METODOLOGIAS DE PROGRAMAÇÃO

1. GENERALIDADES

Todo programa a ser elaborado deve ser considerado como um produto a ser desenvolvido. Para tal, uma série de etapas devem ser cumpridas até a obtenção do programa final, testado e aprovado.

Muitos programadores inexperientes gastam a maior parte do tempo na codificação do programa, e o restante, na depuração e testes. Na verdade, a codificação é apenas uma das etapas, ocupando em torno de 20% do tempo total consumido.

O intuito deste capítulo é apresentar um resumo das principais metodologias de programação, fornecendo uma orientação básica no sentido de disciplinar o desenvolvimento de programas.

2. ETAPAS DE DESENVOLVIMENTO DE SOFTWARE

As etapas do desenvolvimento de software são:

- 1ª etapa: Definição do problema
- 2ª etapa: Projeto do programa
- 3ª etapa: Codificação
- 4ª etapa: Depuração
- 5ª etapa: Testes
- 6ª etapa: Documentação
- 7ª etapa: Manutenção e reprojeto

A figura 2.1 apresenta um fluxograma das etapas do desenvolvimento de software.

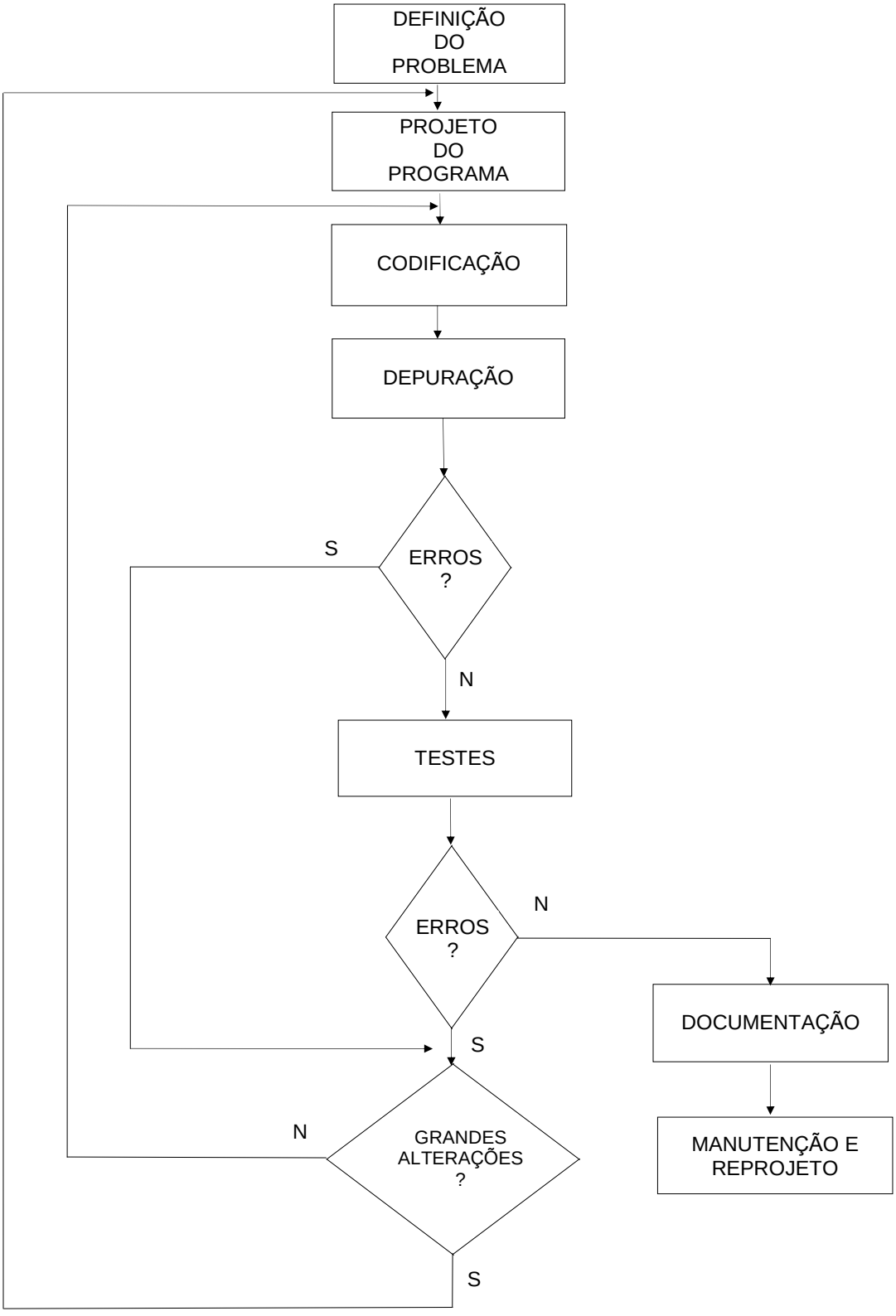


Figura 2.1: Etapas do Desenvolvimento de Software.

A primeira etapa é a DEFINIÇÃO DO PROBLEMA. Todas as tarefas desejadas deverão ser especificadas, tais como:

- definição das entradas: tipos, formas e taxas de aquisição, inter-relações com as saídas, escalas, etc.

- definição das saídas: tipos, taxas de atualização, inter-relações com as entradas, escalas, etc.

- processamento de dados: algoritmos, limitações de velocidade e memória, base de dados, precisões, etc.

- tratamento de erros: tipos que podem ocorrer, tratamento previsto, influência no comportamento do sistema, diagnóstico, reconfigurações, etc.

- interface homem-máquina: formas de comunicação com o operador, procedimentos automáticos, manuais e interativos, facilidades de operação, etc.

A segunda etapa é o PROJETO DO PROGRAMA. Muitas técnicas disponíveis podem ser utilizadas para sistematizar a especificação do programa. É conveniente a descrição das tarefas a serem desempenhadas de maneira a facilitar suas transcrições para programas. Algumas técnicas úteis normalmente usados são:

- fluxogramas

- programação estruturada

- programação modular

- projeto "top-down".

A terceira etapa é CODIFICAÇÃO. Esta torna-se uma tarefa difícil e demorada se as duas etapas anteriores não forem adequadamente cumpridas. Trata-se da escrita de um programa em uma forma que possa ser entendida pelo microprocessador, ou que possa ser traduzido para essa forma, através de montadores ou compiladores.

A quarta etapa é a DEPURAÇÃO. A quinta etapa são os TESTES.

Em linhas gerais, tem-se que a DEPURAÇÃO procura suprimir todos os erros de programa, enquanto que os TESTES, procuram validar o programa, verificando se as tarefas previstas estão sendo desempenhadas a bom termo.

A sexta etapa é a DOCUMENTAÇÃO, nem sempre é tratada com a real importância que deveria ser. Muitos programadores procuram erroneamente deixá-la como última etapa, acarretando problemas, como por exemplo, esquecimentos de detalhes, comentários errados e explicações superficiais.

Na verdade, a DOCUMENTAÇÃO deve começar a ser gerada desde a primeira etapa, descrevendo todos os passos percorridos até a obtenção de um programa testado e aceito.

Nesta etapa, todos os documentos gerados devem ser conferidos e combinados de forma coesa, devendo ser dirigido aqueles que não participaram dos trabalhos, e que com a leitura deste, poderiam compreender o que foi desenvolvido.

A sétima etapa é a de MANUTENÇÃO e REPROJETO do que foi desenvolvido. Correções de pequenos erros, aprimoramentos e ampliação de programas são fatos comuns que deve ser cuidadosamente estudados.

Cada alteração em um programa deve ser considerada como um novo programa, sendo submetida a todos os estágios anteriores. Desta forma, evita-se remendos, efeitos colaterais e dificuldades maiores nas alterações.

Uma boa documentação será de real utilidade nesta etapa, e deve ser adequadamente atualizada.

3. PROJETO DO PROGRAMA

O PROJETO DO PROGRAMA deve ser elaborado inspirado em alguma metodologia adequadamente escolhida, evitando-se um gasto maior de tempo, dificuldades excessivas em programas de maior porte e construções de estruturas confusas e inflexíveis.

