualquer mudança em um deles acarrete alterações em outros. Distinguimos, portanto, dois níveis de decisões na obtenção da modularidade: nível de *design* (que funções devem ser isoladas) e nível de implantação (como isolar estas funções). Neste último nível, as decisões dependem dos recursos da linguagem de implantação.

No nível de *design*, parece estar a grande vantagem da técnica. A facilidade que sua disciplina de decomposição do problema em subproblemas proporciona, parece ser o caminho natural para a obtenção de uma estrutura hierárquica de módulos bem definidos.

4.2.2 – Robustez

Sob este aspecto a Programação Estruturada é neutra. A robustez de um programa depende da inclusão de testes para detecção e localização de erros, blocos de recuperação e outros artifícios que permitam a ele continuar executando suas funções a despeito da ocorrência de falhas de operação (por exemplo, colocação de dados incorretos) ou de *hardware* (queda de força, ruptura de formulário contínuo durante a impressão).

A decisão de incluir ou não estes artifícios depende dos objetivos do programa. A facilidade de implantá-los é função dos recursos da linguagem, como, por exemplo, *on conditions* em PL/I e *declaratives* em COBOL.

4.2.3 – Desempenho

O desempenho reúne um conjunto de fatores que medem o consumo, por parte do programa, dos recursos disponíveis ao sistema. Vamos destacar aqui o fator eficiência. Ele é considerado como o conjunto de duas medidas: tempo de processamento e espaço de memória. Esta última, todavia, não exerce mais grande influência, principalmente em sistemas de grande porte, onde a memória torna-se cada vez mais barata, relativamente a outros recursos.

Segundo Yourdon [26], entre as objeções mais comuns que os programadores sêniors apresentam à introdução da Programação Estruturada (como parte da natural resistência à mudança) encontra-se a de que o "emprego da nova técnica leva à construção de programas tremendamente ineficientes".

Esta reação, na realidade, não tem uma base muito sólida. Talvez seu único fundamento cabível esteja na extrema modularidade que os programas estruturados tendem a obter, principalmente quando a adaptabilidade é um objetivo importante. O uso de muitas chamadas de sub-rotinas tende a diminuir a eficiência. Todavia, as considerações sobre eficiência não dependem tanto da técnica de programação em si, quanto das decisões de *design*.

A otimização de um programa não deve ser tentada durante o desenvolvimento, mas sim num segundo estágio. O primeiro deve ter em mente a obtenção de um produto correto sem preocupações de eficiência. A tentativa de otimizar a velocidade de execução, durante a primeira fase, pode levar a resultados ilusórios e ao agravamento dos custos de programação. Knuth [16] afirma que "muito do tempo de execução é concentrado em cerca de 3% do código-fonte". Então, existe um código crítico, localizado em alguns pontos do programa, que deve ser otimizado. Mas isto requer, antes, a sua identificação, o que pode ser feito com o auxílio do computador, através de geração do perfil do programa.

4.2.4 – Clareza

Enquanto as principais preocupações em termos de qualidade se concentravam na eficiência (minização do gasto de memória e maximização da velocidade de execução), o aumento da complexidade dos programas era uma conseqüência direta desta prática.

Alguns fatores têm modificado esta política, recentemente. A maior disponibilidade de memória nas grandes máquinas já não exige do programador tanta preocupação com economia de espaço. A fase de manutenção, por sua vez, é de grande importância em Sistemas de Processamento de Dados. Agora, a clareza dos programas é importante, pois eles devem ser lidos não só pelos compiladores ou montadores, mas também pelos seres humanos.

Com o uso da Programação Estruturada, a complexidade pode ser controlada com maior facilidade e o resultado são programas mais claros, onde a estrutura do texto reflete a estrutura da computação, sem deformações, tornando-os mais facilmente manuteníveis.

4.2.5 – Documentação e Forma

O fato de que "a única documentação confiável de um programa é o próprio código" (afirmação de Kernighan & Plauger [15]), sugere a concentração do programador neste sentido, libertando-o da preparação de fluxogramas.

O uso de comentários, a partir das primeiras versões do desenvolvimento, leva naturalmente à obtenção de um código final criteriosamente comentado. Para isto, basta iniciar cada

refinamento com um comentário que seja a especificação, em linguagem natural, da função que está sendo refinada (ver o exemplo da Seç. 3.4).

A escrita do código das estruturas de controle da Programação Estruturada, usadno margens verticais para indicar os aninhamentos, constitui-se numa facilidade de formatação do texto que torna a estrutura do programa mais visível.

- A reunião destas características facilita a obtenção de programas autodocumentáveis.

4.2.6 – Custos

O custo dos programadores, geralmente, se constitui no item que mais agrava os custos gerais de programação em Sistemas de Processamento de Dados de porte regular ou grande. Uma das maneiras de diminuir este item de custo é aumentar a produtividade dos programadores. Isto não significa, porém, aumentar o número de linhas de código concreto produzidas por unidade de tempo mas sim elevar a capacidade de produção de boas *soluções* para os problemas a serem resolvidos via programação. Neste particular, a Programação Estruturada desempenha um papel importante. Seu uso incentiva o adiamento da obtenção do código final até o momento em que o programador tenha certeza de que seu produto resolve corretamente o problema proposto. Os custos acrescidos por este adiamento são compensados pela redução do tempo gasto na depuração, uma vez que o procedimento exposto tende a evitar a criação de erros, em lugar de criá-los e depois corrigi-los. Baker e Mills [2] mostram que "de cinco a dez instruções depuradas são produzidas por homem-dia em um grande projeto. O tempo para codificar estas instruções não pode exceder alguns minutos de um dia de trabalho. Que fazem os programadores no tempo restante? Depuram". O uso da Programação Estruturada diminui substancialmente os custos dos programadores na fase de desenvolvimento de programas, devido ao aumento da produtividade que ele proporciona.

Posteriormente, na fase de manutenção, o custo dos programadores é reduzido com o uso da técnica, pois os programas estruturados são mais adaptáveis que os não estruturados.

Se levarmos em conta, ainda, que os custos de documentação final devem ser computados nos custos gerais, e considerando o fato de que a documentação de programas estruturados é subproduto natural de seu desenvolvimento, concluímos que o uso da técnica em apreço apresenta, sem dúvida, sensíveis vantagens em termos de redução dos custos de programação.

4.2.7 – Portabilidade

A *portabilidade* de um programa é a medida do esforço requerido para instalá-lo em outra máquina ou em uma configuração diferente da mesma máquina.

O problema da portabilidade de programas tem causado a elevação dos custos de *software*. Muito tempo tem sido perdido pelos programadores produzindo programas que já foram escritos antes, mas que não estão disponíveis nas formas aceitas pela sua instalação.

Tentou-se resolver este problema por meio de *linguagens orientadas para problemas*, que ajudam o programador a descrever um algoritmo em termos do problema e não da máquina. Era esperado que um programa deste tipo fosse transportável de uma máquina para outra por simples recompilação. Todavia, isto não tem sido inteiramente possível por várias razões. Uma delas é a grande proliferação de dialetos, pois cada compilador define uma linguagem-fonte diferente.

Em vista disto, a portabilidade de um programa continua a depender muito da facilidade que ele oferece para sua adaptação (conversão de uma linguagem para outra, de uma máquina para outra). Reside aí, portanto, a vantagem que o uso da Programação Estruturada apresenta em relação à portabilidade: os programas estruturados, por serem mais claros e adaptáveis, são mais facilmente transportados.

4.3 – CONCLUSÃO

O objetivo de mostrar, ao longo deste capítulo, as vantagens que o uso da Programação Estruturada pode trazer para o aumento da qualidade em programação, pode ter-se diluído na extensão do texto. Uma consolidação dessas idéias é, então, conveniente e vamos fazê-la segundo três aspectos dos programas:

- a correção;
- a construção; e
- as qualidades externas.

O primeiro deles deve ser enfatizado, pois a Programação Estruturada emergiu naturalmente da necessidade de formalizarmos a semântica de linguagens de programação, com vistas à facilidade de verificação da correção dos programas. Sob este prisma, a técnica atinge seus objetivos, pois permite a *obtenção de programas mais corretos*.

O segundo aspecto é importante se considerarmos que um programa, antes de *ser* uma peça física útil, deve *ser construído*, o que implica o consumo de recursos humanos e de máquina. A técnica se constitui numa ferramenta que auxilia o programador no *domínio da extensão*, permitindo a partição de problemas maiores em subproblemas menores e na *disciplina do raciocínio*, através de facilidade com que podemos usar, paralelamente, a linguagem natural e a de programação.

Algumas *qualidades externas*, por sua vez, serão obtidas com maior naturalidade se a técnica for empregada corretamente. Isto nos permite inferir que, de modo geral, os programas estruturados são mais adaptáveis, mais claros, autodocumentáveis, mais portáveis e mais baratos (considerando programas para Sistemas de Processamento de Dados).

Parte II

O Emprego da Programação Estruturada em COBOL

Capítulo 5

Estruturação Básica em COBOL

Implementar Programação Estruturada em linguagem disponível signica achar soluções para exprimir, nessa linguagem, os dois elementos da técnica: a *estruturação básica* e o *desenvolvimento por refinamentos sucessivos*. Para o primeiro elemento, a tarefa se resume, basicamente, em definir a implementação das estruturas de controle válidas (ou pelo menos, um número mínimo satisfatório delas) na linguagem em questão. Para o segundo, deve-se encontrar uma forma de desenvolver programas na linguagem de modo a explorar os benefícios que a Programação Estruturada apresenta neste particular.

A proposição para o emprego da técnica em COBOL, que apresentamos a seguir, reflete a preocupação em preservar ao máximo as características principais tanto da Programação Estruturada quanto da linguagem em si. As soluções apresentadas buscam mais harmonizar as características das duas partes do que, simplesmente, forçar o emprego da técnica em sua forma mais pura, o que poderia causar deformações exageradas no uso da linguagem COBOL.

Esta proposta é limitada ao uso do COBOL ANS em sua forma atual, sem a previsão do emprego de pré-processadores.

5.1 – ESTRUTURAS DE CONTROLE

Apresentamos, a seguir, uma forma de implementação das seis estruturas de controle válidas na Programação Estruturada, apresentadas anteriormente.

5.1.1 – Seqüência

A decomposição de uma computação em um número finito de ações seqüenciais não oferece, em geral, a menor dificuldade em qualquer linguagem de programação. O COBOL não é uma exceção desta regra. Um exemplo serve para mostrá-lo.

Exemplo.

```
MOVE G–SOMAQT–CT1 TO IC1–SOMAQT–DT1.
WRITE F–LINHA FROM IC1–DET1 AFTER ADVANCING 1 LINES.
ADD 3 TO G–NLINHA.
```

5.1.2 – "If-then-else" e "If-then"

O comando IF do COBOL é uma implementação imediata da estrutura de selação IF–THEN–ELSE da Programação Estruturada. O formato que propomos para esta estrutura, em COBOL, é o seguinte:

```
IF condição
   THEN
       comando–1
   ELSE
       comando–2*
```

Exemplo.

```
IF G–DESCRI–CT2 (PSC–IND, IC2–COL) IS = SPACES
   THEN
       MOVE 6 TO IC2–COL
   ELSE
       PERFORM ID3–IMPRIMIR–DETALHE–3 THRU
          DETALHE–3–IMPRESSO.
```

A estrutura IF–THEN, que é o caso particular de um IF–THEN–ELSE com a parte ELSE *nula*, tem a seguinte forma em COBOL estruturado:

```
IF condição
   THEN
       comando
   ELSE
       NEXT SENTENCE
```

A cláusula ELSE NEXT SENTENCE poderá ser omitida se ela preceder imediatamente o ponto terminal da sentença.

Exemplo.

```
IF TAXA IS GREATER THAN 0.05
   THEN
       IF CAPITAL IS EQUAL TO 100.00
          THEN
              ADD 25.000 TO CAPITAL
          ELSE
              NEXT SENTENCE
   ELSE
       IF TAXA IS EQUAL TO 0.05
          THEN
              PERFORM CALCULAR–JUROS THRU JUROS–CALCULADOS
          ELSE
              NEXT SENTENCE.
próxima sentença . . . .
```

---

* Neste contexto, a palavra *comando* poder ser entendida como *grupo de comandos.*

Neste exemplo, a cláusula final

ELSE NEXT SENTENCE

é opcional, pois ela precede imediatamente o ponto terminal da sentença. A cláusula ELSE do comando

IF CAPITAL IS EQUAL TO 100.00,

por sua vez, não pode ser omitida.

A cláusula THEN é uma palavra opcional em algumas versões do COBOL (por exemplo, IBM/370 [5]), mas não existe em outras (por exemplo, B/6700 da Burrought [1]). Neste último caso, recomenda-se o uso da cláusula como um comentário, apenas para efeito de consistência com a formatação do esquema na Programação Estruturada e para boa documentação.

Um grande problema da linguagem COBOL, no que diz respeito ao seu emprego em Programação Estruturda, é o *aninhamento*. Ao aninharmos dois IF's, devemos cuidar para que o caminho entre a saída do IF interno e a do IF externo não tenha nenhum comando. Isto se deve às características de pontuação do COBOL, que não permite *ninho* de sentenças. Por exemplo, a implementação mais natural para o fluxograma da Fig. 5.1 seria o *ninho* de IF's que é apresentado logo a seguir.

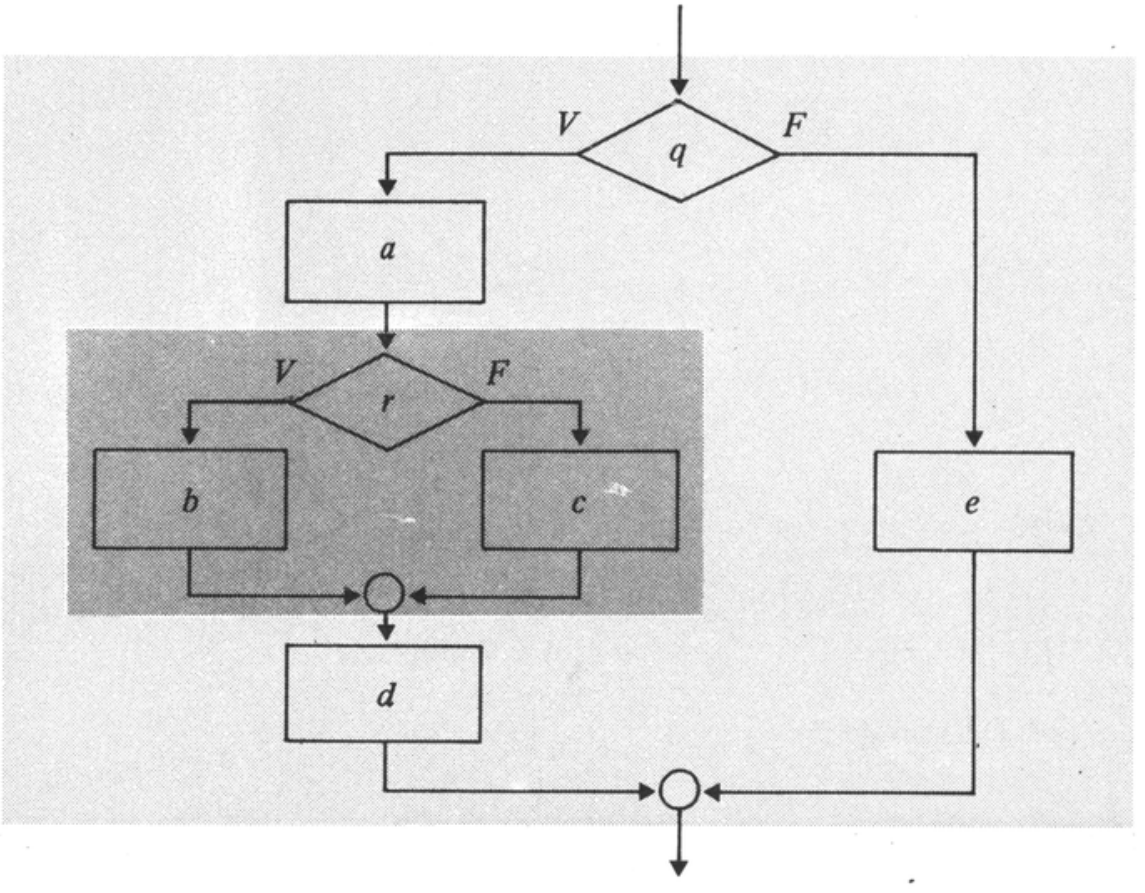
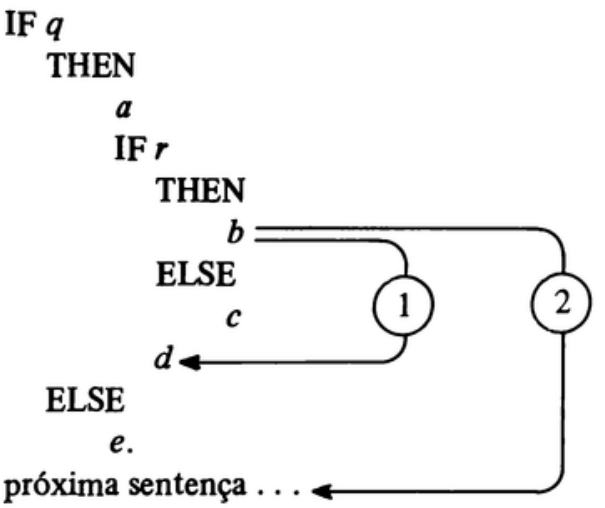


Fig. 5.1



Todavia, esta implementação em COBOL, é errada, pois no caso de a condição *r* do IF interno ser verdadeira, o comando *b* será executado e, logo após, o controle passará para a "próxima sentença" (caminho 2), fazendo com que *d* pertença unicamente à cláusula ELSE daquele IF. Esta não é a intenção expressa no fluxograma, que prevê a execução do comando *d* tanto após *c* como após *b* (caminho 1). Usando ninho de IF's, a solução para este caso é a duplicação do código do comando *d*, incluindo-o também dentro da cláusula THEN do IF interno:

```
IF q
   THEN
        a
        IF r
           THEN
                b
                d
           ELSE
                c
                d
   ELSE
        e.
próxima sentença . . .
```

5.1.3 – "Case"

A estrutura CASE não existe em COBOL. Sua implementação pode ser feita através do comando GO TO DEPENDING ON. Assim, o trecho de programa

```
case identificador
   l1: comando–1
   l2: comando–2
         .
         .
         .
   ln: comando–n
esac
```

é codificado em COBOL por:

```
        GO TO parágrafo–1, parágrafo–2, . . . , parágrafo–n
           DEPENDING ON identificador.
parágrafo–erro.
                comando
                GO TO parágrafo–fim.
parágrafo–1.
                comando–1
                GO TO parágrafo–fim.
parágrafo–2.
                comando–2
                GO TO parágrafo–fim.
        . . . . . .
        . . . . . .
parágrafo–n.
                comando–n.
parágrafo–fim. EXIT.
```

O "parágrafo–erro" é uma procedure que resolve o caso em que "identificador" não assume nenhum valor de 1 até *n*. Quando isto ocorre, o comando GO TO DEPENDING ON é ignorado e o parágrafo–erro deve prever as providências para esta eventualidade.

Exemplo.

```
        GO TO CT1–CARTAO–TIPO–1, CT2–CARTAO–TIPO–2,
           DEPENDING ON OL–TIPO.

OT–OUTRO–TIPO.
        MOVE 'CARTAO TIPO INVALIDO                    ' TO G–ERRO–AR.
        PERFORM RC–REJEITAR–CARTAO THRU CARTAO–REJEITADO.
        GO TO LOTE–OBTIDO.

CT1–CARTAO–TIPO–1.
        MOVE 1 TO LL–FIMLOTE.
        GO TO LOTE–OBTIDO.

CT2–CARTAO–TIPO–2.
        IF LL–PONTEIRO IS > 40 OR LL–PONTEIRO IS = 40
           THEN
              PERFORM RFL–REJEITAR–ATE–FIM–LOTE THRU
                 ATE–FIM–LOTE–REJEITADOS
              MOVE 1 TO  G–LOTEREJ, LL–FIMLOTE
           ELSE
              ADD 1 TO LL–PONTEIRO
              MOVE F–CARTAO TO G–CARTAO2  (LL–PONTEIRO).

LOTE–OBTIDO. EXIT.
```

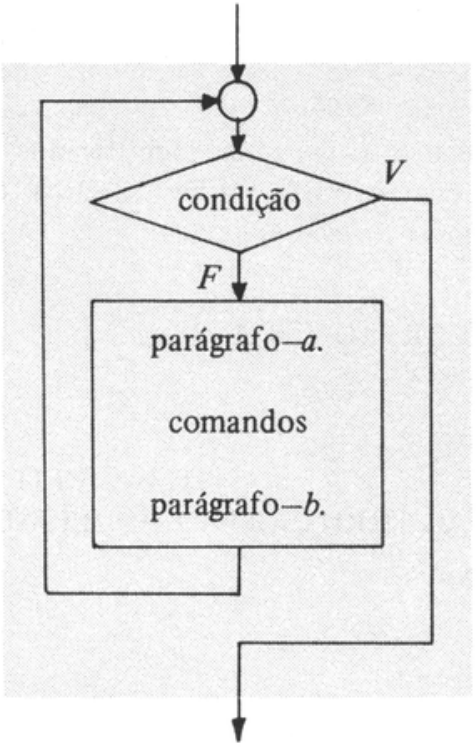
Neste caso, aproveitamos o cabeçalho de parágrafo "LOTE–OBTIDO" como fechamento para o CASE.

5.1.4 – "Do-while"

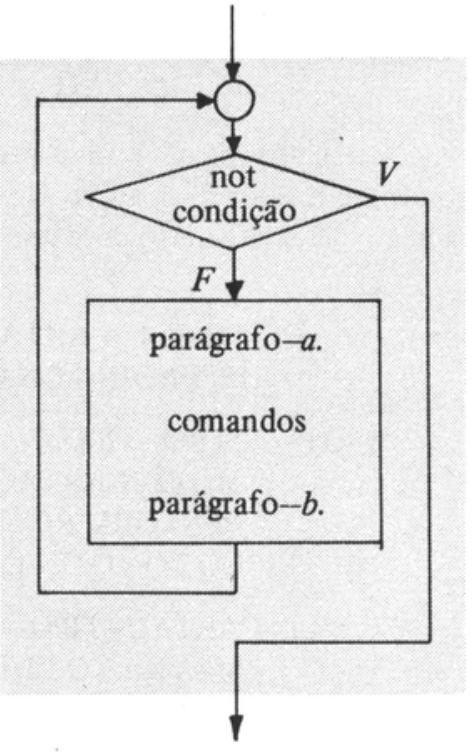
A estrutura de controle DO–WHILE também não existe em COBOL em sua forma pura, como ocorre em PASCAL, ALGOL-W e PL/I. Todavia, uma pequena adaptação ao comando PERFORM UNTIL tende a resolver o problema, se bem que não de forma ideal. A sintaxe deste comando COBOL é

PERFORM parágrafo–*a* [THRU parágrafo–*b*] UNTIL condição.

Ele causa a execução repetida dos comandos entre parágrafo–*a* e parágrafo–*b* até que a *condição* seja satisfeita. O processamento, então, continua a partir do comando seguinte ao PERFORM. O fluxograma (*a*) da Fig. 5.2 apresenta o fluxo de controle do PERFORM–UNTIL.



(*a*) PERFORM–UNTIL



(*b*) DO–WHILE, em COBOL

Fig. 5.2

Embora a palavra UNTIL possa nos induzir a ver neste comando alguma semelhança com o REPEAT–UNTIL da Programação Estruturada, na realidade seu fluxograma tem uma forma bem mais próxima do DO–WHILE, pois apresenta uma característica importante desta estrutura de controle: o teste antes dos comandos a serem repetidos. Uma alteração, todavia, deve ser feita para a implementação do DO–WHILE através do PERFORM UNTIL. A saída da estrutura ocorre com a verificação da verdade da condição, o que é oposto ao DO–WHILE da Programação Estruturada, onde a saída deve ocorrer somente quando a condição for falsa. A simples negação da condição, no teste, resolve esta discrepância.

A implementação da estrutura

```
do while condição
   comando
od
```

é feita então, por

```
PERFORM parágrafo–a THRU parágrafo–b
   UNTIL NOT condição.
. . . . . .
. . . . . .
parágrafo–a.
   comando.
parágrafo–b.
```

Exemplo.

```
*    _ENQUANTO G–LINHA–TB1 (ICL–INDICE) = 0, IMPRIMIR MENSA
*        GEM
         PERFORM IM–IMPRIMIR–MENSAGEM THRU MENSAGEM–IMP
           RESSA UNTIL G–LINHA–TB1  (ICL–INDICE) = 0.
         . . . . . .
         . . . . . .
     IM–IMPRIMIR–MENSAGEM.
         IF G–NLINHA IS > 70
           THEN
              PERFORM IC–IMPRIMIR–CABECALHO THRU
                 CABECALHO–IMPRESSO.
         MOVE SPACES TO IC1–DET1.
         MOVE G–LINHA–TB1  (ICL–INDICE) TO ICL–ERRO.
         WRITE F–LINHA FROM IC1–DET1 AFTER ADVANCING 1 LINES.
         ADD 1 TO G–NLINHA.
         ADD 1 TO ICL–INDICE.
     MENSAGEM–IMPRESSA.
```

5.1.5 – "Repeat-until"

Esta estrutura de controle também não existe em COBOL na sua forma definida pela Programação Estruturada. As características diferenciadoras deste esquema de repetição em relação ao DO–WHILE são:

a) a saída do fluxo de controle da estrutura determinada pela verificação da condição (e não pela sua negação como ocorre no DO–WHILE);

b) a execução, pelo menos uma vez, dos comandos constituintes da estrutura, causada pela disposição do teste após estes comandos.

Evidentemente, estas duas características devem ser mantidas rigorosamente na implementação desta estrutura em COBOL. Assim, uma boa implementação para

```
repeat
   comando
until condição
```

é a seguinte:

```
        PERFORM parágrafo–a THRU parágrafo–b
        PERFORM parágrafo–a THRU parágrafo–b UNTIL condição.
        . . . . . .
        . . . . . .
parágrafo–a.
    comando.
parágrafo–b.
```

O fluxograma desta construção é apresentado a seguir (Fig. 5.3).

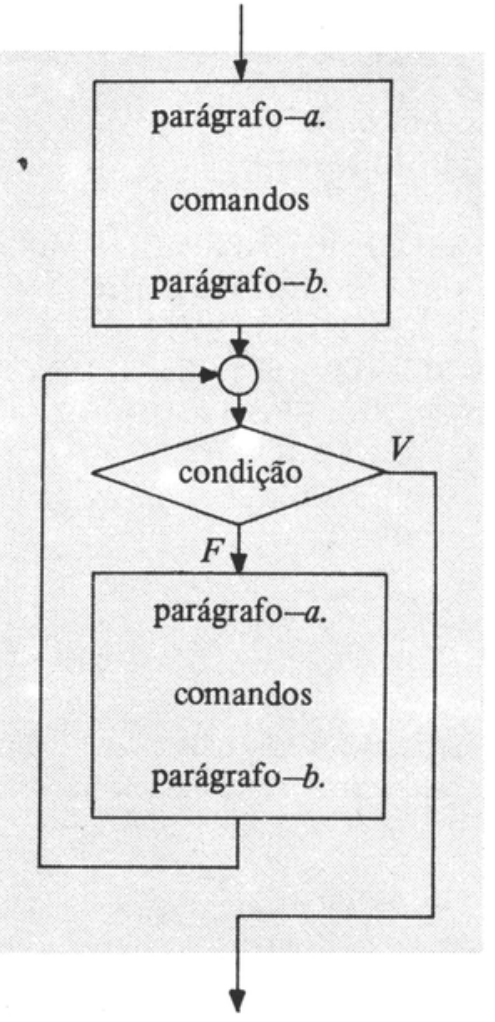


Fig. 5.3

Exemplo.

```
*   _REPETIR MOVER DESCRICAO PRODUTO ATE QUE GAP-IND = 5.
        PERFORM MDP-MOVER-DESCRICAO-PRODUTO THRU
          DESCRICAO-PRODUTO-MOVIDA.
        PERFORM MDP-MOVER-DESCRICAO-PRODUTO THRU
          DESCRICAO-PRODUTO-MOVIDA
        UNTIL GAP-IND = 5.
        . . . . . .
        . . . . . .
MDP-MOVER-DESCRICAO-PRODUTO.
        MOVE G-DESCRI-CT2 (PSC-IND, GAP-IND) TO F-DESCRI-PED
          (GAP-IND).
        ADD 1 TO GAP-IND.
DESCRICAO-PRODUTO-MOVIDA.
```

5.2 – O PROBLEMA DO ANINHAMENTO NAS ESTRUTURAS DE REPETIÇÃO

Da maneira como foram apresentadas as implementações das estruturas DO–WHILE e REPEAT–UNTIL, o aninhamento de estruturas deixa de ser possível em sua forma física, no texto, pois um ninho dessas estruturas será representado por parágrafos colocados em seqüência e não um embutido no outro.

Conforme já demonstramos no início deste capítulo, é nossa intenção, ao tentar implementar a Programação Estruturada em COBOL, resguardar ao máximo as características de ambas.

A implementação do DO–WHILE e do REPEAT–UNTIL com o uso de comandos GO TO poderia resolver o problema do aninhamento. Uma forma de codificar estas estruturas seria:

DO WHILE:

```
        PERFORM parágrafo–a THRU parágrafo–b
        UNTIL NOT condição, GO TO parágrafo–b.
parágrafo–a.
        comando.
parágrafo–b. EXIT.
```

REPEAT–UNTIL:

```
        PERFORM parágrafo–a THRU parágrafo–b
        PERFORM parágrafo–a THRU parágrafo–b.
        UNTIL condição, GO TO parágrafo–b.
parágrafo–a.
        comando.
parágrafo–b. EXIT.
```

Consideramos que, se por um lado, estas formas (que não são as únicas possíveis) resolvem o problema do aninhamento, por outro lado, destroem uma característica fundamental das estruturas de controle da Programação Estruturada, que é o seu alto nível. A introdução do GO TO, apesar de criteriosa, baixa o nível dessas estruturas aos olhos do programador ou do

leitor do texto fonte, além de possibilitar a introdução de aninhamentos de cabeçalhos de parágrafos na margem A, que podem tornar confuso o texto do programa.

A colocação, logo após o PERFORM, dos comandos a serem executados pela sua ação, não é uma característica atual da programação em COBOL (dizemos *atual* porque, no futuro, como atestam as pesquisas, poderá haver uma nova versão do PERFORM em que os comandos a serem executados seguem-se imediatamente a ele, mas sem precisar de parágrafos).

Estas formas de implementação das estruturas de repetição com o uso de GO TO's são perfeitamente válidas, quando feitas por um pré-processador, de modo a se tornarem invisíveis ao programador. Todavia, não queremos discutir aqui o mérito do uso de pré-processadores para a implementação da Programação Estruturada, por fugir aos objetivos deste livro.

A implementação proposta (sem GO TO's) oferece, na realidade, uma vantagem sob o ponto de vista da visualização, através do texto, da hierarquia do programa. Os níveis aparecem em seqüência, começando pelos mais altos e dispostos em ordem decrescente.

Capítulo 6

Desenvolvimento por Refinamentos Sucessivos em COBOL

O desenvolvimento de um programa estruturado por refinamentos sucessivos, pelo uso concomitante das linguagens natural e de programação, depende, fundamentalmente, da adaptabilidade desta última às características da *estruturação básica* da Programação Estruturada.

Em linguagens como PL/I, ALGOL, PASCAL e FORTRAN (WATFIV-S), por exemplo, realizamos o desenvolvimento da lógica juntamente com a codificação do programa. Cada versão, com excessão da última, possui um texto formado pela intercalação de comandos escritos em dois tipos de linguagem: a natural e a de programação.

A linguagem COBOL, como vimos no Cap. 5, não se adapta muito bem à *estruturação básica* da Programação Estruturada, pois exige artifícios para a implementação de algumas estruturas de controle importantes, apresentando ainda sérias restrições quando ao aninhamento. Em vista disto, não é aconselhável usar o COBOL juntamente com a linguagem natural durante a escrita das versões.

Uma forma de contornar este problema é fazer o desdobramento da fase de desenvolvimento do programa em duas subfases: o desenvolvimento lógico e a codificação (Fig. 6.1). Na primeira subfase, fazemos o desenvolvimento propriamente dito utilizando a escrita das várias

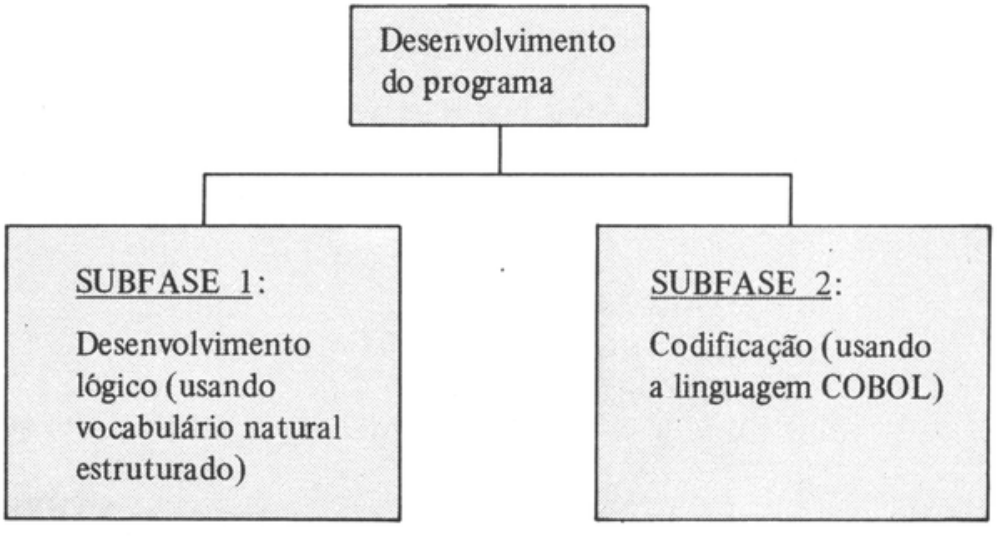


Fig. 6.1

versões e um vocabulário natural estruturado; a segunda subfase, que ocorre sempre que se deseja rodar em máquina alguma versão do programa, é simplesmente a codificação em COBOL do que foi escrito no vocabulário natural.

6.1 – O VOCABULÁRIO NATURAL ESTRUTURADO

O que aqui chamamos de *vocabulário natural estruturado*, nada mais é do que a própria linguagem natural do programador, usada com poucas restrições. Seu objetivo é funcionar como uma linguagem de programação de algo nível, que permita ampla liberdade de expressão das abstrações, em seus vários níveis, além de possuir todas as características da Programação Estruturada e alguma influência da linguagem COBOL.

O vocabulário natural estruturado que apresentamos a seguir pode ser descrito por meio de seus dois únicos e simples elementos:

a) *estruturas de controle.* São apresentadas na Tab. 6.1 as seis estruturas válidas abordadas anteriormente, sua forma em nosso vocabulário e sua correspondente codificação em COBOL;

b) *comandos abstratos*, que são palavras ou orações escritos em português, indicando ações, processos ou variáveis do programa, refletindo o raciocínio do programador. Os comandos abstratos são escritos entre aspas.

Usamos *dois pontos* para indicar a presença de um rótulo (*label*) ou nome de *procedure*.

Exemplo.

a) *Desenvolvimento lógico*

```
Rejeitar até o fim do lote:
   "mover mensagem '40 cartões' para área de saída"
   "imprimir cartão tipo 1"
   "rejeitar cartões da tabela"
   enquanto "tipo não é 1 e há lote"
      "mover mensagem 'cartão rejeitado' para área de rejeição"
      "rejeitar cartão"
      "ler cartão, ao fim de arquivo, não há lote"
   fim-enquanto.
```

b) *Codificação COBOL*

```
RFL-REJEITAR-ATE-FIM-LOTE.
    MOVE 'LOTE COM MAIS DE 40 CARTOES                    ' TO
      IC1-ERRO-DT1.
    PERFORM IC1-IMPRIMIR-CARTAO-1 THRU CARTAO-1-IMPRE
-       SSO.
    PERFORM RCT-REJEITAR-CARTOES-TABELA THRU
      CARTOES-TABELA-REJEITADOS.
```

Tabela 6.1. Estruturas de Controle*

Nome	*Forma Original*	*Vocabulário Natural Estruturado*	*COBOL*
IF–THEN–ELSE	if condição then comando else comando fi	se condição então comando se não comando fim-se	IF condição THEN comando ELSE comando
IF–THEN	if condição then comando fi	se condição então comando fim-se	IF condição THEN comando
CASE	case expressão L1: comando L2: comando ..... L*n*: comando esac	para caso–1: comando caso–2: comando ..... caso–*n*: comando fim-para	GO TO parágrafo–1, . . . , parágrafo–*n* DEPENDING ON identificador. parágrafo–1. comando GO TO parágrafo–fim. ..... parágrafo–*n*. comando parágrafo–fim. EXIT.
DO–WHILE	do while condição comando od	enquanto condição comando fim-enquanto	PERFORM parágrafo–1 THRU parágrafo–2 UNTIL negação da condição. ..... parágrafo–1. comando parágrafo–2.
REPEAT–UNTIL	repeat comando until condição	repetir comando até que condição	PERFORM parágrafo–1 THRU parágrafo–2 PERFORM parágrafo–1 THRU parágrafo–2 UNTIL condição. ..... parágrafo–1. comando parágrafo–2.

* A forma original da SEQÜÊNCIA é válida tanto no vocabulário natural estruturado como em COBOL.

```
*     _ENQUANTO F-TIPO-CT IS NOT = 1 AND G-HALOTE IS = 1, LER
*                                                    REJEITAR.
          PERFORM LR-LER-REJEITAR THRU LIDO-REJEITADO
          UNTIL F-TIPO-CT IS = 1 OR G-HALOTE IS NOT = 1.
ATE-FIM-LOTE-REJEITADOS.

LR-LER-REJEITAR.
    MOVE 'LOTE ERRADO: CARTAO REJEITADO   ' TO G-ERRO-AR.
    PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
    READ CARTAO
      AT END MOVE 0 TO G-HALOTE.
LIDO-REJEITADO.
```

6.2 – A DISTRIBUIÇÃO DO TEXTO NA "PROCEDURE DIVISION"

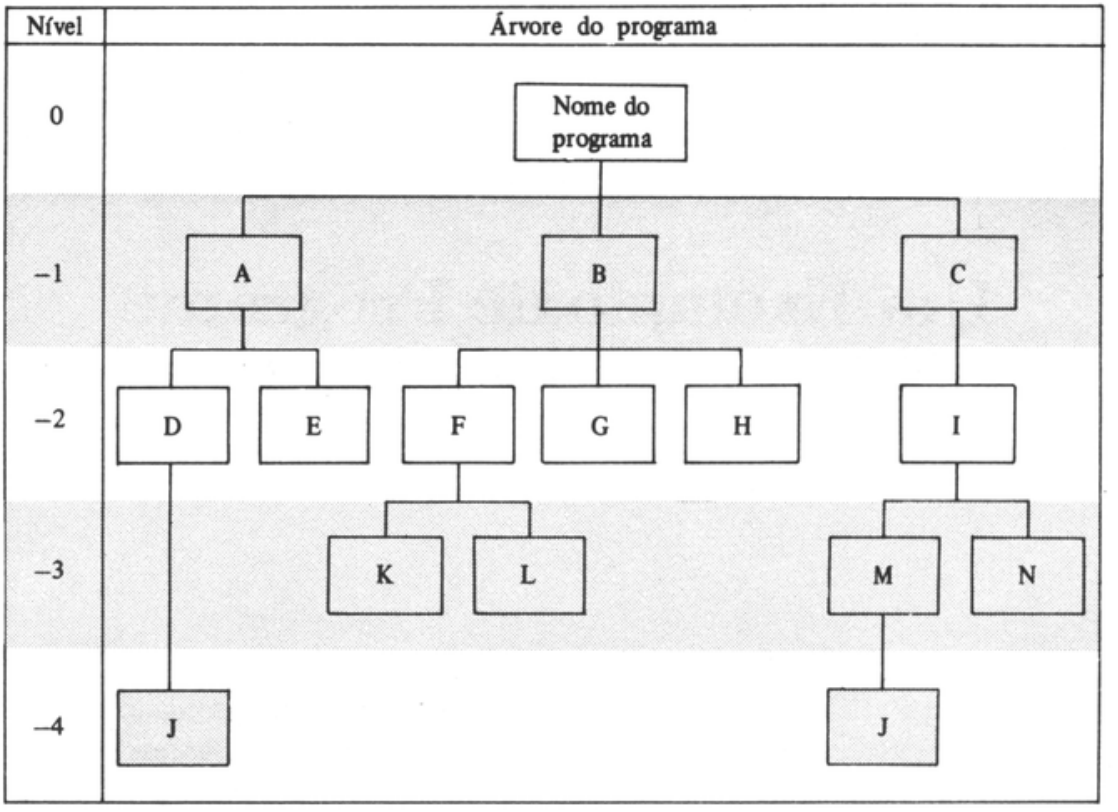
Utilizamos o comando PERFORM em programas estruturados em COBOL de duas maneiras: para implementar as estruturas de repetição DO–WHILE e REPEAT–UNTIL ou simplesmente como chamada de uma seção ou parágrafo.

Devido à utilização em larga escala deste comando, a *Procedure Division* de um programa estruturado em COBOL deve dispor seu texto-fonte de forma a refletir o encaminhamento "TOP–DOWN" de sua árvore, isto é, nível-a-nível, a partir do mais alto.

Em geral, a *Procedure Division* terá, no início, o módulo principal de controle, que resume todo o processamento. Este módulo contém as *procedures* do nível – 1 da árvore do programa. A seguir, aparece a codificação dos nós do nível seguinte, na ordem em que eles aparecem na árvore e assim sucessivamente para os níveis mais baixos. Quando um parágrafo existe unicamente para implementar uma estrutura DO–WHILE ou REPEAT–UNTIL, sua codificação deve vir logo após o parágrafo ou seção que o chama, e ele não consta na árvore, sendo considerado como parte do parágrafo (ou seção) chamador.

A Fig. 6.2 mostra um exemplo onde temos em (*b*) a distribuição do texto da *Procedure Division* correspondente ao programa cuja árvore é dada em (*a*). O parágrafo *J* é chamado por dois parágrafos e, por isso, deve ser disposto em seu nível mais baixo.

A árvore como elemento de documentação é muito importante para a leitura e a compreensão do programa.



(*a*) Árvore do programa

```
PROCEDURE DIVISION.
-------------------------------
          NIVEL –1
-------------------------------
    A
    B
    C
-------------------------------
          NIVEL –2
-------------------------------
    D
    E
    F
    G
    H
    I
-------------------------------
          NIVEL –3
-------------------------------
    K
    L
    M
    N
-------------------------------
          NIVEL –4
-------------------------------
    J
```

(*b*) Disposição do texto-fonte

Fig. 6.2

Capítulo 7

Um Exemplo de Programa Estruturado em COBOL

O exemplo apresentado a seguir tem por objetivo mostrar uma das maneiras de usarmos a Programação Estruturada em COBOL, de acordo com os elementos apresentados até aqui. Escolhemos, para tanto, um programa de crítica, típico de um Sistema de Processamento de Dados Comercial. Embora o exemplo procure ser o mais real possível, certos detalhes não foram levados em conta, como a escolha de uma estrutura de arquivos ideal para o caso e o relacionamento do programa com as demais partes do sistema. Esta simplificação visa a evitar que a atenção do leitor seja desviada para detalhes que não dizem respeito direto ao próprio processo de programação, o que pretendemos destacar.

7.1 – ESPECIFICAÇÃO DO PROGRAMA

Seja um Sistema de Administração de Compra e Venda de uma indústria de componentes eletrônicos em geral. Esta empresa distribui seus produtos para uma grande rede de clientes composta de universidades, emissoras de rádio e televisão, lojas revendedoras de materiais eletrônicos etc.

Um de seus subsistemas, o de Processamento de Ordens de Venda dos Clientes, recebe as ordens de venda, verifica a sua validade examinando as condições de crédito do cliente, calcula e emite a fatura correspondente aos itens solicitados e que estão disponíveis no estoque, tudo automaticamente por meio de computador.

Dentro deste subsistema, vamos desenvolver o programa de crítica dos lotes de ordens de venda.

O fluxograma do procedimento é apresentado na Fig. 7.1, onde destacamos o programa que deverá ser feito.

7.1.1 – Descrição das Entradas

As ordens de venda são transcritas em cartões. Cada ordem de venda pode ocupar de um a dois cartões do tipo 2, que devem ser agrupados em lotes de até 20 ordens, precedidos por um cartão tipo 1, que contém informações e totais de controle do lote correspondente. A quanti-

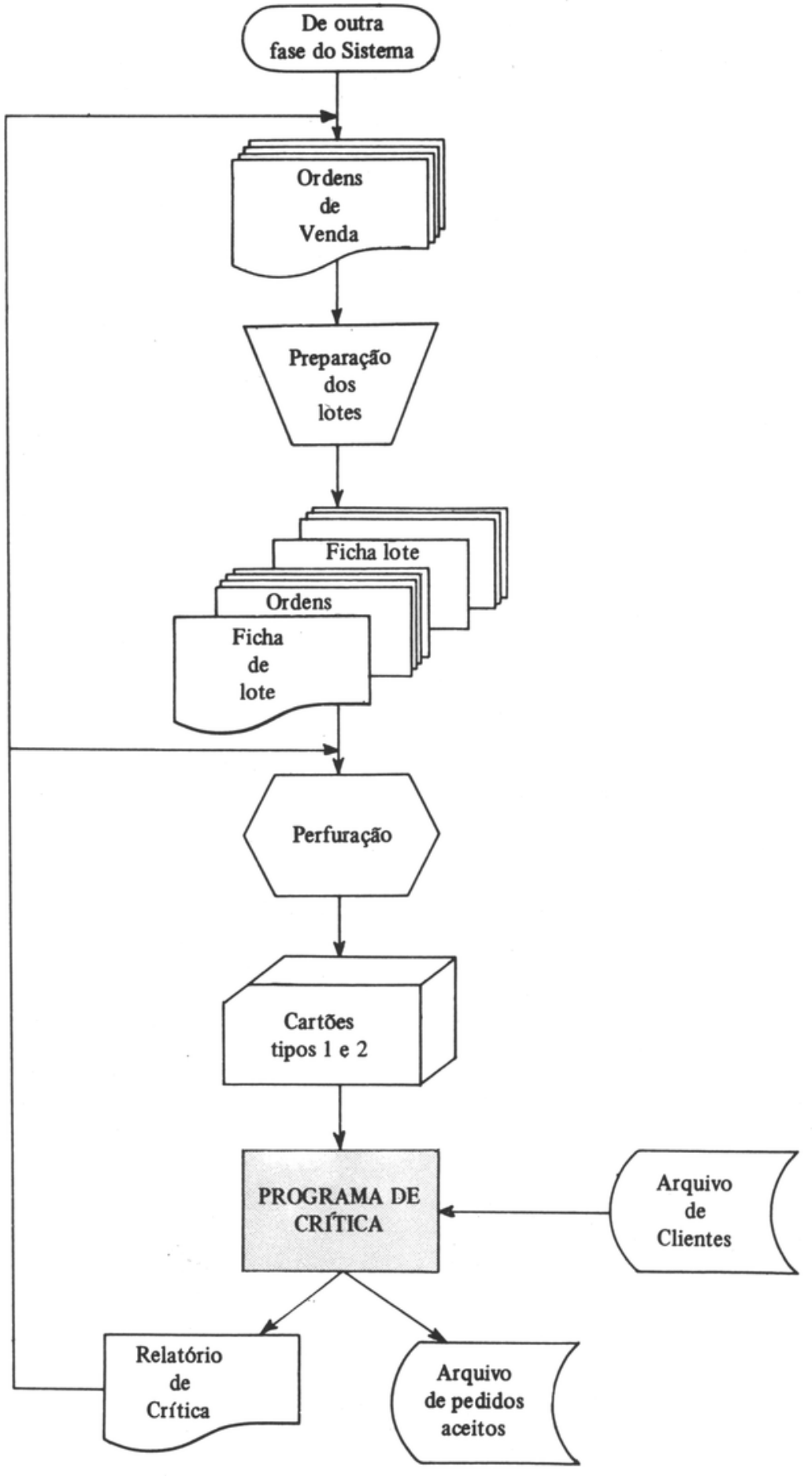

De outra fase do Sistema
Ordens de Venda
Preparação dos lotes
Ficha lote
Ordens
Ficha de lote
Perfuração
Cartões tipos 1 e 2
PROGRAMA DE CRÍTICA
Arquivo de Clientes
Relatório de Crítica
Arquivo de pedidos aceitos

Fig. 7.1

dade de lotes em cada rodada é indeterminada. Os *lay-outs* dos cartões de entrada são apresentados na Fig. 7.2*a*.

O arquivo de clientes (ARQCLI) serve também como entrada para o programa de crítica, pois contém todas as informações necessárias ao cadastramento e à conta corrente dos clientes. É um arquivo em disco magnético, organização indexada seqüencial, acesso direto, cujo *lay-out* do registro é apresentado na Fig. 7.2*b*.