Alguns princípios básicos devem ser observados em cada projeto, tais como:

- divisão do problema em módulos curtos, logicamente divididos, e de mais simples resolução

- elaboração de estruturas de programas simples e de fácil controle

- à medida do possível, utilizar representações gráficas da lógica do programa

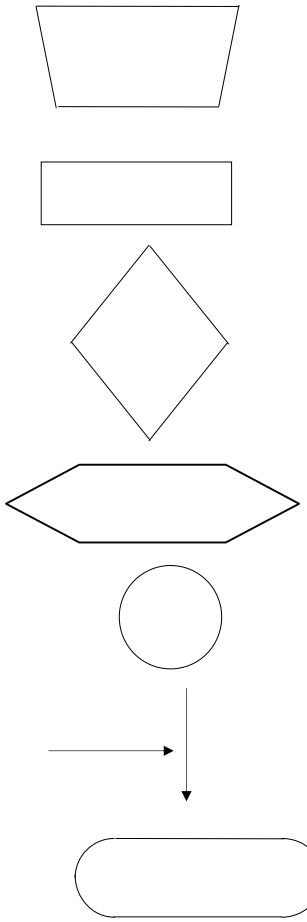
- não elaboração de programas difíceis de serem depurados e alterados

- iniciar a codificação apenas após todo o programa ter sido projetado.

Serão apresentadas a seguir 4 técnicas de PROJETO DE PROGRAMAS comumente utilizadas:
- fluxograma
- programação modular
- programação estruturada
- projeto “Top-Down”

3.1 Fluxogramas

Trata-se de um dos mais conhecidos e utilizados métodos. Utiliza-se de alguns símbolos que permitem uma representação gráfica da estrutura do programa. Os símbolos utilizados são:



ENTRADA/SAÍDA

OPERAÇÕES DE PROCESSAMENTO

DECISÕES LÓGICAS

SUBROTINAS

PONTO DE CONEXÃO

INTERLIGAÇÃO

INÍCIO ou FIM

As principais vantagens dos Fluxogramas são:

- uso de símbolos padronizados e bem conhecidos
- de fácil entendimento por não especialistas
- facilidade de divisão das tarefas em subtarefas
- facilitam a localização de erros por apresentarem a seqüência das operações
- muito conhecidos em diversas áreas de atividades.

As principais desvantagens dos Fluxogramas são:

- apresentam dificuldades no desenho e alterações
- depuração e testes não são fáceis, em geral
- dificuldade na decisão do nível de detalhes a serem colocados nos fluxogramas
- não apresentam a organização de dados e estruturas de entrada e saída
- não facilitam em casos particulares de hardware ou "timing".

Verificar nas figuras 3.1, 3.2 e 3.3, alguns exemplos de utilização de fluxogramas