Cartão tipo 1

1	N.º Lote	Data do Lote			Quant. de Ordens do Lote	Soma dos Números das Ordens	Soma das Quantidades		
		Dia	Mês	Ano					
N	N	N	N	N	N	N	N	B	B
1	2–3	4–5	6–7	8–9	10–11	12–18	19–23	24	– 80

Cartão tipo 2

2	Identificação do Cliente		N.º da Ordem Venda	Data Compra			Produto		Quantidade		
	Matr.	D.C.		Dia	Mês	Ano	Código	D.C.			
N	N	N	N	N	N	N	N	N	N	N	B
1	2–7	8	9–13	14–15	16–17	18–19	20–27	28	29–31		80

|← até 5 destes grupos →| por cartão

(*a*) *Lay-out* dos cartões de entrada

	Identificação do Cliente		Nome do Cliente	Endereço do Cliente	CGC	Limite de Crédito	Cond. de Créd.
	Matr.	D.C.					
	N	N	AN	AN	N	N	N
0	1–6	7	8–37	38–81	82–99	100–108	109

Conta Corrente							
Cód. Ocup.	N.º da Fatura	Valor	Lançamento				
			Dia	Mês	Ano		
N	N	N	N	N	N	N	
110	111–117	118–126	127–128	129–130	131–132	133	178

|← 3 vezes →|

(*b*) *Lay-out* do arquivo de clientes

Convenções: AN = campo alfanumérico; N = campo numérico; B = campo branco; ☐ = chave de pesquisa

Fig. 7.2

7.1.2 – Descrição das Saídas

O programa tem dois tipos de saídas:

a) um relatório de crítica, contendo somente os lotes ou cartões errados (*lay-out* da Fig. 7.3*a*); e

b) um arquivo de pedidos aceitos (ARQPED), contendo somente os conteúdos dos cartões certos. Este arquivo reside em disco magnético, com organização indexada seqüencial e acesso seqüencial (*lay-out* da Fig. 7.3*b*).

```
SISTEMA DE ADMINISTRAÇÃO DE COMPRA E VENDA                    PAGINA: XXX
SUBSISTEMA DE PROCESSAMENTO DE ORDENS

RELATORIO DE CRITICA DE LOTES DE ORDENS DE VENDA          DATA : XX/XX/XX

  TIPO  NO. LOTE    DATA LOTE    QT. ORDENS  SOMA NUM.  SOMA QUANT.   ERRO
        IDENTIF.    NO. DA ORDEM DATA COMPRA  PRODUTO    QUANT.

LOTE NUMERO: XX
   X      XX          XXXXXX        XX       XXXXXXX     XXXXX     X———————X
          XXXXXXX     XXXXX      XX/XX/XX    XXXXXXXXX   XXX       X———————X
                                             XXXXXXXXX   XXX       X———————X
                                             XXXXXXXXX   XXX       X———————X
```

(*a*) *Lay-out* do relatório de crítica

	Chave de Pesquisa	Número da Ordem	Identif. do Cliente	Data da Compra	Produto	Quantidade		
	N	N	N	N	N	N		N
0	1–4	5–9	10–16	17–22	23–31	32–34	35	82

5 vezes

(*b*) *Lay-out* do arquivo de pedidos aceitos

Fig. 7.3

7.1.3 – Descrição do Processamento

O programa deve executar o seguinte procedimento:

a) ler os cartões e analisá-los quanto aos detalhes apresentados na Tab. 7.1;

b) verificar a existência do cliente no arquivo de clientes; se não existir, imprimir a mensagem

CLIENTE NÃO CADASTRADO

e desprezar o pedido do cliente;

c) verificar a disponibilidade de crédito, consultando o campo "condição do crédito" no registro do cliente no arquivo ARQCLI; se o valor do campo for 1, o crédito é bom; se for 0, o crédito está abalado e o pedido deve ser desprezado com a mensagem

CLIENTE COM CRÉDITO SUSPENSO;

d) verificar a conta corrente do cliente no ARQCLI; o campo "conta corrente" do registro do cliente contém as informações sobre as faturas de que o cliente é credor; o número máximo permitido de faturas em aberto, por cliente, é três; se o subcampo "código de ocupação" for 1, então a fatura correspondente está em aberto; se as três faturas estiverem em aberto, imprimir a mensagem

CLIENTE COM TRÊS FATURAS EM ABERTO

e rejeitar o pedido. Pelo menos um "código de ocupação" = 0 permite a aceitação do pedido;

e) verificar o fechamento dos totais de controle do lote (soma dos números das ordens de venda e soma das quantidades); o não fechamento de algum deles implica a impressão de todo o lote e sua rejeição;

f) o relatório de crítica só deve apresentar o cartão tipo 1 quando o erro envolve a rejeição do lote. Neste caso, todos os cartões do lote devem ser impressos. Todavia, se os erros afetarem apenas alguns cartões de um lote, somente estes deverão ser impressos.

7.2 – DESCRIÇÃO DO DESENVOLVIMENTO DO PROGRAMA

Devido à extensão do programa, limitar-nos-emos a descrever aqui apenas os passos principais do desenvolvimento, de modo a evitar que a leitura do texto se torne cansativa.

Antes, porém, de iniciarmos a descrição, vamos apresentar os critérios usados na escolha dos nomes de parágrafos e de variáveis. Critérios para denominar os objetos de um programa contribuem para aumentar sua clareza. Aqui, não estamos prescrevendo nenhuma padronização nesse sentido, mas acreditamos que as normas a seguir poderão ser úteis em instalações que usam COBOL.

7.2.1 – Nomes de Parágrafos

O nome de um parágrafo tenta reproduzir, da maneira mais fiel possível, a idéia expressa no comando abstrato correspondente usado no desenvolvimento, se possível com os mesmos

Tabela 7.1. Crítica de Consistência dos Cartões

Tipo	*Campo*	*Verificação*	*Mensagem de Erro*	*Providência*
Todos	1	deve ser 1 ou 2	CARTAO TIPO INVALIDO	Rejeitar o cartão
1	2–3	deve ser numérico	"NUMERO DO LOTE" NAO NUMERICO	Imprimir e rejeitar todo o lote
	4–9	deve ser numérico	"DATA DO LOTE" NAO NUMERICO	
	4–5	$01 \leqslant$ dia $\leqslant 31$	"DIA" INVALIDO	
	6–7	$01 \leqslant$ mês $\leqslant 12$	"MES" INVALIDO	
	8–9	ano $\geqslant 75$	"ANO" INVALIDO	
	10–11	deve ser numérico	"QUANTIDADE DE ORDENS" NAO NUMERICO	
		$01 \leqslant$ qt. ordens $\leqslant 20$	"QUANTIDADE DE ORDENS" FORA DOS LIMITES	
	12–18	deve ser numérico	"SOMA NUM. ORDENS" NAO NUMERICO	
	19–23	deve ser numérico	"SOMA QUANTIDADES" NAO NUMERICO	
	24–80	deve ser branco	"CAMPO 24–80" NAO BRANCO	
2	2–8	deve ser numérico	"IDENTIFICACAO DO CLIENTE" NAO NUMERICO	Imprimir o cartão e rejeitá-lo
	8	dígito de controle mod. 11 ref. campo 2–7	"IDENTIFICACAO DO CLIENTE" INVALIDO	
	9–13	deve ser numérico	"NUMERO DA ORDEM DE VENDA" NAO NUMERICO	
	14–19	deve ser numérico	"DATA DA COMPRA" NAO NUMERICO	
	14–15	$01 \leqslant$ dia $\leqslant 31$	"DIA" INVALIDO	
	16–17	$01 \leqslant$ mês $\leqslant 12$	"MES" INVALIDO	
	18–19	ano $\geqslant 75$	"ANO" INVALIDO	
	20–31	deve ser numérico	"DESCRICAO DO PRODUTO" NAO NUMERICO	
	28	dígito de controle, idem ao 8, ref. campo 20–27	"PRODUTO INVALIDO"	
	40, 52, 64, 76	idem ao 28, ref. últimas 7 colunas se dif. de branco		

Observações. O dígito de controle é o resto da divisão

$$\frac{\text{algarismo das unidades} + \text{algarismo das dezenas} \cdot 2 + \text{algarismo das centenas} \cdot 3 + \ldots}{11}$$

onde os algarismos do numerador pertencem ao campo ao qual o dígito de controle se refere. Não existe, neste sistema, código com dígito de controle = 10.

termos. Além disso, introduzimos um prefixo dando as iniciais das palavras principais do nome do parágrafo, visando a facilitar a atribuição de nomes a variáveis consideradas *locais*,* além de beneficiar a leitura do texto-fonte. Por exemplo, ao parágrafo COBOL correspondente ao comando abstrato

"analisar cartão tipo 1"

podemos atribuir o nome

AC1–ANALISAR–CARTAO–1.

7.2.2 – Nomes de variáveis

O critério adotado para a escolha de nomes de variáveis visou a:

a) proporcionar uma disciplina para a atribuição de nomes objetivando evitar repetições;
b) facilitar a diferenciação das variáveis que devem ser consideradas como *locais* das globais;
c) facilitar a identificação da função de cada variável pelo seu nome;
d) identificar facilmente a seção da DATA DIVISION onde a variável foi definida.

As regras de formação adotadas foram as seguintes:

a) o nome de uma variável é composto por um prefixo, um mnemônico e, em alguns casos, um sufixo; foram usados os seguintes prefixos:

F – para indicar que a variável foi definida na FILE SECTION;
G – para indicar uma variável global; e
"iniciais das palavras do nome do parágrafo" – para indicar que a variável é considerada *local* àquele parágrafo;

b) as variáveis definidas em nível 01 e 77 não possuem sufixo;
c) as variáveis definidas em outro nível que não o 01 e o 77 devem ter um sufixo que indique a que grupo elas pertencem.

Exemplos

a) G–HALOTE
b) AC1–IND
c) F–CGC–CLI
d) F–PEDIDO.

O exemplo (*a*) mostra uma variável *global* definida na Working Storage Section, cujo mnemônico indica a existência de lotes na fase de leitura. Em (*b*), uma variável local ao parágrafo AC1–ANALISAR–CARTAO–1 é usada como índice. A variável do item (*c*) é definida

---

* Variáveis *locais* são as usadas apenas numa determinada rotina.

na FILE SECTION, em nível diferente de 01, e contém o número do CGC do cliente. Finalmente, o exemplo (*d*) mostra uma variável definida também na FILE SECTION, porém em nível 01, e é o nome do registro do arquivo de pedidos.

7.2.3 – As Primeiras Providências

Para usarmos o computador o mais cedo possível para teste do programa, nossa primeira providência deve ser a escrita dos comandos de controle (JCL).*

Das quatro divisões do programa COBOL, duas podem ser codificadas completamente logo no início do desenvolvimento: a IDENTIFICATION DIVISION, que requer a descrição de alguns atributos dos arquivos (INPUT-OUTPUT SECTION), o que é perfeitamente possível a partir da especificação do programa apresentada na Seç. 7.1,** na DATA DIVISION, a FILE SECTION também pode ser inteiramente codificada a partir da especificação do programa.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.          CRITICA.
  AUTHOR.              ANIBAL PEREIRA.CARDOSO.
  INSTALLATION.        RIO DATACENTRO, PUC - RJ.
  DATE-WRITTEN.        INICIADO EM 08/10/75.
  DATE-COMPILED.
  REMARKS.             ESTE PROGRAMA FAZ PARTE DO SUBSISTEMA DE
                       PROCESSAMENTO DE ORDENS DE VENDA, DO
                       SISTEMA DE ADMINISTRACAO DE COMPRA E VENDA.
                       SUA FUNCAO E' CRITICAR OS LOTES DE ORDENS
                       DE VENDA.

ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
    SOURCE-COMPUTER.   IBM-370.
    OBJECT-COMPUTER.   IBM-370.
    SPECIAL-NAMES.
      C01 IS PAGINA
      DECIMAL-POINT IS COMMA.
  INPUT-OUTPUT SECTION.
    FILE CONTROL.
      SELECT CARTAO ASSIGN TO UR-S-SYSIN.
      SELECT RELAT ASSIGN TO UR-S-SAIDA.
      SELECT ARQCLI ASSIGN TO DA-I-ARQCLI
                    ACCESS IS RANDOM
                    NOMINAL KEY IS G-NOMINAL
                    RECORD KEY IS F-IDENT-CLI.
```

---

* *Job Control Language*: Linguagem de controle do IBM-370.

** Talvez a necessidade de algumas informações da CONFIGURATION SECTION só seja sentida mais tarde. Elas serão inseridas, então, no momento oportuno.

```
        SELECT ARQPED ASSIGN TO DA-I-ARQPED
                      RECORD KEY IS F-CHAVE-PED.

DATA DIVISION
 FILE SECTION.
    FD  CARTAO LABEL RECORD IS OMITTED.
    01  F-CARTAO.
        03  F-TIPO-CT              PIC X.
        03  F-RESTO-CT             PIC X(79).
    FD  RELAT LABEL RECORD IS OMITTED.
    01  F-LINHA                    PIC X(132).
    FD  ARQCLI LABEL RECORDS ARE STANDARD
                      DATA RECORD IS F-CLIENTE.
    01  F-CLIENTE.
        03  F-EXCL-CLI             PIC X.
        03  F-IDENT-CLI.
            05  F-MAT-CLI          PIC 9(6).
            05  F-DIGCTR-CLI       PIC 9.
        03  F-NOME-CLI             PIC X(30).
        03  F-END-CLI              PIC X(50).
        03  F-CGC-CLI              PIC X(12).
        03  F-LIM-CLI              PIC 9(6)V99.
        03  F-COND-CLI             PIC 9.
        03  F-CONCOR-CLI           OCCURS 3 TIMES.
            05  F-OCUP-CLI         PIC 9.
            05  F-NFAT-CLI         PIC X(7).
            05  F-VALOR-CLI        PIC 9(6)V99.
            05  F-LANCA-CLI.
                07  F-DIA-CLI      PIC 99.
                07  F-MES-CLI      PIC 99.
                07  F-ANO-CLI      PIC 99.
    FD  ARQPED LABEL RECORDS ARE STANDARD
                      DATA RECORD IS F-PEDIDO.
    01  F-PEDIDO.
        03  F-EXCL-PED             PIC X.
        03  F-CHAVE-PED            PIC 9(4).
        03  F-NUMOV-PED            PIC 9(5).
        03  F-IDENTCLI-PED         PIC 9(7).
        03  F-DATA-PED             PIC 9(6).
        03  F-DESCRI-PED           OCCURS 5 TIMES.
            05  F-PRODUTO-PED      PIC 9(9).
            05  F-QUANT-PED        PIC 999.
```

A partir de agora, vamos concentrar nossa discussão no desenvolvimento da PROCEDURE DIVISION, considerando implícito que, à medida que novas variáveis sejam necessárias, elas devem ser imediatamente definidas na DATA DIVISION.

7.2.4 – Estratégia de Programação

Antes de escrevermos uma primeira versão do programa, por mais genérica que seja, devemos escolher uma estratégia de programação para possibilitar os primeiros passos. A que decidimos usar é apresentada a seguir.

Para cada lote de ordens de venda, o programa deve ler os cartões correspondentes ao lote e transferi-los para áreas de trabalho na memória – uma área para o cartão tipo 1 e uma para até 40 cartões do tipo 2 – onde serão analisados e criticados. Os erros verificados durante esta análise serão anotados em tabelas correspondentes. Pelo exame destas tabelas, será processada a saída do lote: cartões certos serão gravados no arquivo de pedidos e cartões errados serão impressos no relatório de crítica.

7.2.5 – O Desenvolvimento da "Procedure Division"

O primeiro passo do desenvolvimento já pode ser dado, por meio da escrita do módulo de controle principal do programa. Ele terá, basicamente, três fases: inicialização, processamento e finalização (Fig. 7.4).

```
PASSO 1.
        Crítica:
            "inicializar"
            "processar"
            "finalizar"
```

Fig. 7.4

Na inicialização, fazemos a abertura dos arquivos, a inicialização de algumas variáveis e a impressão do cabeçalho do relatório de crítica; na finalização, fechamos os arquivos e encerramos o programa; o processamento se constitui na parte principal, onde devemos obter os lotes e processá-los.

A entrada dos dados é um ponto importante na lógica deste programa. Um detalhe relevante é que, na leitura, o final de um lote é determinado pela ocorrência do cartão tipo 1 que é *cabeça* do lote seguinte, ou pelo fim de arquivo. A Fig. 7.5 apresenta a disposição dos dados de entrada, destacando a forma como será feita a leitura dos lotes. Podemos observar que a obtenção do primeiro cartão tipo 1 terá de ser feita de forma diferente. Decidimos por sua leitura ainda na fase de inicialização.

O passo 2 (Fig. 7.6) é um refinamento do passo 1.

Expressões como "há lote" e "não há lote" serão usadas em nosso vocabulário para indicar os dois possíveis estados, FALSO E VERDADEIRO, de variáveis de controle (*flags*). Por exemplo, a expressão "há lote" poderá ser interpretada no programa COBOL por G–HALOTE IS EQUAL TO 1; sua negação, "não há lote", por G–HALOTE IS EQUAL TO 0.

A expansão de "ler e processar lote" pode parecer imediata, uma vez que todo lote lido deve ser processado. Todavia, existe um caso em que o lote deve ser totalmente rejeitado na fase

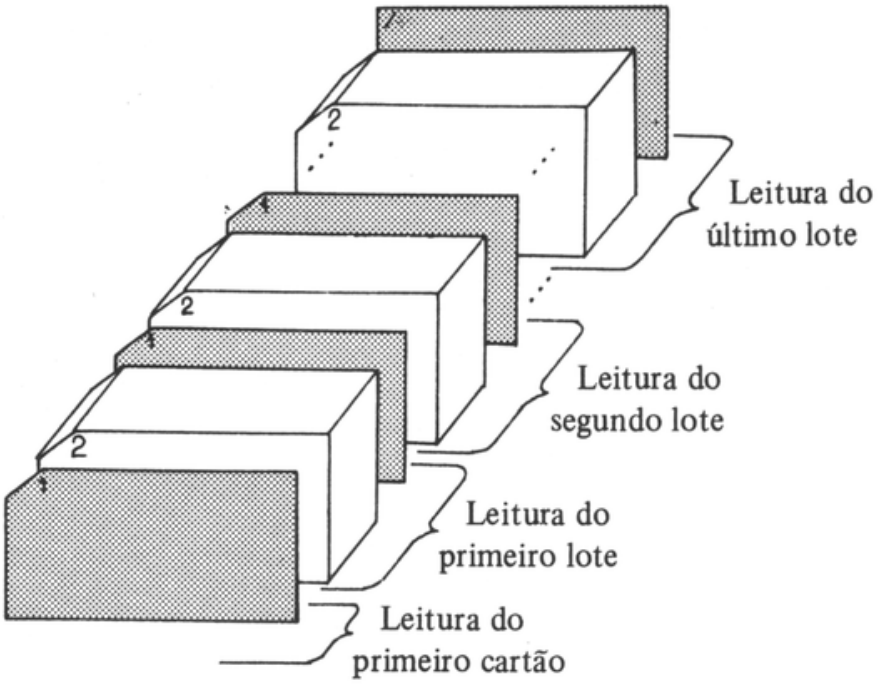


Fig. 7.5

PASSO 2. Refinamento do passo 1.

```
inicializar:
    "abrir arquivos"
    "inicializar variáveis"
    "imprimir cabeçalho"
    "ler o primeiro cartão"
processar:
    enquanto "há lote"
        "ler e processar lote"
    fim-enquanto
finalizar:
    "fechar arquivos"
```

Fig. 7.6

de leitura e não pode ser processado: é quando um lote tem mais de 40 cartões do tipo 2. Isto pode ocorrer pelo esquecimento de um cartão do tipo 1 ou a sua má perfuração. O refinamento de "ler e processar lote" é, então, apresentado na Fig. 7.7.

A árvore do programa (Fig. 7.8), retrata o desenvolvimento do programa até o passo 3. Não aparecem na árvore os nós correspondentes a "abrir arquivos", "inicializar variáveis" e "fechar arquivos", por julgarmos suas expansões bastante imediatas e comuns nas fases de inicialização e finalização de quase todos os programas.

A seguir, vamos refinar "ler o primeiro cartão".

Embora, à primeira vista, possa parecer que este refinamento se constitui num único comando de leitura, devemos prever o caso em que o primeiro cartão da massa não seja do

PASSO 3. Refinamento de "ler e processar lote".

```
"ler lote"
se "lote não rejeitado"
    então
        "processar lote"
fim-se
```

Fig. 7.7

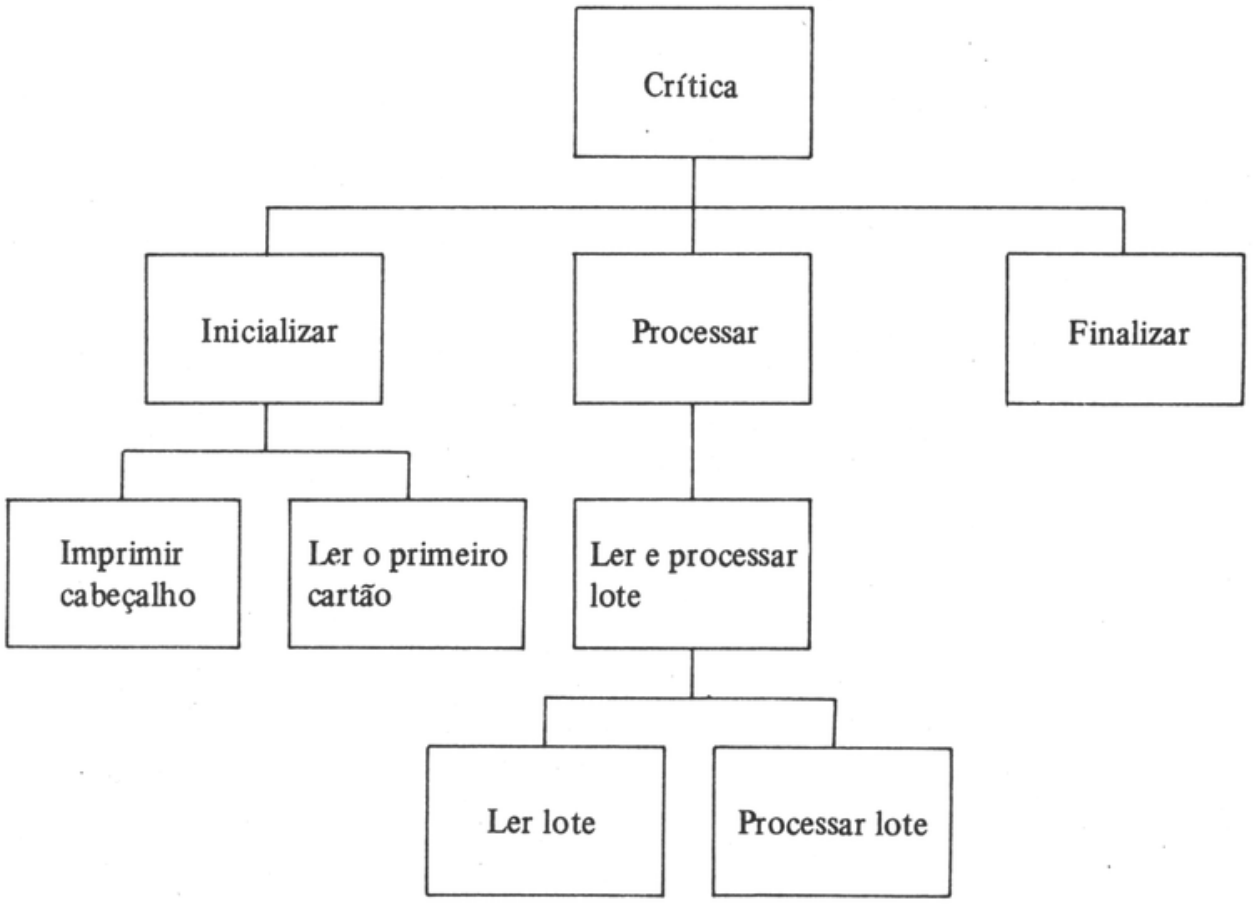


Fig. 7.8

tipo 1. Se tal ocorrer, devemos ler e rejeitar todos os cartões até que apareça um de tipo 1. Se aparecer o fim de arquivo antes de algum cartão tipo 1, a massa não pode ser processada e o programa deve terminar. Isto é programado na Fig. 7.9.

O passo seguinte do desenvolvimento é o refinamento de "ler lote" (Fig. 7.10). "Ponteiro" é uma variável que indica uma posição da tabela "área de trabalho dos cartões tipo 2". Quando o controle é passado para "ler lote", um cartão tipo 1 estará residindo na área de

PASSO 4. Refinamento de "ler o primeiro cartão".

```
enquanto "há cartão e tipo não é 1"
   "ler cartão, ao fim de arquivo,
      não há cartão, não há lote"
   se "há cartão"
      então
         se "tipo não é 1"
            então
               "rejeitar o cartão lido"
         fim-se
   fim-se
fim-enquanto
```

Fig. 7.9

PASSO 5. Refinamento de "ler lote".

```
"zerar ponteiro e tabelas de erros"
"mover cartão lido para área de trabalho do cartão
   tipo 1"
enquanto "não é fim deste lote"
   "ler cartão, ao fim de arquivo,
      não há lote, fim deste lote, ir para fim desta rotina"
   se "tipo é não numérico"
      então
         "mover 3 para tipo"
   fim-se
   para
      cartão tipo 1:  "fim deste lote"
      cartão tipo 2:  se "ponteiro ⩾ máximo de cartões tipo 2"
                         então
                            "rejeitar todos cartões até o fim
                               deste lote"
                            "lote rejeitado, fim deste lote"
                         se não
                            "adiantar ponteiro"
                            "mover cartão lido para área de
                               trabalho dos cartões tipo 2"
                      fim-se
      outro tipo:  "rejeitar cartão tipo inválido"
   fim-para
fim-enquanto
```

Fig. 7.10

leitura. Foi ele que determinou o fim da leitura do lote anterior. Este cartão deve, então, ser movido para a área de trabalho, o que é providenciado por

"mover cartão lido para área de trabalho do cartão tipo 1".

A leitura dos cartões é feita dentro de uma estrutura DO–WHILE. A cada repetição, enquanto não é encontrado o fim do lote (cartão 1 ou fim de arquivo), um cartão é lido e uma estrutura de seleção CASE escolhe entre três alternativas possíveis, de acordo com o tipo do cartão lido. No caso de cartão tipo 2, um teste é feito quanto ao número de cartões do lote. Se ultrapassar 40, o lote todo deve ser rejeitado.

Neste ponto do desenvolvimento, vamos testar o programa em máquina. O teste consiste em ler uma massa de cartões prevendo as possibilidades analisadas no desenvolvimento (cartões invalidos, primeiro cartão com tipo diferente de 1, lote com mais de 40 cartões do tipo 2).

Para o teste, devemos traduzir os diversos passos dados para COBOL e introduzir, como "rotinas fantasmas", aqueles parágrafos que são chamados por esta versão e que serão refinados mais tarde:

```
C8545 V2 LVL78 01MAY72                    IBM OS AMERICAN NATIONAL STANDARD COBOL

  1

00001          IDENTIFICATION DIVISION.

00002           PROGRAM-ID.              CRITICA.
00003           AUTHOR.                  ANIBAL PEREIRA CARDOSO.
00004           INSTALLATION.            RIO DATACENTRO, PUC - RJ.
00005           DATE-WRITTEN.            INICIADO EM 08/10/75.
00006          DATE-COMPILED. NOV 18,1975
00007           REMARKS.                 ESTE PROGRAMA FAZ PARTE DO SUBSISTEMA DE
00008                                    PROCESSAMENTO DE ORDENS DE VENDA, DO
00009                                    SISTEMA DE ADMINISTRACAO DE COMPRA E VENDA.
00010                                    SUA FUNCAO E' CRITICAR OS LOTES DE ORDENS
00011                                    DE VENDA.
```

```
00012           ENVIRONMENT DIVISION.

00013            CONFIGURATION SECTION.
00014             SOURCE-COMPUTER.     IBM-370.
00015             OBJECT-COMPUTER.     IBM-370.
00016             SPECIAL-NAMES.
00017               C01 IS PAGINA
00018               DECIMAL-POINT IS COMMA.

00019            INPUT-OUTPUT SECTION.
00020             FILE-CONTROL.
00021               SELECT CARTAO ASSIGN TO UR-S-SYSIN.
00022               SELECT RELAT ASSIGN TO UR-S-SAIDA.
00023               SELECT ARQCLI  ASSIGN TO DA-I-ARQCLI
00024                              ACCESS IS RANDOM
00025                              NOMINAL KEY IS G-NOMINAL
00026                              RECORD KEY IS F-IDENT-CLI.
00027               SELECT ARQPED ASSIGN TO DA-I-ARQPED
00028                              RECORD KEY IS F-CHAVE-PED.
```

3

```
00029          DATA DIVISION.

00030           FILE SECTION.
00031            FD CARTAO LABEL RECORD IS OMITTED.
00032            01 F-CARTAO.
00033               03 F-TIPO-CT              PIC X.
00034               03 F-RESTO-CT             PIC X(79).

00035            FD RELAT LABEL RECORD IS OMITTED.
00036            01 F-LINHA              PIC X(132).

00037            FD ARQCLI LABEL RECORDS ARE STANDARD
00038                     DATA RECORD IS F-CLIENTE.
00039            01 F-CLIENTE.
C0040               03 F-EXCL-CLI             PIC X.
00041               03 F-IDENT-CLI.
00042                  05 F-MAT-CLI           PIC 9(6).
00043                  05 F-DIGCTR-CLI        PIC 9.
00044               03 F-NOME-CLI             PIC X(30).
00045               03 F-END-CLI              PIC X(50).
00046               03 F-CGC-CLI              PIC 9(12).
C0047               03 F-LIM-CLI              PIC 9(6)V99.
00048               03 F-COND-CLI             PIC 9.
00049               03 F-CONCOR-CLI           OCCURS 3 TIMES.
00050                  05 F-OCUP-CLI          PIC 9.
00051                  05 F-NFAT-CLI          PIC X(7).
00052                  05 F-VALOR-CLI         PIC 9(6)V99.
00053                  05 F-LANCA-CLI.
00054                     07 F-DIA-CLI        PIC 99.
00055                     07 F-MES-CLI        PIC 99.
C0056                     07 F-ANO-CLI        PIC 99.

C0057            FD ARQPED LABEL RECORDS ARE STANDARD
00058                     DATA RECORD IS F-PEDIDO.
C0059            01 F-PEDIDO.
00060               03 F-EXCL-PED             PIC X.
00061               03 F-CHAVE-PED            PIC 9(4).
00062               03 F-NUMOV-PED            PIC 9(5).
00063               03 F-IDENTCLI-PED         PIC 9(7).
00064               03 F-DATA-PED             PIC 9(6).
00065               03 F-DESCRI-PED           OCCURS 5 TIMES.
00066                  05 F-PRODUTO-PED       PIC 9(9).
00067                  05 F-QUANT-PED         PIC 999.
```

```
00068          WORKING-STORAGE SECTION.
00069           77 G-HALOTE                  PIC 9 COMP.
00070           77 G-LOTEREJ                 PIC 9 COMP.
00071           77 G-NOMINAL                 PIC 9(7).
00072           77 LL-FIMLOTE                PIC 9 COMP.
00073           77 LL-PONTEIRO               PIC 99.
00074           77 LPC-HACARTAO              PIC 9 COMP.
00075           77 UL-TIPO                   PIC 9.

00076      ******************************************************************
00077      *                                                                *
00078      *              AREA DE TRABALHO PARA CARTAO TIPO 1               *
00079      *                                                                *
00080      ******************************************************************

00081          01 G-CARTAO1.
00082             03 G-TIPO-CT1             PIC X.
00083             03 G-NUMLOTE-CT1          PIC XX.
00084             03 G-DATALOTE-CT1.
00085                05 G-DIA-CT1           PIC XX.
00086                05 G-MES-CT1           PIC XX.
00087                05 G-ANO-CT1           PIC XX.
00088             03 G-QTORD-CT1            PIC XX.
00089             03 G-SOMANUM-CT1          PIC X(7).
00090             03 G-SOMAQT-CT1           PIC X(5).
00091             03 G-RESTO-CT1            PIC X(57).

00092      ******************************************************************
00093      *                                                                *
00094      *              AREA DE TRABALHO PARA CARTOES TIPO 2              *
00095      *                                                                *
00096      ******************************************************************

00097          01 G-TABCART2.
00098             03 G-CARTAO2              OCCURS 40 TIMES.
00099                05 G-TIPO-CT2          PIC X.
00100                05 G-IDENTCLI-CT2.
00101                   07 G-MAT-CT2        PIC X(6).
00102                   07 G-DIGCTR-CT2     PIC X.
00103                05 G-NUMOV-CT2         PIC X(5).
00104                05 G-DATA-CT2.
00105                   07 G-DIA-CT2        PIC XX.
00106                   07 G-MES-CT2        PIC XX.
00107                   07 G-ANO-CT2        PIC XX.
00108                05 G-DESCRI-CT2        OCCURS 5 TIMES.
00109                   07 G-PROD-CT2.
00110                      09 G-COD-CT2     PIC X(8).
00111                      09 G-DIGCOD-CT2  PIC X.
00112                   07 G-QUANT-CT2      PIC XXX.
00113                05 G-RESTO-CT2         PIC X.
```

```
00114      ******************************************************************
00115      *                                                                *
00116      *            AREA DE REJEICAO DE CARTOES                         *
00117      *                                                                *
00118      ******************************************************************

00119          01 G-AREAREJ.
00120             03 FILLER                PIC X(10) VALUE SPACES.
00121             03 G-CONTEUDO-AR         PIC X(80).
00122             03 G-ERRO-AR             PIC X(31).
00123             03 FILLER                PIC X(25) VALUE SPACES.

00124      ******************************************************************
00125      *                                                                *
00126      *            TABELA PARA ERROS DO CARTAO TIPO 1                  *
00127      *                                                                *
00128      ******************************************************************

00129          01 G-TABCAR1.
00130             03 G-LINHA-TB1           OCCURS 12 TIMES PIC 99.

00131      ******************************************************************
00132      *                                                                *
00133      *            TABELA PARA ERROS DOS CARTOES TIPO 2: VARIAVEL      *
00134      *                                                                *
00135      ******************************************************************

00136          01 G-TABVARCAR2.
00137             03 G-LIN-VAR             OCCURS 40 TIMES.
00138                05 G-COL-VAR          OCCURS 13 TIMES PIC 99.

00139      ******************************************************************
00140      *                                                                *
00141      *            TABELA PARA ERROS DOS CARTOES TIPO 2: FIXA          *
00142      *                                                                *
00143      ******************************************************************

00144          01 G-TABFIXCAR2.
00145             03 G-LIN-FIX             OCCURS 40 TIMES.
00146                05 G-COL-FIX          OCCURS 5 TIMES PIC 99.
```

```
00147          ************************************************************
00148          *                          NIVEL -1                        *
00149          ************************************************************

00150           PROCEDURE DIVISION.

00151           INICIALIZAR SECTION.
00152               OPEN INPUT CARTAO.
00153               OPEN OUTPUT RELAT.
00154               OPEN INPUT ARQCLI.
00155               OPEN OUTPUT ARQPED.
00156               PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00157               PERFORM LPC-LER-PRIMEIRO-CARTAO THRU PRIMEIRO-CARTAO-LIDO.

00158           PROCESSAR SECTION.
00159          *    ENQUANTO G-HALOTE IS = 1, LER E PROCESSAR LOTES.
00160                 PERFORM LPL-LER-PROCESSAR-LOTES THRU
00161                   LOTES-LIDOS-PROCESSADOS
00162                 UNTIL G-HALOTE IS NOT = 1.

00163           FINALIZAR SECTION.
00164               CLOSE CARTAO.
00165               CLOSE RELAT.
00166               CLOSE ARQCLI.
00167               CLOSE ARQPED.
00168               STOP RUN.


00169           LPL-LER-PROCESSAR-LOTES.
00170               MOVE 0 TO G-LOTEREJ.
00171               PERFORM LL-LER-LOTE THRU LOTE-LIDO.
00172               IF G-LOTEREJ IS = 0
00173                  THEN
00174                    PERFORM PL-PROCESSAR-LOTE THRU LOTE-PROCESSADO.
00175           LOTES-LIDOS-PROCESSADOS.
```

```
 7
00176          ******************************************************************
00177          *                          NIVEL -2                              *
00178          ******************************************************************

00179           LPC-LER-PRIMEIRO-CARTAO.
00180               MOVE 1 TO LPC-HACARTAO.
00181               MOVE 1 TO G-HALOTE.
00182               MOVE 0 TO F-TIPO-CT.
00183          *    _ENQUANTO LPC-HACARTAO = 1 AND F-TIPO-CT IS NOT = 1, OBTER
00184          *                                                       CARTAO.
00185                 PERFORM OC-OBTER-CARTAO THRU CARTAO-OBTIDO
00186                 UNTIL LPC-HACARTAO IS NOT =  1 OR F-TIPO-CT IS = 1.
00187           PRIMEIRO-CARTAO-LIDO.


00188           OC-OBTER-CARTAO.
00189               READ CARTAO
00190                  AT END MOVE 0 TO LPC-HACARTAO.
00191                         MOVE 0 TO G-HALOTE.
00192               IF LPC-HACARTAO IS = 1
00193                  THEN
00194          *-----------------PROVISORIO-------------------------------------
00195               DISPLAY 'TIPO = ' F-TIPO-CT
00196          *----------------------------------------------------------------
00197                    IF F-TIPO-CT IS NOT = 1
00198                      THEN
00199                        MOVE 'PRIMEIRO CARTAO DEVE SER TIPO 1' TO G-ERRO-AR
00200                        PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
00201           CARTAO-OBTIDO.
```

```
00202          LL-LER-LOTE.
00203              MOVE ALL SPACES TO G-TABCART2.
00204              MOVE ALL '00' TO G-TABCAR1, G-TABVARCAR2, G-TABFIXCAR2.
00205              MOVE 0 TO LL-PONTEIRO.
00206              MOVE 0 TO LL-FIMLOTE.
00207              MOVE F-CARTAO TO G-CARTAO1.
00208          *    ENQUANTO LL-FIMLOTE IS = 0, OBTER LOTE.
00209                 PERFORM OL-OBTER-LOTE THRU LOTE-OBTIDO
00210                 UNTIL LL-FIMLOTE IS NOT = 0.
00211          LOTE-LIDO.


00212          OL-OBTER-LOTE.
00213              READ CARTAO
00214                AT END MOVE 0 TO G-HALOTE,
00215                       MOVE 1 TO LL-FIMLOTE,
00216                       GO TO LOTE-OBTIDO.
00217          *-----------------PROVISORIO-----------------------------------
00218              DISPLAY 'TIPO = ' F-TIPO-CT
00219          *--------------------------------------------------------------
00220              IF F-TIPO-CT IS NOT NUMERIC
00221                THEN
00222                  MOVE 3 TO F-TIPO-CT.
00223              MOVE F-TIPO-CT TO OL-TIPO.
00224              GO TO CT1-CARTAO-TIPO-1, CT2-CARTAO-TIPO-2,
00225                DEPENDING ON OL-TIPO.

00226            OT-OUTRO-TIPO.
00227                  MOVE 'CARTAO TIPO INVALIDO                TO G-ERRO-AR.
00228                  PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
00229                  GO TO LOTE-OBTIDO.

00230            CT1-CARTAO-TIPO-1.
00231                  MOVE 1 TO LL-FIMLOTE.
00232                  GO TO LOTE-OBTIDO.

00233            CT2-CARTAO-TIPO-2.
00234                  IF LL-PONTEIRO IS > 40 OR LL-PONTEIRO IS = 40
00235                    THEN
00236                      PERFORM RFL-REJEITAR-ATE-FIM-LOTE THRU
00237                        ATE-FIM-LOTE-REJEITADOS
00238                      MOVE 1 TO G-LOTEREJ, LL-FIMLOTE
00239                    ELSE
00240                      ADD 1 TO LL-PONTEIRO
00241                      MOVE F-CARTAO TO G-CARTAO2 (LL-PONTEIRO).
00242          LOTE-OBTIDO. EXIT.
```

9

```
00243          IC-IMPRIMIR-CABECALHO.
00244         *-----------------PROVISORIO-----------------------------------
00245              DISPLAY 'IC CHAMADO'.
00246         *-------------------------------------------------------------
00247          CABECALHO-IMPRESSO.

00248          RC-REJEITAR-CARTAO.
00249         *-----------------PROVISORIO-----------------------------------
00250              DISPLAY 'RC CHAMADO'.
00251         *-------------------------------------------------------------
00252          CARTAO-REJEITADO.

00253          RFL-REJEITAR-ATE-FIM-LOTE.
00254         *-----------------PROVISORIO-----------------------------------
00255              DISPLAY 'RFL CHAMADO'.
00256         *-------------------------------------------------------------
00257          ATE-FIM-LOTE-REJEITADOS.

00258          PL-PROCESSAR-LOTE.
00259         *-----------------PROVISORIO-----------------------------------
00260              DISPLAY 'PL CHAMADO'.
00261         *-------------------------------------------------------------
00262          LOTE-PROCESSADO.
```

```
TIPO = 2
RC CHAMADO
TIPO = 2
RC CHAMADO
TIPO = 2
RC CHAMADO
TIPO = 1
TIPO = 2
TIPO = 3
RC CHAMADO
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
```

```
RFL CHAMADO
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 2
TIPO = 1
FL CHAMADO
TIPO = A
RC CHAMADO
```

```
TIPO = 2
TIPO = 2
PL CHAMADO
```

A Fig. 7.11 mostra a árvore do primeiro teste, destacando as "rotinas fantasmas". A rotina "RC–REJEITAR–CARTÃO" é tratada de forma semelhante a uma sub-rotina, pois é chamada em dois pontos do programa. O refinamento dessas rotinas é por demais simples e sua discussão nos parece dispensável neste texto.

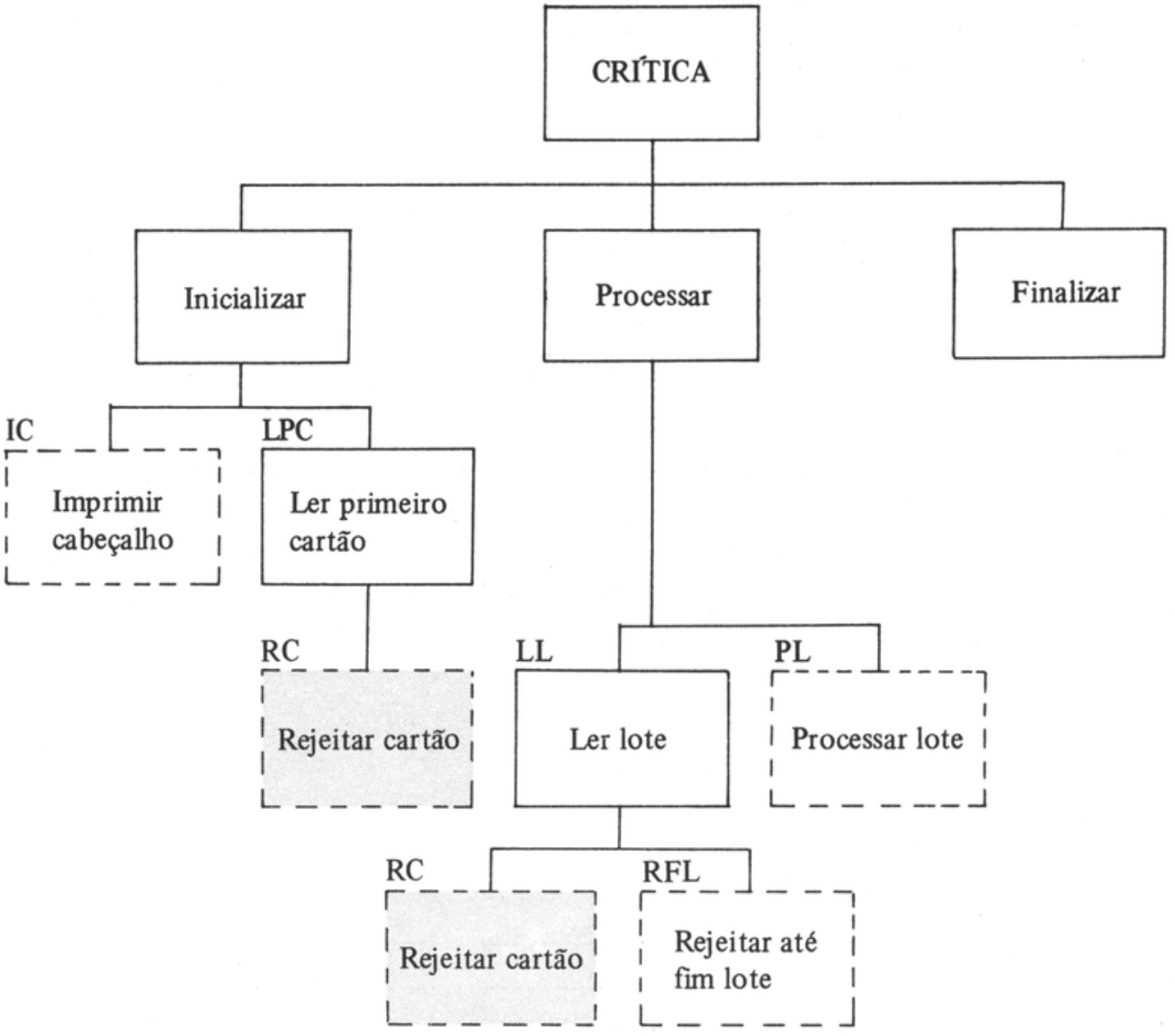


Fig. 7.11

O teste efetuado pode ser considerado como um marco importante no desenvolvimento. Com ele terminamos a programação da primeira parte, que trata da obtenção dos dados. A segunda parte, que desenvolveremos agora, trata do processamento propriamente dito dos lotes.

A Fig. 7.12 contém o passo 6, que é muito importante. Ele resume o tratamento que deve ser dado a cada lote obtido. Inicialmente, fazemos uma análise dos vários campos do cartão tipo 1, quanto aos detalhes apresentados na Tab. 7.1 da especificação do programa (seção anterior). Se alguma inconsistência for verificada, o lote inteiro deve ser rejeitado; caso contrário, o mesmo tipo de análise deve ser feito para os cartões tipo 2. Após, uma conferência dos totais de controle se constitui numa nova oportunidade de rejeitar todo o lote. "Criticar

```
PASSO 6.  Refinamento de "processar lote".