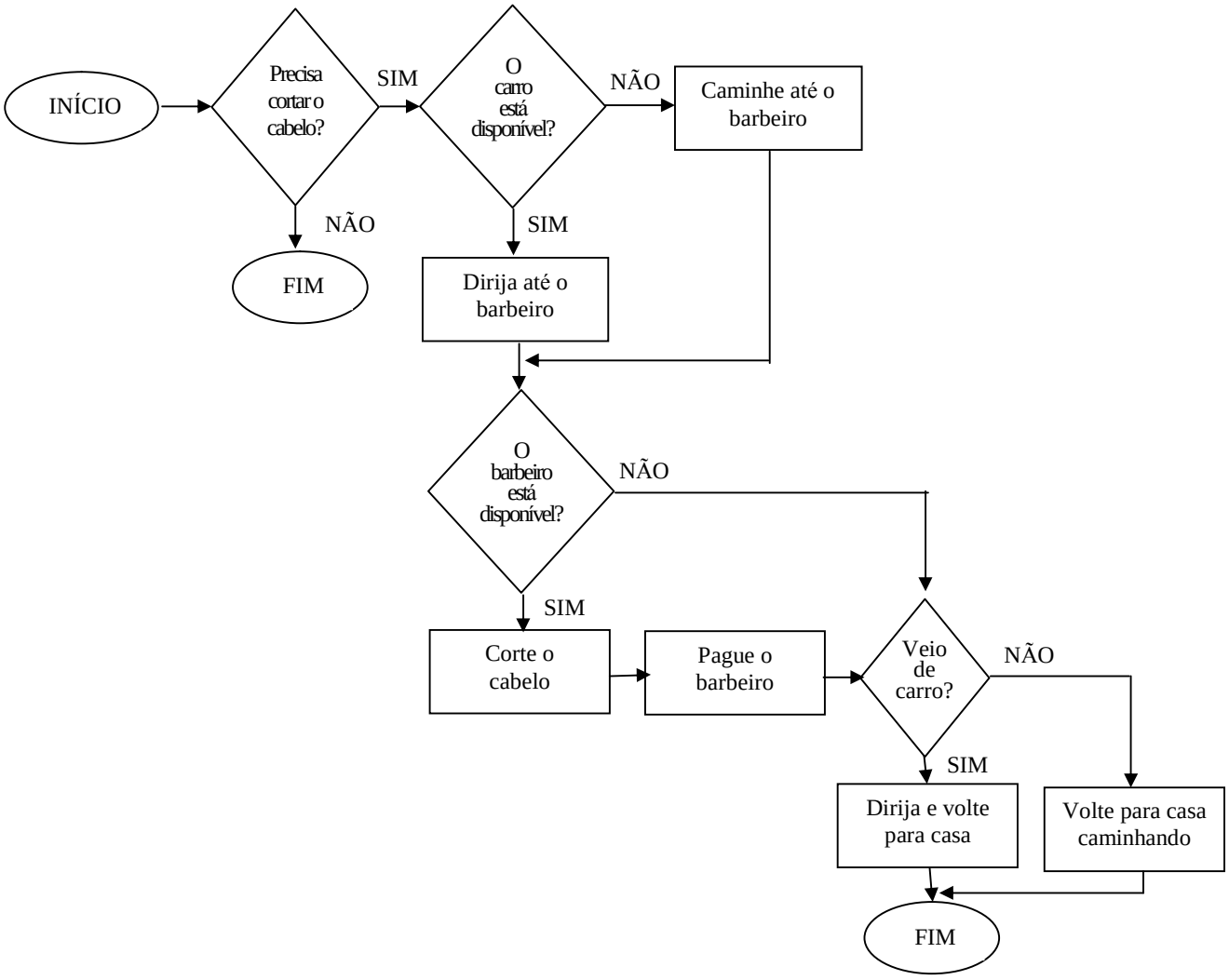


Figura 3.1: Exemplo de Fluxograma para uma pessoa que precisa cortar o cabelo

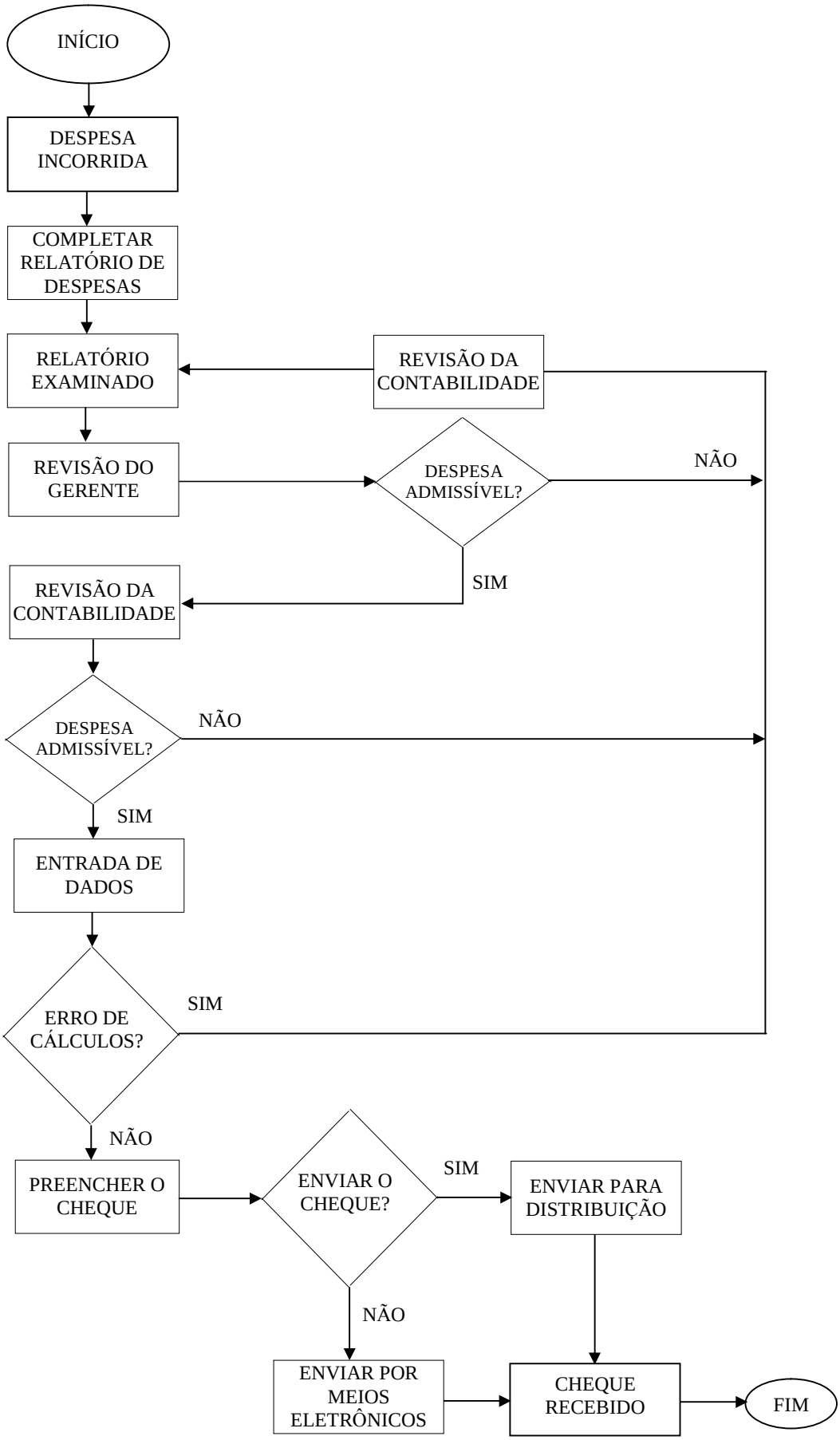


Figura 3.2: Exemplo de Fluxograma para o Processo de Reembolso de Despesa

Exemplo: Apresentar um Fluxograma para quando uma chave for ligada, uma lâmpada deverá permanecer acesa por um segundo.

Uma dificuldade que costuma aparecer é o nível de detalhamento necessário. Na figura 3.3 é apresentada uma versão simples, especificando apenas os aspectos