          "analisar cartão tipo 1"
          se "lote não rejeitado"
             então
                "analisar cartões tipo 2"
                "analisar totais de controle"
          fim-se
          se "lote não rejeitado"
             então
                "criticar clientes"
          fim-se
          "processar saídas"
```

Fig. 7.12

clientes" significa analisar cada cartão tipo 2 do lote quanto às condições de crédito, conta corrente e cadastramento do cliente, consultando o arquivo correspondente. Esta crítica só deve ser feita sobre os cartões que não apresentaram nenhuma inconsistência na fase de análise anterior. Finalmente, em "processar saídas" temos a gravação, no arquivo de pedidos aceitos (ARQPED), do conteúdo dos cartões tipo 2 certos, e a impressão, no relatório de crítica, dos cartões errados. A Fig. 7.13 apresenta a árvore do programa para o passo 6.

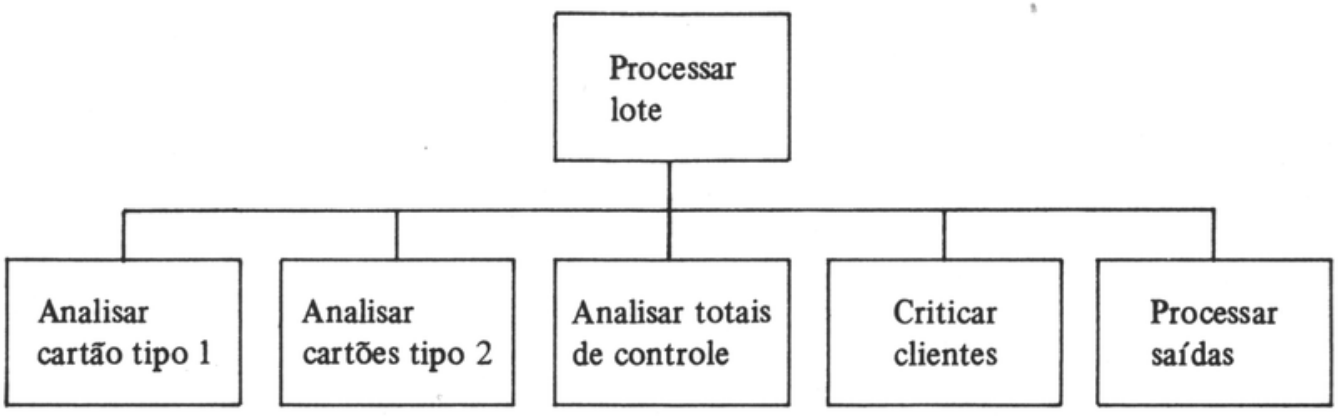


Fig. 7.13

Considerando que discutimos as principais decisões lógicas do desenvolvimento e que demos uma ampla visão ao leitor da política adotada na resolução do problema, encerramos aqui a descrição do desenvolvimento, mostrando a árvore completa do programa (Fig. 7.14). Podemos observar que algumas funções se repetem na árvore. Tais rotinas são codificadas uma única vez e localizadas no nível mais baixo em que elas aparecem (ver Cap. 6).

A listagem completa deste programa, inclusive com o resultado de um teste, é apresentada no Apêndice B.

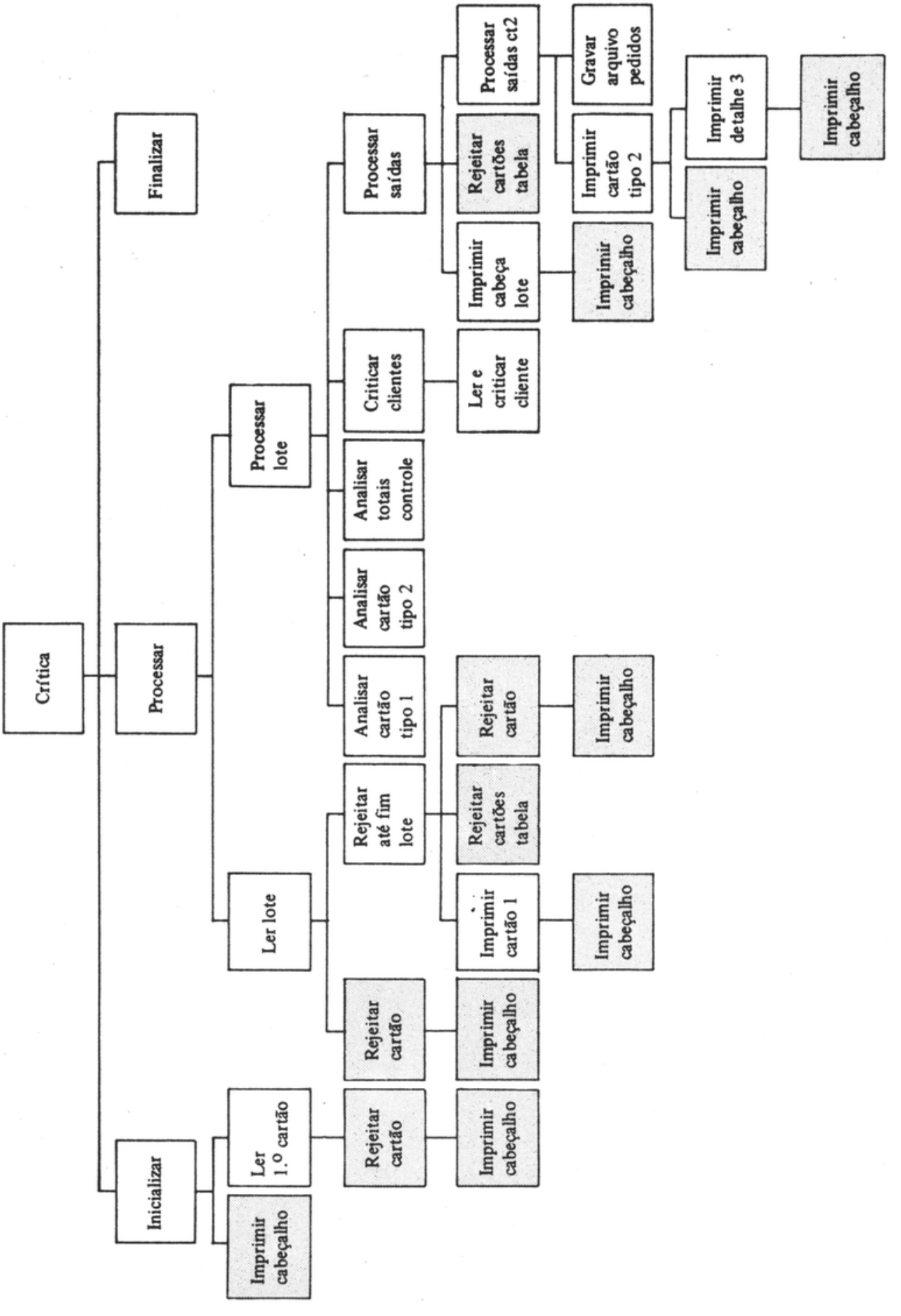

Crítica
Inicializar
Imprimir cabeçalho
Ler 1.º cartão
Rejeitar cartão
Imprimir cabeçalho
Processar
Ler lote
Rejeitar cartão
Imprimir cabeçalho
Rejeitar até fim lote
Rejeitar cartões tabela
Imprimir cartão 1
Imprimir cabeçalho
Processar lote
Analisar cartão tipo 1
Rejeitar cartão
Imprimir cabeçalho
Analisar cartão tipo 2
Analisar totais controle
Criticar clientes
Ler e criticar cliente
Processar saídas
Rejeitar cartões tabela
Imprimir cabeça lote
Imprimir cabeçalho
Processar saídas ct2
Gravar arquivo pedidos
Imprimir cartão tipo 2
Imprimir cabeçalho
Imprimir detalhe 3
Imprimir cabeçalho
Finalizar

Fig. 7.14

Apêndice A

A Notação dos Exemplos de Programas

Este apêndice descreve informal e resumidamente a notação utilizada nos exemplos de programas estruturados da Parte I.

A.1 – SÍMBOLOS BÁSICOS

Letras	$A \ldots Z$, $a \ldots z$
Dígitos	0 1 2 3 4 5 6 7 8 9
Operadores aritméticos	+ (adição)
	– (subtração)
	* (multiplicação)
	/ (divisão)
	** (exponenciação)
Operadores lógicos	$\wedge$ (e, AND)
	$\vee$ (ou, OR)
	$\neg$ (não, NOT)
Operadores relacionais	$= \neq < \leqslant > \geqslant$
Parênteses	( )
Chaves de comentários	{ }
Operador de atribuição	:=
Apóstrofe	'
Separadores	, ; : –
Espaço	␣

A.2 – CONSTANTES

Numéricas	Números reais ou inteiros, com ou sem sinal
Lógicas	*true, false*

A.3 – NOMES

De variáveis simples e de sub-rotinas	Grupo de letras

De variáveis indexadas	Uma variável simples seguida de um índice, entre parênteses, que pode ser composto por uma lista de variáveis e/ou constantes numéricas.
De rótulos (*labels*)	Grupo de letras e/ou dígitos, seguido de dois pontos (:)

A.4 – EXPRESSÕES

Aritméticas e lógicas	Como em COBOL, PL/I, FORTRAN, ALGOL etc., exceto quanto à notação usada para os operadores (ver símbolos básicos)

A.5 – "STRING" DE CARACTERES

Conjunto de símbolos básicos (exceto apóstrofe), entre apóstrofes

A.6 – DECLARAÇÃO DE SUB-ROTINAS

Procedure: nome da sub-rotina

A.7 – COMENTÁRIO

Qualquer grupo de símbolos básicos (exceto chaves), entre chaves

A.8 – COMANDOS

Atribuição	Nome de variável : = expressão;
Transferência incondicional	go to label;
Início de processamento	start;
Fim de processamento	stop; (programa principal)
Saída de sub-rotina	return;
Chamada de sub-rotina	call nome da sub-rotina;
Entrada	read lista de variáveis;
Saída	print lista de expressões;
Seqüência	Conjunto de comandos válidos
Seleção	if expressão lógica then comando else comando fi; if expressão lógica then comando fi; case expressão aritmética lista de *labels* comando lista de *labels* comando . . . . . . lista de *labels* comando esac; onde lista de *labels* é um conjunto de um ou mais *labels*
Repetição	do while expressão lógica; comando; od; repeat comando until expressão lógica;

Apêndice B

Versão Final do Programa "CRÍTICA"

Este apêndice contém o texto-fonte da versão final do programa "CRÍTICA", desenvolvido como exemplo de programa estruturado em COBOL no Cap. 7.

```
CB545 V2 LVL78 01MAY72                IBM OS AMERICAN NATIONAL STANDARD COBOL

  1

00001          IDENTIFICATION DIVISION.

00002           PROGRAM-ID.           CRITICA.
00003           AUTHOR.               ANIBAL PEREIRA CARDOSO.
00004           INSTALLATION.         RIO DATACENTRO, PUC - RJ.
00005           DATE-WRITTEN.         INICIADO EM 08/10/75.
00006          DATE-COMPILED. NOV 27,1975
00007           REMARKS.              ESTE PROGRAMA FAZ PARTE DO SUBSISTEMA DE
00008                                 PROCESSAMENTO DE ORDENS DE VENDA, DO
00009                                 SISTEMA DE ADMINISTRACAO DE COMPRA E VENDA.
00010                                 SUA FUNCAO E  CRITICAR OS LOTES DE ORDENS
00011                                 DE VENDA.
```

2

```
00012          ENVIRONMENT DIVISION.

00013           CONFIGURATION SECTION.
00014            SOURCE-COMPUTER.    IBM-370.
00015            OBJECT-COMPUTER.    IBM-370.
00016            SPECIAL-NAMES.
00017              C01 IS PAGINA
00018              DECIMAL-POINT IS COMMA.

00019           INPUT-OUTPUT SECTION.
00020            FILE-CONTROL.
00021              SELECT CARTAO ASSIGN TO UR-S-SYSIN.
00022              SELECT RELAT ASSIGN TO UR-S-SAIDA.
00023              SELECT ARQCLI ASSIGN TO DA-I-ARQCLI
00024                            ACCESS IS RANDOM
00025                            NOMINAL KEY IS G-NOMINAL
00026                            RECORD KEY IS F-IDENT-CLI.
00027              SELECT ARQPED ASSIGN TO DA-I-ARQPED
00028                            RECORD KEY IS F-CHAVE-PED.
```

3

```
00029          DATA DIVISION.

00030           FILE SECTION.
00031            FD CARTAO LABEL RECORD IS OMITTED.
00032            01 F-CARTAO.
00033               03 F-TIPO-CT          PIC X.
00034               03 F-RESTO-CT         PIC X(79).

00035            FD RELAT LABEL RECORD IS OMITTED.
00036            01 F-LINHA               PIC X(132).

00037            FD ARQCLI LABEL RECORDS ARE STANDARD
00038                      DATA RECORD IS F-CLIENTE.
00039            01 F-CLIENTE.
00040               03 F-EXCL-CLI         PIC X.
00041               03 F-IDENT-CLI.
00042                  05 F-MAT-CLI       PIC 9(6).
00043                  05 F-DIGCTR-CLI    PIC 9.
00044               03 F-NOME-CLI         PIC X(30).
00045               03 F-END-CLI          PIC X(50).
00046               03 F-CGC-CLI          PIC X(12).
00047               03 F-LIM-CLI          PIC 9(6)V99.
00048               03 F-COND-CLI         PIC 9.
00049               03 F-CONCOR-CLI       OCCURS 3 TIMES.
00050                  05 F-OCUP-CLI      PIC 9.
00051                  05 F-NFAT-CLI      PIC X(7).
00052                  05 F-VALOR-CLI     PIC 9(6)V99.
00053                  05 F-LANCA-CLI.
00054                     07 F-DIA-CLI    PIC 99.
00055                     07 F-MES-CLI    PIC 99.
00056                     07 F-ANO-CLI    PIC 99.

00057            FD ARQPED LABEL RECORDS ARE STANDARD
00058                      DATA RECORD IS F-PEDIDO.
00059            01 F-PEDIDO.
00060               03 F-EXCL-PED         PIC X.
00061               03 F-CHAVE-PED        PIC 9(4).
00062               03 F-NUMOV-PED        PIC 9(5).
00063               03 F-IDENTCLI-PED     PIC 9(7).
00064               03 F-DATA-PED         PIC 9(7).
00065               03 F-DESCRI-PED       OCCURS 5 TIMES.
00066                  05 F-PRODUTO-PED   PIC 9(9).
00067                  05 F-QUANT-PED     PIC 999.
```

```
00068           WORKING-STORAGE SECTION.
00069            77 G-HALOTE                PIC 9 COMP.
00070            77 G-LOTEREJ               PIC 9 COMP.
00071            77 G-MAXLOTE               PIC 999 VALUE 40.
00072            77 G-NLINHA                PIC 99.
00073            77 G-NOMINAL               PIC 9(7).
00074            77 AC1-IND                 PIC 99.
00075            77 AC2-COLUNA              PIC 99.
00076            77 AC2-LINHA               PIC 99.
00077            77 AC2-NUMOV               PIC 9(5).
00078            77 AC2-QUANT               PIC 999.
00079            77 AC2-TOTAL1              PIC 9(7).
00080            77 AC2-TOTAL2              PIC 9(5).
00081            77 CC-COLUNA               PIC 99.
00082            77 CC-LINHA                PIC 99.
00083            77 GAP-CHAVE               PIC 9(4) VALUE 0.
00084            77 GAP-IND                 PIC 9.
00085            77 IC2-COL                 PIC 99.
00086            77 IC2-ERRO                PIC 99.
00087            77 IC-NPAG                 PIC 999.
00088            77 IC-SALVA                PIC XX.
00089            77 ICL-ERRO                PIC 99.
00090            77 ICL-INDICE              PIC 99.
00091            77 LL-FIMLOTE              PIC 9 COMP.
00092            77 LL-PONTEIRO             PIC 99.
00093            77 LCC-CHAVE               PIC 9 COMP.
00094            77 LCC-INDICE              PIC 9.
00095            77 LPC-HACARTAO            PIC 9 COMP.
00096            77 OL-TIPO                 PIC 9.
00097            77 PL-LOTEREJ              PIC 9 COMP.
00098            77 PSC-IND                 PIC 99.
00099            77 RCT-LIN                 PIC 99.
00100            77 VC-RESTO1               PIC 99.
00101            77 VC-DIGCTR               PIC 99.
00102            77 VC-DIVIDENDO1           PIC 999.
00103            77 VC-INDEX                PIC 9.
00104            77 VRC-DIGCOD              PIC 99.
00105            77 VRC-DIVIDENDO2          PIC 999.
00106            77 VRC-RESTO2              PIC 99.
```

```
00107          ******************************************************************
00108          *                                                                *
00109          *             AREA DE TRABALHO PARA CARTAO TIPO 1                *
00110          *                                                                *
00111          ******************************************************************

00112             01 G-CARTAO1.
00113                03 G-TIPO-CT1           PIC X.
00114                03 G-NUMLOTE-CT1        PIC XX.
00115                03 G-DATALOTE-CT1.
00116                   05 G-DIA-CT1         PIC XX.
00117                   05 G-MES-CT1         PIC XX.
00118                   05 G-ANO-CT1         PIC XX.
00119                03 G-QTORD-CT1          PIC XX.
00120                03 G-SOMANUM-CT1        PIC X(7).
00121                03 G-SOMAQT-CT1         PIC X(5).
00122                03 G-RESTO-CT1          PIC X(57).

00123          ******************************************************************
00124          *                                                                *
00125          *             AREA DE TRABALHO PARA CARTOES TIPO 2               *
00126          *                                                                *
00127          ******************************************************************

00128             01 G-TABCART2.
00129                03 G-CARTAO2            OCCURS 40 TIMES.
00130                   05 G-TIPO-CT2        PIC X.
00131                   05 G-IDENTCLI-CT2.
00132                      07 G-MAT-CT2      PIC X(6).
00133                      07 G-DIGCTR-CT2   PIC X.
00134                   05 G-NUMOV-CT2       PIC X(5).
00135                   05 G-DATA-CT2.
00136                      07 G-DIA-CT2      PIC XX.
00137                      07 G-MES-CT2      PIC XX.
00138                      07 G-ANO-CT2      PIC XX.
00139                   05 G-DESCRI-CT2      OCCURS 5 TIMES.
00140                      07 G-PROD-CT2.
```

```
5
00141          09 G-COD-CT2 PIC X(8).
00142          09 G-DIGCOD-CT2 PIC X.
00143       07 G-QUANT-CT2  PIC XXX.
00144    05 G-RESTO-CT2     PIC X.
```

```
00145          ***************************************************************
00146          *                                                             *
00147          *            AREA DE REJEICAO DE CARTOES                      *
00148          *                                                             *
00149          ***************************************************************

00150             01 G-AREAREJ.
00151                03 FILLER              PIC X(10) VALUE SPACES.
00152                03 G-CONTEUDO-AR       PIC X(80).
00153                03 G-ERRO-AR           PIC X(31).
00154                03 FILLER              PIC X(25) VALUE SPACES.

00155          ***************************************************************
00156          *                                                             *
00157          *            TABELA CONTENDO AS MENSAGENS DE ERRO             *
00158          *                                                             *
00159          ***************************************************************

00160             01 G-MENSAGEM.
00161                03 FILLER              PIC X(40) VALUE
00162               '"NUMERO DO LOTE" NAO NUMERICO          '.
00163                03 FILLER              PIC X(40) VALUE
00164               '"DATA DO LOTE" NAO NUMERICO            '.
00165                03 FILLER              PIC X(40) VALUE
00166               '"QUANTIDADE DE ORDENS" NAO NUMERICO    '.
00167                03 FILLER              PIC X(40) VALUE
00168               '"SOMA NUM. DE ORDENS" NAO NUMERICO     '.
00169                03 FILLER              PIC X(40) VALUE
00170               '"SOMA QUANTIDADES" NAO NUMERICO        '.
00171                03 FILLER              PIC X(40) VALUE
00172               'CAMPO "24 - 80" NAO BRANCO             '.
00173                03 FILLER              PIC X(40) VALUE
00174               '"DIA" INVALIDO                         '.
00175                03 FILLER              PIC X(40) VALUE
00176               '"MES" INVALIDO                         '.
```

```
00177                03 FILLER              PIC X(40) VALUE
00178               '"ANO" INVALIDO                          '.
00179                03 FILLER              PIC X(40) VALUE
00180               '"QUANTIDADE DE ORDENS" FORA DOS LIMITES '.
00181                03 FILLER              PIC X(40) VALUE
00182               '"IDENTIFICACAO DO CLIENTE" NAO NUMERICO '.
00183                03 FILLER              PIC X(40) VALUE
00184               '"NUMERO DA ORDEM DE VENDA" NAO NUMERICO '.
00185                03 FILLER              PIC X(40) VALUE
00186               '"DATA DA COMPRA" NAO NUMERICO           '.
00187                03 FILLER              PIC X(40) VALUE
00188               '"DESCRICAO DO PRODUTO" NAO NUMERICO     '.
00189                03 FILLER              PIC X(40) VALUE
00190               '"COLUNA 80" NAO BRANCO                  '.
00191                03 FILLER              PIC X(40) VALUE
00192               '"IDENTIFICACAO DO CLIENTE" INVALIDO     '.
00193                03 FILLER              PIC X(40) VALUE
00194               '"DIA" INVALIDO                          '.
00195                03 FILLER              PIC X(40) VALUE
00196               '"MES" INVALIDO                          '.
00197                03 FILLER              PIC X(40) VALUE
00198               '"ANO" INVALIDO                          '.
00199                03 FILLER              PIC X(40) VALUE
00200               '"PRODUTO" INVALIDO                      '.
00201                03 FILLER              PIC X(40) VALUE
00202               '"SOMA NUMEROS DE ORDEM" NAO CONFERE     '.
00203                03 FILLER              PIC X(40) VALUE
00204               '"SOMA DAS QUANTIDADES" NAO CONFERE      '.
00205                03 FILLER              PIC X(40) VALUE
00206               'CLIENTE NAO ESTA CATALOGADO             '.
00207                03 FILLER              PIC X(40) VALUE
00208               'CLIENTE COM CREDITO SUSPENSO            '.
00209                03 FILLER              PIC X(40) VALUE
00210               'CLIENTE COM TRES FATURAS EM ABERTO      '.

00211            01 PS-TABMENSAGEM         REDEFINES G-MENSAGEM.
00212               03 PS-MEN-TB           OCCURS 25 TIMES PIC X(40).
```

```
   7
00213          ******************************************************************
00214          *                                                                *
00215          *           AREA DE IMPRESSAO DO RELATORIO: LINHAS DE CABECALHO  *
00216          *                                                                *
00217          ******************************************************************

00218             01 IC-CAB1.
00219                03 FILLER              PIC X(4) VALUE SPACES.
00220                03 IC-TIT1-CB1         PIC X(42) VALUE 'SISTEMA DE ADMINISTRA
00221          -                            'CAO DE COMPRA E VENDA'.
00222                03 FILLER              PIC X(71) VALUE SPACES.
00223                03 IC-TIT2-CB1         PIC X(8) VALUE 'PAGINA:
00224                03 IC-PAG-CB1          PIC X(3).
00225                03 FILLER              PIC X(4) VALUE SPACES.

00226             01 IC-CAB2.
00227                03 FILLER              PIC X(4) VALUE SPACES.
00228                03 IC-TIT1-CB2         PIC X(37) VALUE 'SUBSISTEMA DE PROCESS
00229          -                            'AMENTO DE ORDENS'.
00230                03 FILLER              PIC X(91) VALUE SPACES.

00231             01 IC-CAB3.
00232                03 FILLER              PIC X(4) VALUE SPACES.
00233                03 IC-TIT1-CB3         PIC X(48) VALUE 'RELATORIO DE CRITICA
00234          -                            'DE LOTES DE ORDENS DE VENDA'.
00235                03 FILLER              PIC X(62) VALUE SPACES.
00236                03 IC-TIT2-CB3         PIC X(6) VALUE 'DATA: '
00237                03 IC-DATA-CB3         PIC X(8).
00238                03 FILLER              PIC X(4) VALUE SPACES.
```

```
00239            01 IC-CAB4.
00240               03 FILLER              PIC X(8) VALUE SPACES.
00241               03 IC-TIT1-CB4         PIC X(69) VALUE 'TIPO   NO. LOTE   DAT
00242            -                        'A   LOTE   QT.  ORDENS   SOMA NUM.
00243            -                        'SOMA QUANT.'.
00244               03 FILLER              PIC X(23) VALUE SPACES.
00245               03 IC-TIT2-CB4         PIC X(4) VALUE 'ERRO'.
00246               03 FILLER              PIC X(28) VALUE SPACES.

00247            01 IC-CAB5.
00248               03 FILLER              PIC X(15) VALUE SPACES.
00249               03 IC-TIT1-CB5         PIC X(60) VALUE 'IDENTIF.   NO.DA ORDE
00250            -                        'M   DATA COMPRA    PRODUTO        QUAN
00251            -                        'T.'.
00252               03 FILLER              PIC X(57) VALUE SPACES.
00253            01 IC1-CAB6.
00254               03 FILLER              PIC X(4) VALUE SPACES.
00255               03 IC1-TIT-CB5         PIC X(13) VALUE 'LOTE NUMERO: '.
00256               03 IC1-NLOTE-CB6       PIC 99.
00257               03 FILLER              PIC X(113) VALUE SPACES.
```

8

```
00258      ******************************************************************
00259      *                                                                *
00260      *           AREA DE IMPRESSAO DE RELATORIO: LINHAS DE DETALHE    *
00261      *                                                                *
00262      ******************************************************************

00263         01 IC1-DET1.
00264            03 FILLER              PIC X(10) VALUE SPACES.
00265            03 IC1-TIPO-DT1        PIC X.
00266            03 FILLER              PIC X(7) VALUE SPACES.
00267            03 IC1-NLOTE-DT1       PIC XX.
00268            03 FILLER              PIC X(7) VALUE SPACES.
00269            03 IC1-DATALOTE-DT1    PIC X(8).
00270            03 FILLER              PIC X(9) VALUE SPACES.
00271            03 IC1-QTORD-DT1       PIC XX.
00272            03 FILLER              PIC X(9) VALUE SPACES.
00273            03 IC1-SOMANUM-DT1     PIC X(7).
00274            03 FILLER              PIC X(7) VALUE SPACES.
00275            03 IC1-SOMAQT-DT1      PIC X(5).
00276            03 FILLER              PIC X(8) VALUE SPACES.
00277            03 IC1-ERRO-DT1        PIC X(40).
00278            03 FILLER              PIC X(10) VALUE SPACES.
```

```
00279          01 IC2-DET2.
00280             03 FILLER              PIC X(10) VALUE SPACES.
00281             03 IC2-TIPO-DT2        PIC X.
00282             03 FILLER              PIC X(4) VALUE SPACES.
00283             03 IC2-IDENT-DT2       PIC X(7).
00284             03 FILLER              PIC X(6) VALUE SPACES.
00285             03 IC2-NUMOR-DT2       PIC X(5).
00286             03 FILLER              PIC X(9) VALUE SPACES.
00287             03 IC2-DATA-DT2        PIC X(8).
00288             03 FILLER              PIC X(4) VALUE SPACES.