fundamentais do programa. Uma versão mais detalhada poderia ser elaborada, utilizando variáveis e com maiores informações nos blocos.

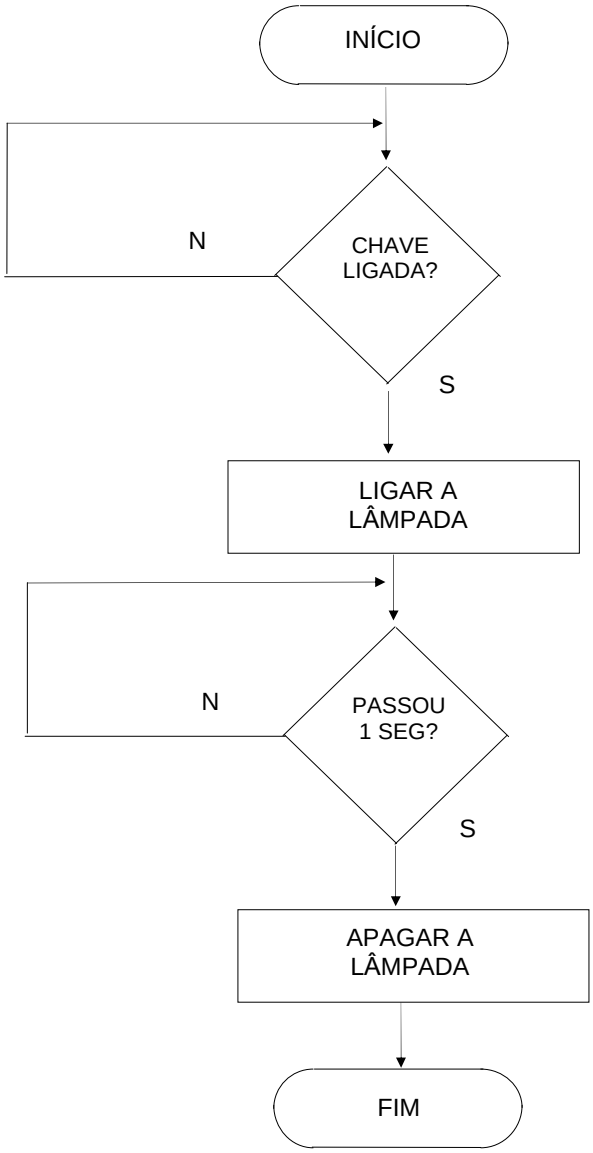


Figura 3.3: Exemplo de Fluxograma para quando uma chave for ligada, uma lâmpada deverá permanecer acesa por um segundo

3.2 Programação Modular

Consiste na divisão de programas em subtarefas de menor complexibilidade, eliminando o inconveniente dos fluxogramas de se tornarem extremamente grandes e complexos, em aplicações de maior porte.

As principais vantagens da Programação Modular são:

- maior facilidade na escrita e depuração dos módulos, por serem menores e mais simples

- os módulos podem ser utilizados diversas vezes em programas e também arquivados em bibliotecas, para uso futuro

- restringe as alterações aos módulos ao invés de todo o programa

- maior facilidade de isolação dos erros

- maior facilidade na quantificação do trabalho a ser/que foi desenvolvido.

As principais desvantagens da Programação Modular são:

- dificuldades na elaboração de critérios objetivos de divisão de módulos

- necessidade de padronização da documentação de módulos, uma vez que eles podem ser elaborados por diferentes programadores

- dificuldade, em alguns casos, em se depurar e testar módulos isoladamente, por necessitarem de outros dados de outros módulos (necessidade de elaboração de programas de testes)

- programas inadequadamente modularizados podem se tornar de difícil manuseio

- eventual necessidade de maior quantidade de memória e tempo de processamento, em função de repetições de partes de módulos separados.

Algumas sugestões básicas podem auxiliar o programador, tais como:

- usar módulos de 20 a 50 linhas (geralmente módulos menores podem não apresentar vantagens, enquanto que módulos maiores, tornam-se de difícil integração)

- criar módulos flexíveis, adaptáveis às mudanças de opções de uma mesma função

- funções básicas tais como atrasos, rotinas aritméticas, etc., devem ser elaboradas com maior cuidado para futuro aproveitamento

- evitar a mistura de funções distintas em um mesmo módulo.

Exemplo: Em relação ao exemplo da Figura 3.3, tem-se a possibilidade de divisão do programa em dois módulos:

- módulo 1: espera uma chave ser ligada e acende uma lâmpada

- módulo 2: espera um segundo.

Enquanto o módulo 1 é específico, o módulo 2 pode ser de uso geral, podendo ser construído de uma maneira mais genérica.

3.3 Programação Estruturada

A Programação Estruturada baseia-se em 4 diretrizes:
- o programa é desenvolvido de cima para baixo
- qualquer programa pode ser escrito utilizando apenas 3 estruturas básicas de controle
- cada estrutura básica de controle tem apenas uma entrada e uma saída
- minimização do uso de desvios incondicionais

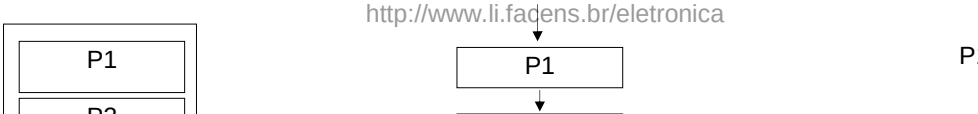
As 3 estruturas básicas de controle, conforme mostra a figura 3.4, e com as quais pode-se construir qualquer programa são:

- estrutura seqüencial

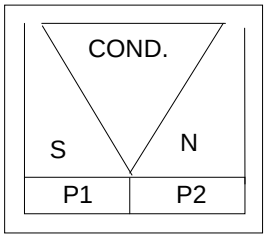
- estrutura de controle condicional

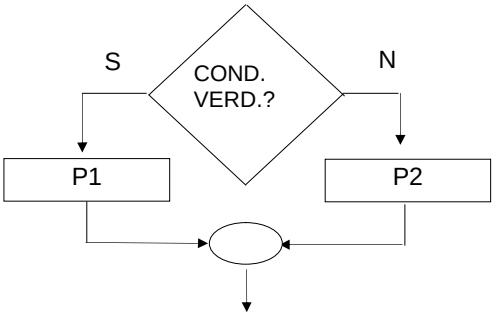
- estrutura de malha ("loop").