00289             03 IC2-PROD-DT2        PIC X(9).
00290             03 FILLER              PIC X97) VALUE SPACES.
00291             03 IC2-QUANT-DT2       PIC XXX.
00292             03 FILLER              PIC X(9) VALUE SPACES.
00293             03 IC2-ERRO-DT2        PIC X(40).
00294             03 FILLER              PIC X(10) VALUE SPACES.
00295          01 ID3-DET3.
00296             03 FILLER              PIC X(54) VALUE SPACES.
00297             03 ID3-PROD-DT3        PIC X(9).
00298             03 FILLER              PIC X(7) VALUE SPACES.
00299             03 ID3-QUANT-DT3       PIC XXX.
00300             03 FILLER              PIC X(9) VALUE SPACES.
00301             03 ID3-ERRO-DT3        PIC X(40).
00302             03 FILLER              PIC X(10) VALUE SPACES.
```

9

```
00303          ****************************************************************
00304          *                                                              *
00305          *            TABELA PARA ERROS DO CARTAO TIPO 1                *
00306          *                                                              *
00307          ****************************************************************

00308             01 G-TABCAR1.
00309                03 G-LINHA-TB1        OCCURS 12 TIMES PIC 99.

00310          ****************************************************************
00311          *                                                              *
00312          *            TABELA PARA ERROS DOS CARTOES TIPO 2: VARIAVEL    *
00313          *                                                              *
00314          ****************************************************************

00315             01 G-TABVARCAR2.
00316                03 G-LIN-VAR           OCCURS 40 TIMES.
00317                   05 G-COL-VAR        OCCURS 13 TIMES PIC 99.

00318          ****************************************************************
00319          *                                                              *
00320          *            TABELA PARA ERROS DOS CARTOES TIPO 2: FIXA        *
00321          *                                                              *
00322          ****************************************************************

00323             01 G-TABFIXCAR2.
00324                03 G-LIN-FIX           OCCURS 40 TIMES.
00325                   05 G-COL-FIX        OCCURS 5 TIMES PIC 99.

00326          ****************************************************************
00327          *                                                              *
00328          *            VARIAVEIS AUXILIARES                              *
00329          *                                                              *
00330          ****************************************************************
```

```
00331          01 IC-DATA.
00332             03 IC-MES-DT          PIC XX.
00333             03 IC-BARRA1DT        PIC X.
00334             03 IC-DIA-DT          PIC XX.
00335             03 IC-RESTO-DT        PIC XXX.
00336          01 IC2-DATA.
00337             03 IC2-DIA-DT         PIC XX.
00338             03 IC2-BARRA-DT       PIC X VALUE '/'.
00339             03 IC2-MES-DT         PIC XX.
00340             03 IC2-BARRA1-DT      PIC X VALUE '/'.
00341             03 IC2-ANO-DT         PIC XX.

00342          01 VC-TESTE1.
00343             03 VC-D6              PIC 9.
00344             03 VC-D5              PIC 9.
00345             03 VC-D4              PIC 9.
00346             03 VC-D3              PIC 9.
00347             03 VC-D2              PIC 9.
00348             03 VC-D1              PIC 9.

00349          01 VRC-TESTE2.
00350             03 VRC-D8             PIC 9.
00351             03 VCR-D7             PIC 9.
00352             03 VCR-D6             PIC 9.
00353             03 VCR-D5             PIC 9.
00354             03 VCR-D4             PIC 9.
00355             03 VCR-D3             PIC 9.
00356             03 VCR-D2             PIC 9.
00357             03 VCR-D1             PIC 9.
```

```
  10

00358          ***************************************************************
00359          *                        NIVEL -1                             *
00360          ***************************************************************

00361           PROCEDURE DIVISION.

00362           INICIALIZAR SECTION.
00363               OPEN INPUT CARTAO.
00364               OPEN OUTPUT RELAT.
00365               OPEN INPUT ARQCLI.
00366               OPEN OUTPUT ARQPED.
00367               MOVE 1 TO IC-NPAG.
00368               MOVE CURRENT-DATE TO IC-DATA.
00369               MOVE IC-MES-DT TO IC-SALVA.
00370               MOVE IC-DIA-DT TO IC-MES-DT.
00371               MOVE IC-SALVA TO IC-DIA-DT.
00372               MOVE IC-DATA TO IC-DATA-CB3.
00373               PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00374               PERFORM LPC-LER-PRIMEIRO-CARTAO THRU PRIMEIRO-CARTAO-LIDO.
```

```
00375          PROCESSAR SECTION.
00376         *    _ENQUANTO G-HALOTE IS = 1, LER E PROCESSAR LOTES.
00377               PERFORM LPL-LER-PROCESSAR-LOTE THRU
00378                 LOTES-LIDOS-PROCESSADOS
00379               UNTIL G-HALOTE IS NOT = 1.

00380          FINALIZAR SECTION.
00381              CLOSE CARTAO.
00382              CLOSE RELAT.
00383              CLOSE ARQCLI.
00384              CLOSE ARQPED.
00385              STOP RUN.

00386          LPL-LER-PROCESSAR-LOTES.
00387              MOVE 0 TO G-LOTEREJ.
00388              PERFORM LL-LER-LOTE THRU LOTE-LIDO.
00389              IF G-LOTEREJ IS = 0
00390                THEN
00391                  PERFORM PL-PROCESSAR-LOTE THRU LOTE-PROCESSADO.
00392          LOTES-LIDOS-PROCESSADOS.
```

```
  11

00393          ****************************************************************
00394          *                        NIVEL -2                              *
00395          ****************************************************************

00396           LPC-LER-PRIMEIRO-CARTAO.
00397               MOVE 1 TO LPC-HACARTAO.
00398               MOVE 1 TO G-HALOTE.
00399               MOVE 0 TO F-TIPO-CT.
00400          *    _ENQUANTO LPC-HACARTAO = 1 AND F-TIPO-CT IS NOT = 1, OBTER
00401          *                                                        CARTAO.
00402                 PERFORM OC-OBTER-CARTAO THRU CARTAO-OBTIDO
00403                 UNTIL LPC-HACARTAO IS NOT = 1 OR F-TIPO-CT IS = 1.
00404           PRIMEIRO-CARTAO-LIDO.


00405           OC-OBTER-CARTAO.
00406               READ CARTAO
00407                 AT END MOVE 0 TO LPC-HACARTAO,
00408                        MOVE 0 TO G-HALOTE.
00409               IF LPC-HACARTAO IS = 1
00410                 THEN
00411                   IF F-TIPO-CT IS NOT = 1
00412                     THEN
00413                       MOVE 'PRIMEIRO CARTAO DEVE SER TIPO 1' TO G-ERRO-AR
00414                       PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
00415           CARTAO-OBTIDO.
```

```
00416           LL-LER-LOTE.
00417               MOVE ALL SPACES TO G-TABCART2.
00418               MOVE ALL '00' TO G-TABCAR1, G-TABVARCAR2, G-TABFIXCAR2.
00419               MOVE 0 TO LL-PONTEIRO.
00420               MOVE 0 TO LL-FIMLOTE.
00421               MOVE F-CARTAO TO G-CARTAO1.
00422           *   _ENQUANTO LL-FIMLOTE IS = 0, OBTER LOTE.
00423                 PERFORM OL-OBTER-LOTE THRU LOTE-OBTIDO
00424                 UNTIL LL-FIMLOTE IS NOT = 0.
00425           LOTE-LIDO.


00426           OL-OBTER-LOTE.
00427               READ CARTAO
00428                 AT END MOVE 0 TO G-HALOTE,
00429                        MOVE 1 TO LL-FIMLOTE,
00430                        GO TO LOTE-OBTIDO.
00431               IF T-TIPO-CT IS NOT NUMERICO
00432                 THEN
00433                   MOVE 3 TO F-TIPO-CT.
00434               MOVE F-TIPO-CT TO OL-TIPO.
00435               GO TO CT1-CARTAO-TIPO-1, CT2-CARTAO-TIPO-2,
00436                 DEPENDING ON OL-TIPO.

00437             OT-OUTRO-TIPO.
00438                   MOVE 'CARTAO TIPO INVALIDO           ' TO G-ERRO-AR.
00439                   PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
00440                   GO TO LOTE-OBTIDO.
```

```
00441           CT1-CARTAO-TIPO-1.
00442                 MOVE 1 TO LL-FIMLOTE.
00443                 GO TO LOTE-OBTIDO.

00444           CT2-CARTAO-TIPO-2.
00445                 IF LL-PONTEIRO >  G-MAXLOTE OR LL-PONTEIRO = G-MAXLOTE
00446                   THEN
00447                     PERFORM RFL-REJEITAR-ATE-FIM-LOTE THRU
00448                       ATE-FIM-LOTE-REJEITADOS
00449                     MOVE 1 TO G-LOTEREJ, LL-FIMLOTE
00450                   ELSE
00451                     ADD 1 TO LL-PONTEIRO
00452                     MOVE F-CARTAO TO G-CARTAO2 (LL-PONTEIRO).
00453           LOTE-OBTIDO. EXIT.
```

```
00454          PL-PROCESSAR-LOTE.
00455              MOVE 0 TO PL-LOTEREJ.
00456              PERFORM AC1-ANALISAR-CARTAO-1 THRU CARTAO-1-ANALISADO.
00457              IF PL-LOTEREJ IS = 0
00458                THEN
00459                  PERFORM AC2-ANALISAR-CARTOES-2 THRU
00460                    CARTOES-2-ANALISADOS
00461                  PERFORM ATC-ANALISAR-TOTAIS-CONTROLE THRU
00462                    TOTAIS-CONTROLE-ANALISADOS.
00463              IF PL-LOTEREJ IS = 0
00464                THEN
00465                  PERFORM CC-CRITICAR-CLIENTES THRU CLIENTES-CRITICADOS.
00466              PERFORM PS-PROCESSAR-SAIDAS THRU SAIDAS-PROCESSADAS.
00467          LOTE-PROCESSADO.
```

14

```
00468          ****************************************************************
00469          *                          NIVEL -3                            *
00470          ****************************************************************

00471           RFL-REJEITAR-ATE-FIM-LOTE.
00472               MOVE 'LOTE COM MAIS DE 40 CARTOES                TO
00473                 IC1-ERRO-DT1.
00474               PERFORM IC1-IMPRIMIR-CARTAO-1 THRU CARTAO-1-IMPRESSO.
00475               PERFORM RCT-REJEITAR-CARTOES-TABELA THRU
00476                 CARTOES-TABELA-REJEITADOS.
00477          *    _ENQUANTO F-TIPO-CT IS NOT = 1 AND G-HALOTE IS = 1, LER
00478          *                                          REJEITAR.
00479                 PERFORM LP-LER-REJEITAR THRU LIDO-REJEITADO
00480                 UNTIL F-TIPO-CT IS = 1 OR G-HALOTE IS NOT = 1.
00481           ATE-FIM-LOTE-REJEITADOS.


00482           LR-LER-REJEITAR.
00483               MOVE 'LOTE ERRADO: CARTAO REJEITADO  ' TO G-ERRO-AR.
00484               PERFORM RC-REJEITAR-CARTAO THRU CARTAO-REJEITADO.
00485               READ CARTAO
00486                 AT END MOVE 0 TO G-HALOTE.
00487           LIDO-REJEITADO.
```

```
  15

00488           AC1-ANALISAR-CARTAO-1.
00489               MOVE 0 TO AC1-IND.
00490               IF G-NUMLOTE-CT1 IS NOT NUMERIC
00491                 THEN
00492                   ADD 1 TO AC1-IND
00493                   MOVE 01 TO G-LINHA-TB1 (AC1-IND).
00494               IF G-DATALOTE-CT1 IS NOT NUMERIC
00495                 THEN
00496                   ADD 1 TO AC1-IND
00497                   MOVE 02 TO G-LINHA-TB1 (AC1-IND).
00498               IF G-QTORD-CT1 IS NOT NUMERIC
00499                 THEN
00500                   ADD 1 TO AC1-IND
00501                   MOVE 03 TO G-LINHA-TB1 (AC1-IND).
00502               IF G-SOMANUM-CT1 IS NOT NUMERIC
00503                 THEN
00504                   ADD 1 TO AC1-IND
00505                   MOVE 04 TO G-LINHA-TB1 (AC1-IND).
00506               IF G-SOMAQT-CT1 IS NOT NUMERIC
00507                 THEN
00508                   ADD 1 TO AC1-IND
00509                   MOVE 05 TO G-LINHA-TB1 (AC1-IND).
00510               IF G-RESTO-CT1 IS NOT = SPACES
00511                 THEN
00512                   ADD 1 TO AC1-IND
00513                   MOVE 06 TO G-LINHA-TB1 (AC1-IND).
00514               IF G-DIA-CT1 IS < 01 OR G-DIA-CT1 IS > 31
00515                 THEN
00516                   ADD 1 TO AC1-IND
00517                   MOVE 07 TO G-LINHA-TB1 (AC1-IND).
00518               IF G-MES-CT1 IS < 01 OR G-MES-CT1 IS > 12
00519                 THEN
00520                   ADD 1 TO AC1-IND
00521                   MOVE 08 TO G-LINHA-TB1 (AC1-IND).
00522               IF G-ANO-CT1 IS < 75
00523                 THEN
00524                   ADD 1 TO AC1-IND
00525                   MOVE 09 TO G-LINHA-TB1 (AC1-IND).
00526               IF G-QTORD-CT1 IS > 20 OR G-QTORD-CT1 IS < 01
00527                 THEN
00528                   ADD 1 TO AC1-IND
00529                   MOVE 10 TO G-LINHA-TB1 (AC1-IND).
00530               IF AC1-IND IS NOT = 0
00531                 THEN
00532                   MOVE 1 TO PL-LOTEREJ.
00533           CARTAO-1-ANALISADO.
```

```
00534          AC2-ANALISAR-CARTOES-2.
00535              MOVE 0 TO AC2-TOTAL1, AC2-TOTAL2.
00536              MOVE 0 TO AC2-COLUNA.
00537              MOVE 1 TO AC2-LINHA.
00538         *    _ENQUANTO AC2-LINHA IS <= LL-PONTEIRO, VERIFICAR CAMPOS.
00539                PERFORM VC-VERIFICAR-CAMPOS THRU CAMPOS-VERIFICADOS
00540                UNTIL AC2-LINHA IS > LL-PONTEIRO.
00541          CARTOES-2-ANALISADOS.


00542          VC-VERIFICAR-CAMPOS.
00543              IF G-IDENTCLI-CT2 (AC2-LINHA) IS NOT NUMERIC
00544                THEN
00545                  ADD 1 TO AC2-COLUNA
00546                  MOVE 11 TO G-COL-VAR (AC2-LINHA,AC2-COLUNA).
00547              IF G-NUMOV-CT2 (AC2-LINHA) IS NOT NUMERIC
00548                THEN
00549                  ADD 1 TO AC2-COLUNA
00550                  MOVE 12 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA)
00551                ELSE
00552                  MOVE G-NUMOV-CT2 (AC2-LINHA) TO AC2-NUMOV
00553                  ADD AC2-NUMOV TO AC2-TOTAL1.
00554              IF G-DATA-CT2 (AC2-LINHA) IS NOT NUMERIC
00555                THEN
00556                  ADD 1 TO AC2-COLUNA
00557                  MOVE 13 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00558              IF G-DESCRI-CT2 (AC2-LINHA, 1) IS NOT NUMERIC
00559                THEN
00560                  ADD 1 TO AC2-COLUNA
00561                  MOVE 14 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA)
00562                ELSE
00563                  MOVE G-QUANT-CT2 (AC2-LINHA, 1) TO AC2-QUANT
00564                  ADD AC2-QUANT TO AC2-TOTAL2.
00565              IF G-RESTO-CT2 (AC2-LINHA) IS NOT = ' '
00566                THEN
00567                  ADD 1 TO AC2-COLUNA
00568                  MOVE 15 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00569              MOVE G-MAT-CT2 (AC2-LINHA) TO VC-TESTE1.
```

```
00570            COMPUTE VC-DIVIDENDO1 = VC-D6 * 6 + VC-D5 * 5 + VC-D4 * 4 +
00571                                    VC-D3 * 3 + VC-D2 * 2 + VC-D1.
00572            COMPUTE VC-RESTO1 = VC-DIVIDENDO1 - VC-DIVIDENDO1 / 11 * 11.
00573            MOVE G-DIGCTR-CT2 (AC2-LINHA) TO VC-DIGCTR.
00574            IF VC-DIGCTR IS NOT = VC-RESTO1
00575              THEN
00576                ADD 1 TO AC2-COLUNA
00577                MOVE 16 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00578            IF G-DIA-CT2 (AC2-LINHA)  < 01 OR G-DIA-CT2 (AC2-LINHA) > 31
00579              THEN
00580                ADD 1 TO AC2-COLUNA
00581                MOVE 17 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00582            IF G-MES-CT2 (AC2-LINHA) < 01 OR G-MES-CT2 (AC2-LINHA) > 12
00583              THEN
00584                ADD 1 TO AC2-COLUNA
00585                MOVE 18 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00586            IF G-ANO-CT2 (AC2-LINHA) IS < 75
00587              THEN
00588                ADD 1 TO AC2-COLUNA
00589                MOVE 19 TO G-COL-VAR (AC2-LINHA, AC2-COLUNA).
00590            MOVE 0 TO G-COL-FIX (AC2-LINHA, 1).
00591            MOVE 2 TO VC-INDEX.
00592        *   _ENQUANTO VC-INDEX IS < 6, VERIFICAR RESTO CARTAO2.
00593              PERFORM VRC-VERIFICAR-RESTO-CARTAO2 THRU
00594                RESTO-CARTAO2-VERIFICADO
00595              UNTIL VC-INDEX IS NOT < 6.
00596            MOVE 0 TO AC2-COLUNA.
00597            ADD 1 TO AC2-LINHA.
00598         CAMPOS-VERIFICADOS.


00599         VRC-VERIFICAR-RESTO-CARTAO2.
00600            IF G-DESCRI-CT2 (AC2-LINHA, VC-INDEX) IS NOT = SPACES
00601              THEN
00602                MOVE G-COD-CT2 (AC2-LINHA, VC-INDEX) TO VRC-TESTE2
00603                COMPUTE VRC-DIVIDENDO2 = VRC-D8 * 8 + VRC-D7 * 7 +
00604                  VRC-D6 * 6 + VRC-D5 * 5 + VRC-D4 * 4 + VRC-D3 * 3 +
00605                  VRC-D2 * 2 + VRC-D1
00606                COMPUTE VRC-RESTO2 = VRC-DIVIDENDO2 - VRC-DIVIDENDO2 / 11
```

17

```
00607                    * 11
00608                  MOVE G-QUANT-CT2 (AC2-LINHA, VC-INDEX) TO AC2-QUANT
00609                  ADD AC2-QUANT TO AC2-TOTAL2
00610                  MOVE G-DIGCOD-CT2 (AC2-LINHA, VC-INDEX) TO VRC-DIGCOD
00611                  IF VRC-DIGCOD IS NOT = VRC-RESTO2
00612                    THEN
00613                      MOVE 20 TO G-COL-FIX (AC2-LINHA, VC-INDEX)
00614                      ADD 1 TO G-COL-FIX (AC2-LINHA, 1),
00615              ADD 1 TO VC-INDEX.
00616          RESTO-CARTAO2-VERIFICADO.
```

18

```
00617           ATC-ANALISAR-TOTAIS-CONTROLE.
00618               IF G-SOMANUM-CT1 IS NOT = AC2-TOTAL1
00619                 THEN
00620                   ADD 1 TO AC1-IND
00621                   MOVE 21 TO G-LINHA-TB1 (AC1-IND)
00622                   MOVE 1 TO PL-LOTEREJ.
00623               IF G-SOMAQT-CT1 IS NOT = AC2-TOTAL2
00624                 THEN
00625                   ADD 1 TO AC1-IND
00626                   MOVE 22 TO G-LINHA-TB1 (AC1-IND)
00627                   MOVE 1 TO PL-LOTEREJ.
00628           TOTAIS-CONTROLE-ANALISADOS.
```

19

```
00629          CC-CRITICAR-CLIENTES.
00630              MOVE 1 TO CC-LINHA, CC-COLUNA.
00631          *   ENQUANTO CC-LINHA <= LL-PONTEIRO, VER SITUACAO CLIENTE.
00632              PERFORM VSC-VER-SITUACAO-CLIENTE THRU
00633                SITUACAO-CLIENTE-VISTA
00634              UNTIL CC-LINHA IS > LL-PONTEIRO.
00635          CLIENTES-CRITICADOS.


00636          VSC-VER-SITUACAO-CLIENTE.
00637              IF G-COL-VAR (CC-LINHA, 1) = 0 AND G-COL-FIX (CC-LINHA,
00638                1) IS = 0
00639                THEN
00640                  PERFORM LCC-LER-CRITICAR-CLIENTE THRU
00641                    CLIENTE-LIDO-CRITICADO.
00642              ADD 1 TO CC-LINHA.
00643          SITUACAO-CLIENTE-VISTA.
```

20

```
00644           PS-PROCESSAR-SAIDAS.
00645               IF PL-LOTEREJ IS = 1
00646                 THEN
00647                   PERFORM ICL-IMPRIMIR-CABECA-LOTE THRU
00648                     CABECA-LOTE-IMPRESSA
00649                   PERFORM RCT-REJEITAR-CARTOES-TABELA THRU
00650                     CARTOES-TABELA-REJEITADOS
00651                 ELSE
00652                   PERFORM PSC-PROCESSAR-SAIDA-CARTOES2 THRU
00653                     SAIDA-CARTOES2-PROCESSADA.
00654           SAIDAS-PROCESSADAS.
```

21

```
00655          ****************************************************************
00656          *                        NIVEL -4                              *
00657          ****************************************************************

00658           IC1-IMPRIMIR-CARTAO-1.
00659               MOVE G-NUMLOTE-CT1 TO IC1-NLOTE-CB6.
00660               IF G-NLINHA IS > 64
00661                 THEN
00662                   PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00663               WRITE F-LINHA FROM IC1-CAB6 AFTER ADVANCING 2 LINES.
00664               MOVE G-TIPO-CT1 TO IC1-TIPO-DT1.
00665               MOVE G-NUMLOTE-CT1 TO IC1-NLOTE-DT1.
00666               MOVE G-DATALOTE-CT1 TO IC1-DATALOTE-DT1.
00667               MOVE G-QTORD-CT1 TO IC1-QTORD-DT1.
00668               MOVE G-SOMANUM-CT1 TO IC1-SOMANUM-DT1.
00669               MOVE G-SOMAQT-CT1 TO IC1-SOMAQT-DT1.
00670               WRITE F-LINHA FROM IC1-DET1 AFTER ADVANCING 1 LINES.
00671               ADD 3 TO G-NLINHA.
00672           CARTAO-1-IMPRESSO.
```

```
00673          LCC-LER-CRITICAR-CLIENTE.
00674              MOVE G-IDENTCLI-CT2 (CC-LINHA) TO G-NOMINAL.
00675              READ ARQCLI
00676                INVALID KEY, MOVE 23 TO G-COL-VAR (CC-LINHA, CC-COLUNA),
00677                             GO TO CLIENTE-LIDO-CRITICADO.
00678              IF F-COND-CLI IS = 0
00679                THEN
00680                  MOVE 24 TO G-COL-VAR (CC-LINHA, CC-COLUNA)
00681                  ADD 1 TO CC-COLUNA.
00682              MOVE 1 TO LCC-INDICE, LCC-CHAVE.
00683          *    ENQUANTO LCC-INDICE <= 3 AND LCC-CHAVE IS = 1, VERIFICAR
00684          *                             CONTA CORRENTE
00685                PERFORM VCC-VERIFICAR-CTA-CORRENTE THRU
00686                  CTA-CORRENTE-VERIFICADA
00687                UNTIL LCC-INDICE IS > 3 OR LCC-CHAVE IS NOT = 1.
00688              IF LCC-INDICE IS > 3
00689                THEN
00690                   MOVE 25 TO G-COL-VAR (CC-LINHA, CC-COLUNA).
00691          CLIENTE-LIDO-CRITICADO.


00692          VCC-VERIFICAR-CTA-CORRENTE.
00693              IF F-OCUP-CLI (LCC-INDICE) IS = 0
00694                THEN
00695                  MOVE 0 TO LCC-CHAVE
00696                ELSE
00697                  ADD 1 TO LCC-INDICE.
00698          CTA-CORRENTE-VERIFICADA.
```

23

```
00699           ICL-IMPRIMIR-CABECA-LOTE.
00700               MOVE G-LINHA-TB1 (1) TO ICL-ERRO.
00701               MOVE PS-MEN-TB (ICL-ERRO) TO IC1-ERRO-DT1.
00702               PERFORM IC1-IMPRIMIR-CARTAO-1 THRU CARTAO-1-IMPRESSO.
00703               MOVE SPACES TO IC1-DET1.
00704               MOVE 2 TO ICL-INDICE.
00705           *   _ENQUANTO G-LINHA-TB1 (ICL-INDICE) ¬= 0, IMPRIMIR MENSAGEM.
00706                 PERFORM IM-IMPRIMIR-MENSAGEM THRU MENSAGEM-IMPRESSA
00707                 UNTIL G-LINHA-TB1 (ICL-INDICE) IS = 0.
00708               MOVE SPACES TO F-LINHA.
00709               WRITE F-LINHA AFTER ADVANCING 1 LINES.
00710               ADD 1 TO G-NLINHA.
00711           CABECA-LOTE-IMPRESSA.