ESTRUTURA SEQUENCIAL

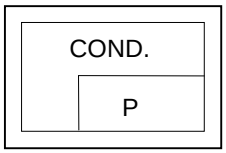


ESTRUTURA DE CONTROLE CONDICIONAL





IF COND. = VERD.
THEN P1
ELSE P2



As principais vantagens da Programação Estruturada são:

- os programas normalmente são claros e bem organizados, facilitando o entendimento

- uso de poucas estruturas padronizadas, facilitando a modularização, a documentação e a descrição com Fluxogramas ou outros métodos

- permite redução no tempo de elaboração de programas.

As principais desvantagens da Programação Estruturada são:

- no caso de linguagem Assembly deve-se fazer conversões das estruturas para as instruções do microprocessador

- elaboração de programas maiores (consome mais memória) e mais lentos

- excessivos encadeamentos de estruturas geram confusão

Sugere-se a utilização de programação estruturada nos seguintes casos:

- grandes programas (mais que 1000 instruções)

- aplicações onde a quantidade de memória utilizada não é crítica, e o custo de programação é importante (pouca demanda de produto)

- aplicações sem estruturas de dados complexas, controle de processos, uso de algoritmos, etc.

- disponibilidade de linguagem de alto nível.

Exemplo: Em relação ao exemplo da Figura 3.3 , tem-se a seguinte versão estruturada:

```
CHAVE = DESLIGADA
do While CHAVE = DESLIGADA
LEIA ESTADO DA CHAVE
end

LÂMPADA = LIGADA
ESPERA 1
LÂMPADA = DESLIGADA
End
```

3.4 Projeto "Top-Down"

Esta técnica sugere que o PROJETO DO PROGRAMA se inicie pela definição da estrutura principal do mesmo, deixando os sub-programas indefinidos temporariamente. Desta forma, pode-se mais facilmente avaliar e testar a estrutura principal de um programa.

Posteriormente, tal procedimento deve ser repetido para os subprogramas, sucessivamente, até o término da elaboração e testes do programa.

As principais vantagens do Projeto “Top-Down” são:

- elaboração e testes a cada nível, sem a necessidade da construção de programas auxiliares de teste

- pressupõe o uso de Programação Modular, e é compatível com Programação Estruturada

- fornece uma idéia clara do que falta ser desenvolvido.

As principais desvantagens do Projeto “Top-Down” são:

- poucas vantagens apresenta em caso da existência de partes de programas prontos

- dificulta a elaboração de módulos mais genéricos

- erros no nível superior costumam apresentar efeitos muito abrangentes.

Exemplo: Em relação ao exemplo da Figura 3.3, tem-se que a estrutura apresentada corresponde à primeira etapa do projeto "top-down". Algumas das declarações devem ser detalhadas:

- LEIA ESTADO DA CHAVE:

```
CHAVE = DADO (PORTA 1) and máscara
leitura de um bit específico de uma porta.
end
```

- ESPERA 1:

```
CONTADOR = VALOR INICIAL
do while CONTADOR  # 0
CONTADOR = CONTADOR - 1
end
```

O programa é expandido :

```
CHAVE = 0
do while CHAVE = 0
CHAVE = DADO (PORTA 1) and máscara
end

LÂMPADA  = LIGADA
CONTADOR = VALOR INICIAL
do while CONTADOR # 0
CONTADOR = CONTADOR - 1
end

LÂMPADA = NOT (LÂMPADA)
```

Nesta forma, o programa já pode ser mais facilmente traduzido em termos de instruções da linguagem utilizada.

4. DEPURAÇÃO

O tempo nesta fase do desenvolvimento de um programa costuma ser considerável. A escolha de um método adequado de programação, por si só já simplifica a depuração e os testes.

Entretanto, é necessário que o programador conte com ferramentas adequadas, e com um roteiro de procura de erros, no sentido de não tornar tal tarefa aleatória.

Serão apresentadas, de forma resumida, as principais ferramentas e metodologias de depuração.

4.1 Execução Passo-a-Passo (Single-Step)

Permite a execução de instrução por instrução de um programa, sob controle do operador. Após uma instrução ter sido executada, os registradores, posições de memória, linhas de saída e sinais de controle podem ser examinados. Apresenta limitações em casos de interrupções, DMA (acesso direto à memória) e tempo de execução de programas.