00712           IM-IMPRIMIR-MENSAGEM.
00713               IF G-NLINHA IS > 70
00714                 THEN
00715                   PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00716               MOVE SPACES TO IC1-DET1.
00717               MOVE G-LINHA-TB1 (ICL-INDICE) TO ICL-ERRO.
00718               MOVE PS-MEN-TB (ICL-ERRO) TO IC1-ERRO-DT1.
00719               WRITE F-LINHA FROM IC1-DET1 AFTER ADVANCING 1 LINES.
00720               ADD 1 TO G-NLINHA.
00721               ADD 1 TO ICL-INDICE.
00722           MENSAGEM-IMPRESSA.
```

```
00723          PSC-PROCESSAR-SAIDA-CARTOES2.
00724              MOVE 1 TO PSC-IND.
00725          *   _ENQUANTO PSC-IND <= LL-PONTEIRO, IMPRIMIR GRAVAR.
00726                PERFORM IG-IMPRIMIR-GRAVAR THRU IMPRESSO-GRAVADO
00727                UNTIL PSC-IND IS > LL-PONTEIRO.
00728          SAIDA-CARTOES2-PROCESSADA.


00729          IG-IMPRIMIR-GRAVAR.
00730              IF G-COL-VAR (PSC-IND, 1) IS NOT = 0 OR G-COL-VAR
00731                (PSC-IND, 1) NOT = 0
00732                THEN
00733                  PERFORM IC2-IMPRIMIR-CARTAO2 THRU CARTAO2-IMPRESSO
00734                ELSE
00735                  PERFORM GAP-GRAVAR-ARQUIVO-PEDIDOS THRU
00736                    ARQUIVO-PEDIDO-GRAVADO.
00737              ADD 1 TO PSC-IND.
00738          IMPRESSO-GRAVADO.
```

25

```
00739          ****************************************************************
00740          *                          NIVEL -5                            *
00741          ****************************************************************

00742           RCT-REJEITAR-CARTOES-TABELA.
00743               MOVE SPACES TO F-LINHA.
00744               WRITE F-LINHA AFTER ADVANCING 1 LINES.
00745               ADD 1 TO G-NLINHA.
00746               MOVE 1 TO RCT-LIN.
00747          *    _ENQUANTO RCT-LIN <= LL-PONTEIRO, REJEITAR CARTAO 2.
00748                 PERFORM RC2-REJEITAR-CARTAO2 THRU CARTAO2-REJEITADO
00749                 UNTIL RCT-LIN IS >  LL-PONTEIRO.
00750           CARTOES-TABELA-REJEITADOS.


00751           RC2-REJEITAR-CARTAO2.
00752               MOVE 'LOTE ERRADO: CARTAO REJEITADO  ' TO G-ERRO-AR.
00753               MOVE G-CARTAO2 (RCT-LIN) TO G-CONTEUDO-AR.
00754               IF G-NLINHA IS > 70
00755                 THEN
00756                   PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00757               WRITE F-LINHA FROM G-AREAREJ AFTER ADVANCING 1 LINES.
00758               ADD 1 TO G-NLINHA, RCT-LIN.
00759           CARTAO2-REJEITADO.
```

```
00760          IC2-IMPRIMIR-CARTAO2.
00761              MOVE G-COL-VAR (PSC-IND, 1) TO IC2-ERRO.
00762              MOVE PS-MEN-TB (IC2-ERRO) TO IC2-ERRO-DT2.
00763              MOVE G-TIPO-CT2 (PSC-IND) TO IC2-TIPO-DT2.
00764              MOVE G-IDENTCLI-CT2 (PSC-IND) TO IC2-IDENT-DT2.
00765              MOVE G-NUMOV-CT2 (PSC-IND) TO IC2-NUMOR-DT2.
00766              MOVE G-DIA-CT2 (PSC-IND) TO IC2-DIA-DT.
00767              MOVE G-MES-CT2 (PSC-IND) TO IC2-MES-DT.
00768              MOVE G-ANO-CT2 (PSC-IND) TO IC2-ANO-DT.
00769              MOVE IC2-DATA TO IC2-DATA-DT2.
00770              MOVE G-PROD-CT2 (PSC-IND, 1) TO IC2-PROD-DT2.
00771              MOVE G-QUANT-CT2 (PSC-IND, 1) TO IC2-QUANT-DT2.
00772              IF G-NLINHA IS  > 70
00773                THEN
00774                  PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00775              WRITE F-LINHA FROM IC2-DET2 AFTER ADVANCING 1 LINES.
00776              ADD 1 TO G-NLINHA.
00777              MOVE 2 TO IC2-COL.
00778          *   _ENQUANTO  G-COL-VAR (PSC-IND, IC2-COL) ¬= 0,
00779          *                        IMPRIMIR ERRO.
00780                PERFORM IE-IMPRIMIR-ERRO THRU ERRO-IMPRESSO
00781                UNTIL G-COL-VAR (PSC-IND, IC2-COL) IS = 0.
00782              MOVE 2 TO IC2-COL.
00783          *   _ENQUANTO IC2-COL <= 5, IMPRIMIR RESTO CARTAO 2.
00784                PERFORM IRC-IMPRIMIR-RESTO-CARTAO2 THRU
00785                  RESTO-CARTAO2-IMPRESSO
00786                UNTIL IC2-COL IS > 5.
00787              MOVE SPACES TO F-LINHA.
00788              WRITE F-LINHA AFTER ADVANCING 1 LINES.
00789              ADD 1 TO G-NLINHA.
00790          CARTAO2-IMPRESSO.
```

```
00791          IE-IMPRIMIR-ERRO.
00792              MOVE SPACES TO ID3-DET3.
00793              MOVE G-COL-VAR (PSC-IND, IC2-COL) TO IC2-ERRO.
00794              MOVE PS-MEN-TB (IC2-ERRO) TO ID3-ERRO-DT3.
00795              IF G-NLINHA IS > 70
00796                THEN
00797                PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00798              WRITE F-LINHA FROM ID3-DET3 AFTER ADVANCING 1 LINES.
00799              ADD 1 TO G-NLINHA, IC2-COL.
00800          ERRO-IMPRESSO.
00801          IRC-IMPRIMIR-RESTO-CARTAO2.
00802              IF G-DESCRI-CT2 (PSC-IND, IC2-COL) IS = SPACES
00803                THEN
00804                  MOVE 6 TO IC2-COL
00805                ELSE
00806                  PERFORM ID3-IMPRIMIR-DETALHE-3 THRU DETALHE-3-IMPRESSO.
00807          RESTO-CARTAO2-IMPRESSO.
```

```
00808          GAP-GRAVAR-ARQUIVO-PEDIDOS.
00809              ADD 1 TO GAP-CHAVE.
00810              MOVE GAP-CHAVE TO F-CHAVE-PED.
00811              MOVE G-NUMOV-CT2 (PSC-IND) TO F-NUMOV-PED.
00812              MOVE G-IDENTCLI-CT2 (PSC-IND) TO F-IDENTCLI-PED.
00813              MOVE G-DATA-CT2 (PSC-IND) TO F-DATA-PED.
00814              MOVE 1 TO GAP-IND.
00815          *    _REPETIR MOVER DESCRICAO PRODUTO ATE QUE GAP-IND = 5.
00816                 PERFORM MDP-MOVER-DESCRICAO-PRODUTO THRU
00817                   DESCRICAO-PRODUTO-MOVIDA.
00818                 PERFORM MDP-MOVER-DESCRICAO-PRODUTO THRU
00819                   DESCRICAO-PRODUTO-MOVIDA
00820                 UNTIL GAP-IND IS = 5.
00821              WRITE F-PEDIDO
00822                INVALID KEY, DISPLAY 'ERRO NA GRAVACAO DO ARQPED'.
00823          ARQUIVO-PEDIDOS-GRAVADO.


00824          MDP-MOVER-DESCRICAO-PRODUTO.
00825              MOVE G-DESCRI-CT2 (PSC-IND, GAP-IND) TO F-DESCRI-PED
00826                (GAP-IND).
00827              ADD 1 TO GAP-IND.
00828          DESCRICAO-PRODUTO-MOVIDA.
```

28

```
00829          ****************************************************************
00830          *                          NIVEL -6                            *
00831          ****************************************************************

00832           RC-REJEITAR-CARTAO.
00833               MOVE F-CARTAO TO G-CONTEUDO-AR.
00834               IF G-NLINHA >  70
00835                 THEN
00836                   PERFORM IC-IMPRIMIR-CABECALHO THRU CABECALHO-IMPRESSO.
00837               WRITE F-LINHA FROM G-AREAREJ AFTER ADVANCING 1 LINES.
00838               ADD 1 TO G-NLINHA.
00839           CARTAO-REJEITADO.
```

```
00840          ID3-IMPRIMIR-DETALHE-3.
00841              MOVE SPACES TO ID3-ERRO-DT3.
00842              MOVE G-PROD-CT2 (PSC-IND, IC2-COL) TO ID3-PROD-DT3.
00843              MOVE G-QUANT-CT2 (PSC-IND, IC2-COL) TO ID3-QUANT-DT3.
00844              IF G-COL-FIX (PSC-IND, IC2-COL) IS NOT = 0
00845                THEN
00846                  MOVE G-COL-FIX (PSC-IND, IC2-COL) TO IC2-ERRO
00847                  MOVE PS-MEN-TB (IC2-ERRO) TO ID3-ERRO-DT3.
00848              IF G-NLINHA IS > 70
00849                THEN
00850                  PERFORM IC-IMPRIMIR-CABECALHO THRU CABEÇALHO-IMPRESSO.
00851              WRITE F-LINHA FROM ID3-DET3 AFTER ADVANCING 1 LINES.
00852              ADD 1 TO G-NLINHA, IC2-COL.
00853          DETALHE-3-IMPRESSO.
```

30

```
00854          ****************************************************************
00855          *                          NIVEL -7                            *
00856          ****************************************************************