4.2 Ponto de Parada (Breakpoint)

É um endereço de memória que, quando atingido, provoca a parada do programa, permitindo o mesmo tipo de monitoração conseguida na execução passo-a-passo. Costuma ser implementada com a instrução RST 1 (8080, 8085 e Z-80), a qual é colocada no endereço de parada desejado, e chama uma subrotina que faz a parada do programa.

Amplia as possibilidades da execução passo-a-passo, permitindo interrupções, DMA e não afeta o tempo de execução de trechos de programas.

4.3 Visualização de Registrador (Register Dump)

Apresenta o conteúdo de todos os registradores do microprocessador ao programador, ou de algum selecionado. Utilizado em conjunto com os recursos de execução passo-a-passo e ponto de parada.

4.4 Visualização de Memória (Memory Dump)

Apresenta o conteúdo de uma região de memória especificada. Algumas variações deste recurso podem ser encontradas, tais como a visualização da área de código de um programa, onde é mostrada uma instrução por linha, em hexadecimal ou até mesmo em mnemônicos. Também utilizada em conjunto com os recursos de execução passo-a-passo e ponto de parada.

4.5 Simuladores

Programas elaborados para simular o processo.

4.6 Analisador Lógico

Basicamente é uma versão digital paralela de um osciloscópio.

Mostra informações em binário ou hexadecimal na tela, comandado por diversas formas de disparo. Costumam apresentar memória, registrando fatos passados.

Utilizado basicamente no registro de informações até/após um certo evento particular, na via de dados ou endereços, ou execução de uma instrução de entrada/saída particular.

São efetivamente necessários em sistemas com "timing" complexo.

4.7 Listas de Verificação

Muitos dos erros comumente encontrados pelos programadores poderiam ser eliminados com facilidade, a partir de uma Lista de Verificação. Esta apresenta alguns itens básicos que devem ser conferidos antes do uso de qualquer outra ferramenta.

Alguns dos itens que podem ser incluídos na lista são:

- verificar se cada elemento do projeto do programa se encontra no programa, e vice-versa
- verificar se todos os registradores e posições de memória utilizados em malhas ("loops") são previamente inicializados
- conferir todos os desvios condicionais
- verificar se todas as malhas se iniciam e terminam corretamente
- verificar se todas as condições são tratadas
- verificar se as regras da linguagem não estão sendo desobedecidas, e se as instruções não estão sendo utilizadas erradamente.

Para este último item, uma nova Lista de Verificação pode ser elaborada, particular a cada linguagem de microprocessador.

4.8 Simuladores de Entrada/Saída

4.9 Emuladores

São acoplados a Sistemas de Desenvolvimento permitindo a execução em tempo real, e entradas/saídas programáveis

4.10 Programas de Testes Diversos.

5. TESTES

Procura-se efetuar a validação (verificação que realmente os programas estão depurados) dos programas elaborados, verificando se os mesmos atendem às especificações estabelecidas.

Esta etapa está intimamente ligada à etapa de Depuração. As ferramentas usadas na etapa anterior também poderão ser utilizadas nesta etapa.

6. LISTA DE EXERCÍCIOS DE FLUXOGRAMA

1) Fazer um Fluxograma para ler posições de memória até encontrar um conteúdo igual a 20H.
   Guardar a posição de memória em RESULT quando encontrar o conteúdo. Colocar 0000H em RESULT caso não seja encontrado um conteúdo igual a 20H.
   Ler no máximo o conteúdo de 100H posições de memória.

2) Ler o conteúdo de 1000 posições de memória.
   Guardar em RESULT o valor do maior conteúdo.

3) Fazer o Fluxograma para calcular o fatorial do número N, cujo valor é lido do teclado.

4) Fazer o Fluxograma para ler 50 arquivos contendo, cada um, a altura e o código do sexo de uma pessoa (código = H para homem e M para mulher), calcular:

   - a média de altura dos homens.

5) Fazer o Fluxograma de um algoritmo para fazer a soma de 2 setores de mesmo número de elementos.

6) Fazer o Fluxograma de um algoritmo para calcular o número de alunos que tiraram nota acima da nota média da turma. A turma tem 40 alunos.

7) Dado um arquivo com uma frase de 80 letras (incluindo brancos). Escrever o Fluxograma de um algoritmo para:

   a) contar quantos brancos existem na frase
   b) contar quantas vezes aparece a letra “A”