00857           IC-IMPRIMIR-CABECALHO.
00858               MOVE IC-NPAG TO IC-PAG-CB1.
00859               MOVE SPACES TO F-LINHA.
00860               WRITE F-LINHA AFTER ADVANCING PAGINA.
00861               WRITE F-LINHA FROM IC-CAB1 AFTER ADVANCING 1 LINES.
00862               WRITE F-LINHA FROM IC-CAB2 AFTER ADVANCING 1 LINES.
00863               WRITE F-LINHA FROM IC-CAB3 AFTER ADVANCING 2 LINES.
00864               WRITE F-LINHA FROM IC-CAB4 AFTER ADVANCING 2 LINES.
00865               WRITE F-LINHA FROM IC-CAB5 AFTER ADVANCING 1 LINES.
00866               MOVE SPACES TO F-LINHA.
00867               WRITE F-LINHA AFTER ADVANCING 1 LINES.
00868               MOVE 1 TO G-NLINHA.
00869               ADD 1 TO IC-NPAG.
00870           CABECALHO-IMPRESSO.
```

RELATORIO DE CRITICA DE LOTES DE ORDENS DE VENDA

DATA: 27/11/75

TIPO	NO. LOTE IDENTIF.	DATA NO.DA LOTE ORDEM	QT. ORDENS DATA COMPRA	SOMA NUM. PRODUTO	SOMA QUANT. QUANT.	ERRO
2	1102141	01300	20/10/75	110000243	100	CLIENTE NAO ESTA CATALOGADO
				110000026	031	
2	1200161	00175	20/10/75	110000026	020	CLIENTE COM CREDITO SUSPENSO
						CLIENTE COM TRES FATURAS EM ABERTO
2	1100317	00028	20/10/75	111100004	005	"COLUNA 80" NAO BRANCO
3110213000315231075110000004100111111118150						CARTAO TIPO INVALIDO
LOTE NUMERO: 02						
1	02	231075	03	0001541	01025	"SOMA NUMEROS DE ORDEM" NAO CONFERE
						"SOMA DAS QUANTIDADES" NAO CONFERE
2241200900301231075110000004005111100004005011100007100001100000200001110004020						LOTE ERRADO: CARTAO REJEITADO
224120090030123107500111100710011111111 8200						LOTE ERRADO: CARTAO REJEITADO
2310146700312232075000111103100000111114100100001000005 1000A000101010001001 2010						LOTE ERRADO: CARTAO REJEITADO
231014670031223107500000997020						LOTE ERRADO: CARTAO REJEITADO
LOTE NUMERO: 03						
1	03	231075	03	0001815	03000	"SOMA DAS QUANTIDADES" NAO CONFERE
2140188900115231075550000111100						LOTE ERRADO: CARTAO REJEITADO
2210444801020231075002001059100						LOTE ERRADO: CARTAO REJEITADO
2220008800680231075002020111100						LOTE ERRADO: CARTAO REJEITADO
2	2104448	01600	23/10/75	000111114	100	CLIENTE COM CREDITO SUSPENSO
						CLIENTE COM TRES FATURAS EM ABERTO
				000111001	003	
2	1101234	00180	23/10/75	500000006	100	CLIENTE COM CREDITO SUSPENSO
2	2200089	00130	23/10/75	110000004	100	"IDENTIFICACAO DO CLIENTE" INVALIDO

Apêndice C

Programação Estruturada em Equipes

Na Parte I, estudamos detalhadamente a Programação Estruturada, que se constitui numa ferramenta que auxilia o programador a obter programas mais corretos e de melhor qualidade, de maneira mais disciplinada. Sem dúvida, o uso desta técnica no desenvolvimento de Sistemas de Processamento de Dados tem trazido benefícios indiscutíveis.

No entanto, em ambientes de produção de programas em projetos de sistemas de grande porte, muitos problemas são mais de ordem gerencial do que tecnológica. Por exemplo, a comunicação entre o pessoal envolvido no projeto, a distribuição de tarefas, a especialização do trabalho, a produtividade das equipes e o controle da programação são alguns dos fatores que geram problemas organizacionais e comportamentais para a gerência de projetos, os quais não podem ser resolvidos pela simples introdução de uma nova técnica de programação.

O advento da Programação Estruturada aliado às facilidades de máquina para Bibliotecas de Programas possibilitou a introdução de um novo enfoque gerencial na produção de programas, unindo estas técnicas a novos tipos de procedimentos e a uma estrutura organizacional bem definida numa metodologia chamada *Chief Programmer Teams*. Esta metodologia foi inicialmente desenvolvida pela Federal Systems Division [5] da IBM Corporation.

"*Chief programmer team* (CPT) é um pequeno grupo de pessoas, organizado para trabalhar eficientemente como uma unidade de programação" [5], utilizando as técnicas de Programação Estruturada, facilidades de Bibliotecas de Programas e uma série de normas de procedimentos.

Podemos identificar dois objetivos principais na metodologia. Ambos dizem respeito à fase de programação:

a) melhorar o controle;
b) aumentar a produtividade.

A metodologia de *Chief Programmer Teams* envolve três elementos principais:

- organização funcional;
- Biblioteca de Produção de Programas; e
- Programação Estruturada.

C.1 – ORGANIZAÇÃO FUNCIONAL

Um CPT compõe-se de um núcleo permanente formado por um programador-chefe, um programador-assistente e um secretário de programação. Além deste núcleo, a equipe pode contar com outros elementos (principalmente programadores), cujo número depende da extensão e prazo do projeto (Fig. C.1). Se durante o desenvolvimento, for sentida a necessidade de reforçar a equipe, cabe ao programador-chefe convocar novos membros.

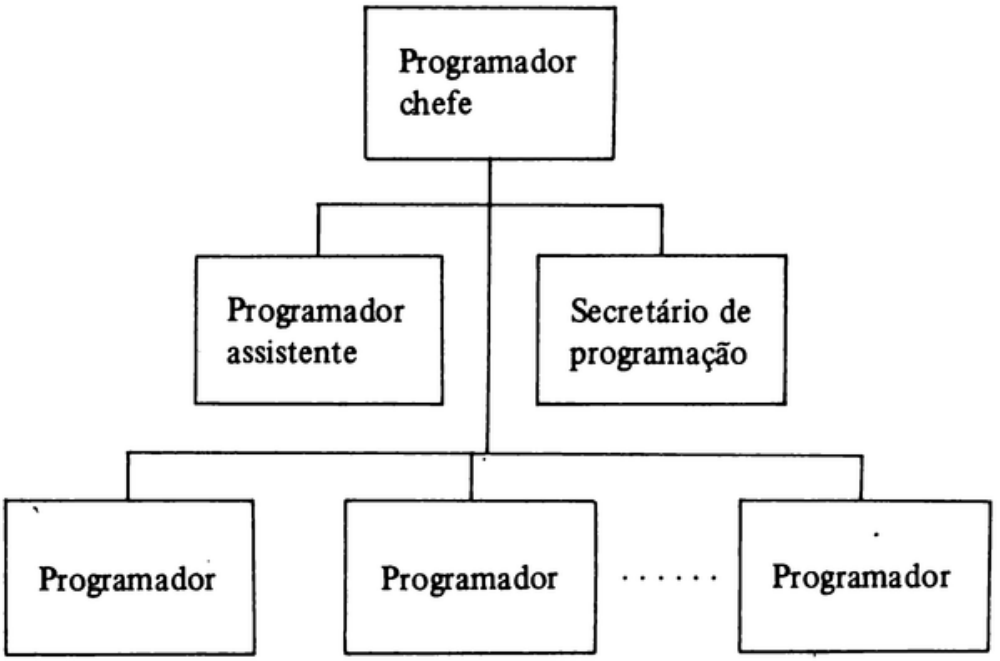


Fig. C.1

C.1.1 – Programador-Chefe

O programador-chefe é um gerente-técnico que acumula duas funções:

a) como gerente, ele é responsável pela manutenção da disciplina de desenvolvimento da programação, pela supervisão da equipe e pelo conteúdo da Biblioteca de Produção de Programas;

b) como técnico, ele projeta e codifica os principais programas em seus níveis mais altos e define as unidades subordinadas, passando-as aos outros membros do grupo, para completá-las. A integração dessas unidades é feita pelo programador-chefe após uma revisão de sua lógica e de seu código.

Para executar suas atribuições e poder arcar com a responsabilidade sobre a programação, o programador-chefe precisa conhecer o sistema para o qual está programando. Ele deve ter acompanhado o desenvolvimento do projeto lógico e do projeto físico ou, mesmo, ter participado ativamente dessas duas fases.

C.1.2 – Programador-Assistente

Para aliviar a carga de trabalho do programador-chefe, existe um segundo membro no núcleo permanente da equipe, o programador-assistente, que deve também estar totalmente familiarizado com a lógica do sistema em desenvolvimento.

Sua principal função é participar, com o programador-chefe, do *design* de programas e na tomada das principais decisões.

Como resultado desta participação, o programador-assistente estará sempre apto a assumir as funções de programador-chefe em qualquer eventualidade e evitar, assim, a interrupção do trabalho da equipe ou o abalo de seus princípio de funcionamento.

Cabe também ao programador-assistente servir como auxiliar na busca de melhores estratégias e táticas de programação, liberando o programador-chefe para a concentração em problemas que dizem respeito mais diretamente com o sistema.

O programador-assistente pode, também, planejar os testes de programas, independentemente.

C.1.3 – Secretário de Programação

O secretário de programação, dentro do CPT, tem como principal função manter atualizados os registros do estado de desenvolvimento do projeto numa Biblioteca de Produção de Programas, da qual falaremos a seguir. Nesta biblioteca, o andamento do projeto se apresenta de forma legível: internamente, pela máquina, e externamente, pelos membros da equipe e outras pessoas. Sua atualização é feita exclusivamente pelo secretário.

Os demais elementos da equipe ficam liberados de certos trabalhos burocráticos, como codificação de cartões de controle, encaminhamento de requerimentos de compilação e testes de programas à Operação, busca e arquivamento de listagens etc., que são atribuições exclusivas do secretário de programação.

C.1.4 – Demais Membros

Os demais membros da equipe são, em geral, programadores. Eles se reportam diretamente ao programador-chefe, que faz a distribuição das tarefas (programas ou módulos de programas) de acordo com as especialidades de cada um. O programador, então, projeta, codifica e testa seu programa. Quando o produto está pronto, leva-o ao programador-chefe para revisão final e integração.

A comunicação do programador com a máquina é feita via secretário de programação e o conteúdo externo da biblioteca pode ser consultado por ele a qualquer momento.

C.2 – BIBLIOTECA DE PRODUÇÃO DE PROGRAMAS

A manutenção do estado de desenvolvimento de forma facilmente acessível é de fundamental importância para a comunicação e o controle do trabalho da equipe na metodologia de *Chief Programmer Teams*.

A Biblioteca de Produção de Programas é um sistema de procedimentos que visa não só a suprir esta exigência como a facilitar todo o desenvolvimento da programação em equipe.

Fisicamente, este elemento da metodologia subdivide-se em duas partes:

a) *biblioteca interna* – conjunto de arquivos em disco magnético, que guarda todos os elementos do projeto legíveis pela máquina, como o código-fonte, o código-objeto, os dados de teste etc.;

b) *biblioteca externa* – conjunto de arquivos que centraliza de maneira padronizada, todos os resultados correntes e históricos do trabalho da equipe. Esta biblioteca é acessível a todos os membros do time.

Correspondendo a cada um deste elementos, existe uma série de procedimentos necessários à sua manutenção, divididos em dois grupos:

a) *procedimentos internos*, que são as *procedures* catalogadas, que inicializam a biblioteca interna e fazem sua manutenção, compilação, teste e armazenamento;

b) *procedimentos externos*, que são um conjunto de normas burocráticas usadas pelos secretário de programação para chamar os procedimentos internos, preparar as entradas em máquina, arquivar as listagens, memorandos, relatórios de acompanhamento, alterações na especificação do sistema etc.

A manutenção da biblioteca (interna e externamente) é tarefa excecutada pelo secretário de programação, em nome da equipe. Qualquer membro escreve o código-fonte de seu programa em folhas de codificação e entrega-as ao secretário. Este insere os comandos de controle necessários e coloca o programa na biblioteca interna. As listagens são arquivadas na biblioteca externa, sendo propriedade da equipe e não do autor do programa. Mesmo as listagens de testes que contêm erros de lógica ou de sintaxe são arquivadas. Se o programador quizer fazer adições ou alterações em seu código, marca-as simplesmente nas páginas das listagens da biblioteca externa e solicita ao secretário as providências necessárias para as mudanças correspondentes na biblioteca interna. A Fig. C.2 mostra o fluxo de informação entre os vários elementos da metodologia.

C.3 – PROGRAMAÇÃO ESTRUTURADA

As vantagens que a Programação Estruturada apresenta como uma técnica de desenvolvimento de programas são aproveitadas substancialmente dentro da metodologia de *Chief Programmer Teams*.

Por exemplo, o desenvolvimento de programas por refinamentos sucessivos permite a distribuição disciplinada das tarefas entre os membros da equipe. O programador-chefe usa a Programação Estruturada para desenvolver os níveis mais altos dos programas principais e delega tarefas dos níveis mais baixos aos demais membros da equipe. Ele deve exigir o uso da Programação Estruturada por parte de todo o grupo e, ainda mais, padronizar certas características externas dos programas, como a forma do código-fonte, o uso de comentários, a escolha dos nomes de variáveis etc. Isto é necessário para assegurar a padronização do conteúdo da Biblioteca de Produção de Programas e, assim, facilitar a comunicação dentro da equipe.

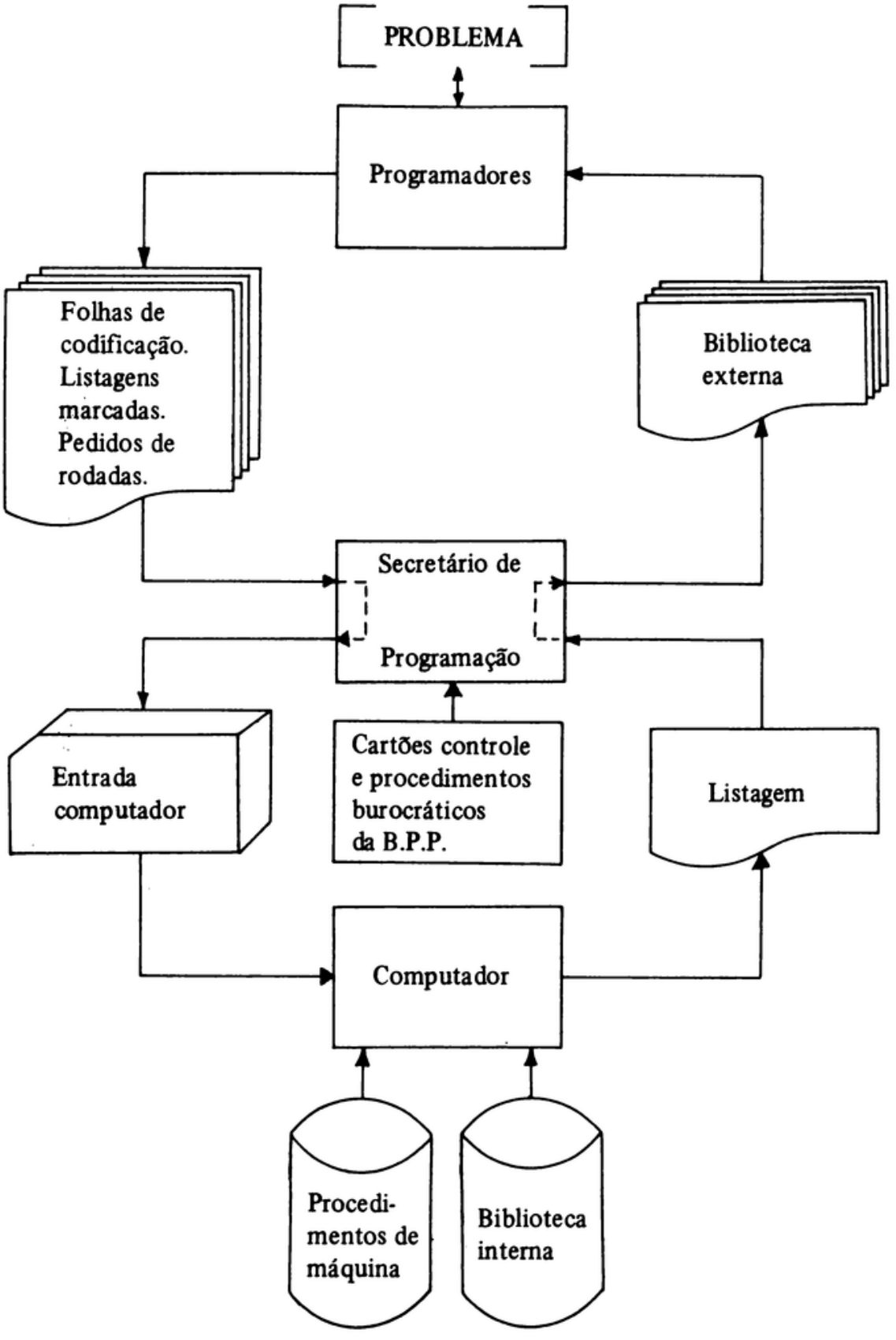


Fig. C.2. Extraída de [1] e [4].

C.4 – ALGUNS BENEFÍCIOS DE ORDEM GERENCIAL

C.4.1 – A Dupla Função do Programador-Chefe

Como foi visto na Seç. C.1, o programador-chefe acumula duas funções num CPT: gerente e técnico.

Pode parecer, à primeira vista, que este acúmulo de funções seja conflitante. Todavia, ele tem sido considerado uma boa solução organizacional para o problema de obtenção de alta produtividade da equipe.

Na realidade, a necessidade do trabalho em equipe tem exigido, em muitas profissões, que certos elementos assumam posição de gerência sem se desligar de suas funções técnicas. Citemos alguns exemplos: numa equipe cirúrgica, o chefe é um cirurgião que participa das operações; numa aeronave, o comandante gerencia uma equipe de profissionais com uma estrutura organizacional bem definida, ao mesmo tempo que executa tarefas técnicas; numa agência de publicidade, o diretor de arte é um elemento que participa ativamente do trabalho criativo concomitantemente com a gerência de uma equipe.

Na profissão de programador, isto também é indicado. E a metodologia de *Chief Programmer Teams* provê a equipe de programação com uma estrutura organizacional que facilita as atividades do programador-chefe como gerente e como técnico.

C.4.2 – Equipe de Especialistas

Uma característica importante da metodologia é que um CPT se constitui numa equipe de especialistas. A distribuição das funções permite a definição clara dos perfis profissionais dos vários membros.

O programador-assistente é um elemento criativo que participa, juntamente com o programador-chefe, da tomada de decisões.

Já o secretário de programação trabalha em nível de manipulação de dados e tarefas burocráticas.

Os demais programadores podem ser admitidos de acordo com suas especialidades. Por exemplo, um programa de atualização de arquivos dentro do sistema será atribuído, provavelmente, àquele programador que demonstrar maior talento para este tipo de serviço.

C.4.3 – Vantagens da Biblioteca de Produção de Programas

O uso da Biblioteca de Produção de Programas contribui decisivamente para o aumento da produtividade e a melhoria da gerência de um CPT.

Primeiramente, a separação entre o trabalho burocrático e trabalho criativo, que é possível graças à série de serviços prestados pelo secretário aos programadores, possibilita a estes melhor aproveitamento de seu tempo.

Uma outra característica importante é a visibilidade que a biblioteca proporciona, tanto do estado corrente como do histórico do projeto. Durante o desenvolvimento da programação, ela mostra com clareza as interligações que existem entre os módulos de programas e entre os programadores; após o desenvolvimento dos programas ela se constitui numa documentação clara da fase de programação e isto é muito importante para a fase de manutenção.

Finalmente, a centralização das operações e da documentação na Biblioteca de Produção de Programas faz dela um pólo de comunicação do time.

C.4.4 – A Comunicação num CPT

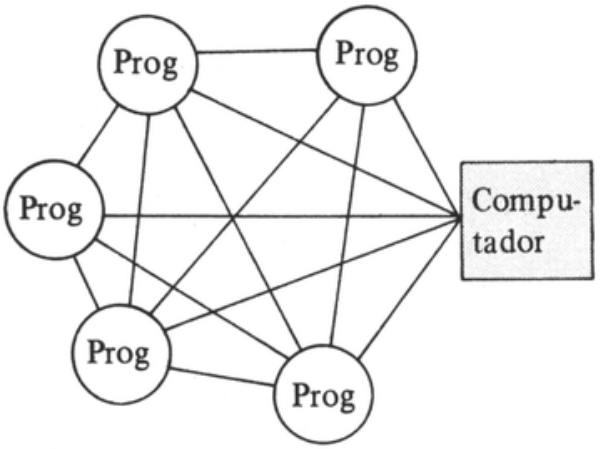
Um fator importante para a coordenação da equipe é a forma de comunicação entre seus elementos. Neste particular, a forma usada pelo CPT beneficia sobremaneira o planejamento e o controle do desenvolvimento da programação.

A comunicação dentro do time é feita através de dois pólos: o programador-chefe e a Biblioteca de Produção de Programas.

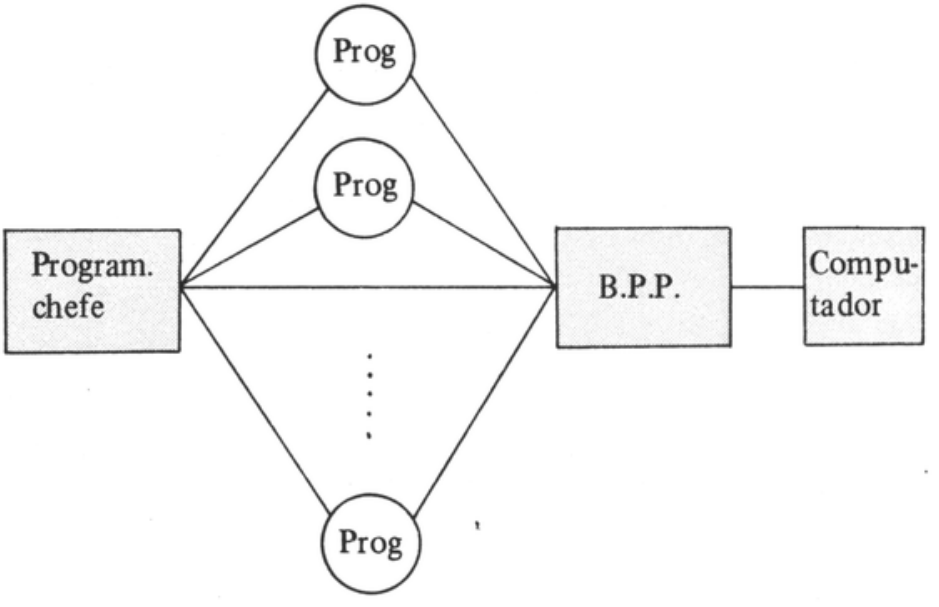
A distribuição das tarefas de programação é realizada exclusivamente pelo programador-chefe, pois é ele quem codifica os módulos de controle dos programas principais.

A Biblioteca de Produção de Programas centraliza a comunicação com a máquina, o trabalho burocrático e a documentação do desenvolvimento. Quando um programador precisa consultar o programa de um colega, ele se dirige à Biblioteca onde este trabalho está arquivado de forma clara e acessível. Isto é possível graças ao princípio de que o produto de um certo membro não é propriedade sua, mas patrimônio da equipe.

A Fig. C.3*b* mostra o grafo de comunicação dentro de um CPT em comparação com o de uma equipe tradicional, que é apresentado no item *a* da mesma figura.



(*a*) Grafo de comunicação em uma equipe tradicional de programação



(*b*) Grafo de comunicação num CPT

Fig. C.3

A simplicidade da comunicação permite ao programador-chefe exercer com maior eficiência o controle tanto da programação em si como da produtividade e disciplina da equipe.

C.5 – "STRUCTURED WALK-THROUGHS"

Embora não faça parte obrigatória da metodologia de *Chief Programmer Teams*, a descrição, neste ponto, de uma ferramenta de gerência de projetos chamada *Structured Walk-Throughs* é conveniente, pois seu uso pode trazer vantagens à programação em equipe.

O desenvolvimento da programação de um Sistema de Processamento de Dados exige *revisões* periódicas para análise quantitativa (volume da produção em relação ao plano de trabalho) e qualitativa (controle da qualidade). Em geral, tiramos pouco proveito dessas revisões, pois elas se caracterizam mais por um conflito entre *revisor* e *revisado*. Este último não vê vantagem neste encontro, encarando-o como uma *prestação de contas* que lhe é imposta e cujo resultado servirá como avaliação direta de seu desempenho. Com a preocupação voltada para aspectos pessoais, o *revisado* tem uma atitude defensiva durante a reunião, e pouco efeito, em termos de aprendizagem, ela poderá surtir sobre ele. Nestas condições, o projeto perde em qualidade.

As revisões podem ser convertidas em sessões altamente produtivas através do uso dos *Structured Walk-Throughs*. Esta nova ferramenta explora muito bem as características da metodologia de *Chief Programmer Teams* e da técnica de Programação Estruturada.

C.5.1 – O Que é "Structured Walk-Through"

O *Structured Walk-Through* (SWT) constitui-se numa série de revisões, cada uma com diferentes objetivos e ocorrendo em tempos distintos no ciclo de desenvolvimento da aplicação.

As características básicas das sessões são:

a) sua convocação é feita pela pessoa (revisado) que desenvolveu o trabalho a ser examinado;

b) os resultados da reunião não são utilizados como base para avaliação do empregado. Isto exclui a participação de gerentes não-técnicos;

c) o material a ser revisado é distribuído aos demais participantes do SWT (revisores) antes da sessão, de modo que os mesmos possam familiarizar-se com seu conteúdo;

d) a ênfase é dada mais à detecção do que à correção de erros;

e) todos os membros técnicos do time têm seu trabalho revisado, inclusive o programador-chefe.

C.5.2 – Como Funciona um "Structured Walk-Through"

O mecanismo básico do SWT é o mesmo para os mais diferentes estágios do projeto. O revisado é responsável por convocar a reunião, selecionar seus participantes, distribuir o trabalho a ser revisado, estabelecer os objetivos do *walk-through* e especificar as normas que os revisores devem seguir.

A seleção dos participantes depende dos objetivos da reunião. Em se tratando de programação, os objetivos, em geral, são a detecção de erros e dos desvios em relação às especificações.

Neste caso, os demais companheiros da equipe devem ser convocados. Todavia, há casos em que até um representante do usuário pode estar presente.

Em geral, o número de participantes varia de quatro a seis pessoas e a duração da reunião é de uma a duas horas.

Os erros e desvios observados no trabalho em revisão são registrados por um secretário (que pode ser o próprio secretário de programação do *Chief Programmer Team*) numa lista de observações que se constitui num meio de comunicação entre os revisores e numa lista de ações para o revisado.

No início da reunião, o secretário recebe dos revisores as primeiras anotações sobre erros encontrados no trabalho durante suas leituras prévias. Embora esta prática leve naturalmente à detecção de pequenos erros de sintaxe, a atenção dos participantes deve estar mais voltada para problemas maiores como falhas no *design*, lógica pobre, técnicas de codificação não apropriadas ou ineficientes etc.

A reunião prossegue com um comentário feito pelos revisores sobre o acabamento, cuidado e qualidade geral do trabalho produzido, identificando-se as áreas de interesse para o prosseguimento. A seguir, o revisado dá uma visão geral do trabalho e depois ele *conduz* os revisores através do mesmo, passo-a-passo, simulando a função sob investigação. Durante essa execução manual, os pontos citados no início da reunião são debatidos em maiores detalhes, bem como novos problemas são levantados para discussão. Os fatores que requerem ações são anotados.

Ao final da reunião, o secretário distribui cópias manuscritas da lista de ações a todos os presentes, cabendo ao revisado garantir que as providências solicitadas na lista serão tomadas.

O gerente não faz uso desta lista, nem para efeito de futura "cobrança" das providências nem para avaliação do funcionário.

Uma prática recomendável é o uso dos SWT para o perfeito desenvolvimento, em paralelo, do programa e de seus casos de teste. O programador-assistente, em geral, é o encarregado de criar os casos de teste. Ele deve, então, submeter o revisado aos seus casos de teste durante a sessão e observar, juntamente com os demais participantes, os erros e inconsistências. Isto serve para o desenvolvimento parcelado de uma biblioteca de casos de testes, que servirá para o teste da versão final do(s) programa(s).

C.5.3 – Como são Usadas Certas Características da Programação Estruturada e dos "Chief Programmer Teams"

Certas características da Programação Estruturada e dos *Chief Programmer Teams* são muito bem exploradas nos *Structured Walk-Throughs*. Citemos algumas.

a) As qualidades de clareza e boa documentação dos programas estruturados fazem-se sentir nas reuniões, pois os revisores devem ler e compreender um programa escrito por outra pessoa – isto seria tremendamente difícil com programas desenvolvidos de maneira tradicional.

b) O secretário de programação do CPT é a pessoa indicada para secretariar os *Structured Walk-Throughs*, devendo, inclusive, ser treinado também neste sentido.

c) A Biblioteca de Produção de Programas é um pólo de comunicação indireta entre os elementos de um CPT. Todos os programas são ali arquivados e acessíveis a todos os membros do time. Isto não só torna mais fácil o debate direto sobre um programa, nos *Structured Walk-Throughs*, como também facilita a observação das providências após a reunião.

C.5.4 – Quais as Vantagens dos "Structured Walk-Throughs"

As vantagens que o uso dos SWT proporciona dentro de um CPT podem ser apresentadas segundo dois aspectos:

a) psicológico;
b) gerencial.

Sob o ponto de vista psicológico, certas vantagens são bem aparentes. A ausência do gerente e a certeza de que a sessão não objetiva avaliar o desempenho funcional, coloca os participantes mais à vontade, numa atitude aberta e não defensiva que propicia maior produtividade. O revisado encara a reunião como uma oportunidade de aperfeiçoamento de seu trabalho, onde os colegas devem ajudá-lo a localizar erros e não a criticá-los.

O programador-chefe e o programador-assistente são os primeiros a terem seu trabalho revisado, pois eles são encarregados do desenvolvimento dos módulos principais do sistema. Isto apresenta vantagens importantes sob o aspecto psicológico: o fato de ser posto em debate, em primeiro lugar, o trabalho destes dois elementos serve para afastar o receio, porventura ainda existente em outros membros da equipe, sobre os objetivos de tais sessões; esta primeira reunião se constitui numa experiência de aprendizado pelo grupo sobre a própria mecânica do *walk-through* – as atitudes e os mecanismos estabelecidos por esses dois elementos mais experientes tenderão a se propagar pelos próximos encontros; ao revisarem o trabalho do programador-chefe e do assistente, os demais membros do time estão tendo uma boa oportunidade de aprender sobre o sistema.

Sob o aspecto gerencial, as vantagens não são tão aparentes, mas existem. O padrão de comunicação num CPT, através de dois pólos centralizadores (Seç. C.4.4), é mudado, temporariamente, nestas reuniões o que se constitui em fator de maior unidade na programação. Por outro lado, os custos tendem a diminuir, pois a localização antecipada dos erros diminui a extensão da fase de depuração. Além disso, os SWT proporcionam a sincronização da evolução dos programas e de seus casos de teste em cada passo do desenvolvimento.

Bibliografia

PARTE I

1. BAKER, F. T. Chief programmer team management of production programming. *IBM Systems Journal, 11* (1): 56-73, 1972.

2. BAKER, F. T. & MILLS, H. D. Chief programmer teams. *Datamation, 19* (12): 58-61, Dec. 1973.

3. BARTH, C. W. Notes on the *case* statement. *Software – Practice and Experience, 4* (3): 289-98, 1974.

4. BOHM, C. & JACOPINI, G. Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the ACM, 9* (5): 366-71, May 1966.

5. BROWN, W. S. Software portability. In: BUXTON, J. N. & RANDELL, B., ed. *Software engineering techniques.*

6. CARDOSO, A. P. *Programação Estruturada, equipes de programação e COBOL*. Tese de Mestrado. PUC-RJ, Dez. 1975.

7. CHAPIN, N. New format for flowcharts. *Software – Practice and Experience, 4* (4): 341-57, 1974.

8. CONWAY, R. W. & GRIES, D. *An introduction to programming – a structured approach using PL/I and PL/C*. Cambridge, Massachusetts, Winthrop, 1973.

9. DENNIS, J. B. Semantics of languages and systems. In: SYMPOSIUM ON THE HIGH COSTS OF SOFTWARE, Monterey, California, 1973. Menlo Park, California, Stanford Research Institute, 1974. pp. 53-80.

10. DIJKSTRA, E. W. GO TO statement considered harmful. *Communications of the ACM, 11* (3): 147-8, Mar. 1968.

11. _____. Notes on structured programming. In: DAHL, O. J. et alii. *Structured programming*. London, Academic Press, 1972. Cap. 1, pp. 1-82.

12. _____. *Programming methodologies, their objectives and their nature*. Nuenen, Netherlands, Burroughs, 1974. EWD469-0.

13. _____. The humble programmer. *Communications of the ACM, 15* (10): 859-66, Oct. 1972.

14. HOARE, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM, 12* (10): 576-83, Oct. 1969.

15. KERNIGHAN, B. W. & PLAUGER, P. J. *The elements of programming style*. Murray Hill, New Jersey, Bell Telephone Laboratories, 1974.

16. KNUTH, D. E. Structured programming with GO TO statements. *Computing Surveys, 6* (4): 261-301, Dec. 1974.

17. LONDON, R. L. A view of program verification. In: INTERNATIONAL CONFERENCE ON RELIABLE SOFTWARE, Los Angeles, 1975. New York, ACM, 1975. p. 534-45.

18. MILLS, H. D. On the development of large reliable programs. In: IEEE SYMPOSIUM ON COMPUTER SOFTWARE RELIABILITY, New York, 1973. p. 155-9.

19. _____. Top down programming in large systems. In: RUSTIN, R., ed. *Debbuging Techniques in large systems*. Englewood Cliffs, New Jersey, 1971.

20. NASSI, I. & SHNEIDERMAN, B. Flowchart techniques for structured programming. *Sigplan Notices, 8* (8): 12-26, Aug. 1973.

21. STEVENS, W. P. *et alii*. Structured design. *IBM Systems Journal, 13* (2): 115-39, June 1974.

22. TSICHRITZIS, D. Reliability. In: BAUER, F. L., ed. *Advanced Course on Software Engineering*. Berlin, Springer Verlag, 1973. p. 319-31.

23. WIRTH, N. On the development of well-structured programs. *Computing Surveys, 6* (4): 247-59, Dec. 1974.

24. _____. *Systematic programming: an introduction*. Englewood-Cliffs, New Jersey, Prentice-Hall, 1973.

25. _____. The programming language PASCAL. *Acta Informatica, 1* (1): 35-63, 1971.

26. YOURDON, E. Making the move to structured programming. *Datamation, 21* (1): 52-6, June 1975.

27. ZAHN JR., C. T. A control statement for natural top-down structured programming. In: PROGRAMMING LANGUAGES SYMPOSIUM. Berlin, Springer Verlag, 1975.

PARTE II

1. BURROUGHS. *B6700/B7700 COBOL reference manual*. 5000656.

2. BUTTERWORTH, R. A. Structured Symbols. *Datamation, 21* (4): 79-83, April 1975.

3. CARDOSO, A. P. *Programação Estruturada, equipes de programação e COBOL*. Tese de Mestrado. PUC-RJ, Dez. 1975.

4. COOPER, R. H. *et alii. File processing with COBOL/WATBOL – a strutuctured approach*. Waterloo, Ontario, WATFAC, 1973.

5. IBM. *OS COBOL (E, F) language*. GC28-6516.

6. IBM. *OS COBOL (F) programmer's guide*. GC28-6380.

7. IBM. *OS full American National Standard COBOL*. GC28-6396.

8. IBM. Introduction to structured programming using COBOL. In: EXXON. *Program Structure Technology*. 1973.

9. McCLURE, C. L. Structured Programming in COBOL. *Sigplan Notices, 10* (4): 25-33, April 1975.

10. NICHOLLS, J. E. *The structure and design of programming languages*. Menlo Park, Massachusetts, Addison-Wesley, 1975.

APÊNDICE C

1. BAKER, F. T. Chief programmer team management of production programming. *IBM Systems Journal, 11* (1): 56-73, 1972.

2. _____. Structured programming in a production programming environment. In: INTERNATIONAL CONFERENCE ON RELIABLE SOFTWARE, Los Angeles, 1975. New York, ACM, 1975. p. 172-85.

3. _____. System quality through structured programming. In: FALL JOINT COMPUTER CONFERENCE, S. L., Afips, 1972. p. 339-43.

4. BAKER, F. T. & MILLS, H. D. Chief programmer teams. *Datamation, 19* (12): 58-61, Dec. 1973.

5. IBM. *Chief programmer teams principles and procedures*. Gainnersburg, Maryland, Federal Systems Division, 1971. FSD-5106.

6. IBM. *Structured walk-throughs: a project management tool*. S. L., Data Processing Division, 1973.

Terminou-se de imprimir este livro no décimo segundo mês do ano de 1981, nas oficinas da Graphos Industrial Gráfica Ltda., para LTC – LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A., na cidade de São Sebastião do Rio de Janeiro.

ANIBAL P. CARDOSO

Programação Estruturada em Cobol

O tema central deste livro é a *Programação Estruturada*.

Esta técnica, que tem despertado no Brasil, a exemplo do que ocorre em outros países, grande interesse por parte da comunidade de Processamento de Dados, é tratada nesta obra através da seguinte abordagem:

1.º – faz-se uma descrição fiel dos fundamentos básicos da técnica, independentemente da linguagem de programação usada pelo leitor;

2.º – com base nesta fundamentação, é sugerida uma forma de implementar a Programação Estruturada em COBOL.

Com isto, o livro é capaz de atender a dois grupos de leitores com interesses específicos: primeiramente, aos professores e estudantes dos diversos cursos de graduação e pós-graduação na área de Processamento de Dados, que encontrarão nesta obra um material da maior importância para subsidiar as diversas disciplinas que ensinam a Programação Estruturada. Em segundo lugar, aos profissionais de Programação e aos CPDs em geral, interessados em usar a técnica em linguagem COBOL, a mais difundida, hoje em dia, no processamento comercial.

O texto reflete o compromisso assumido pelo autor em apresentar os vários conteúdos de forma didática, sem se afastar do rigor e da fidelidade com que os conceitos de Programação Estruturada devem ser tratados.

Sobre o Autor

O Prof. Aníbal Pereira Cardoso é graduado em Matemática pela Faculdade Porto-Alegrense de Educação, Ciências e Letras e Mestre em Informática pela PUC/RJ.

É professor do Instituto de Informática da PUC/RS e do Departamento de Computação da Universidade do Vale do Rio dos Sinos (UNISINOS), RS.

É Assessor Técnico para assuntos de Informática da Caixa Econômica do Estado do Rio Grande do Sul.

Tem lecionado Programação Estruturada em várias disciplinas de cursos de graduação e pós-graduação e tem apresentado palestras e seminários sobre Programação Estruturada.

MAIS UM LANÇAMENTO
DA
LTC LIVROS TÉCNICOS E CIENTÍFICOS EDITORA

ISBN: 85-216-0163-8