# Cap. II - ALGORITMOS E PROGRAMAS EM C

## 2.1 - Conceitos de Algoritmo e Programa

- **Algoritmo:** seqüência ordenada e sem ambigüidades de comandos que levam à execução de uma tarefa ou à solução de um problema.

- **Exemplo 2.1:** Troca de um pneu furado

```
Troca_Pneu_Furado {
    Se  (o estepe estiver vazio)
        Chamar o borracheiro;

    Senão {
        Afrouxar todos os parafusos da roda;
        Levantar o carro com o macaco;
        Retirar todos os parafusos da roda;
        Retirar o pneu furado;
        Colocar o estepe;
        Recolocar e apertar ligeiramente os parafusos;
        Baixar o carro com macaco;
        Apertar fortemente os parafusos;
    }
}
```

- **Exemplo 2.2:** Troca de uma lâmpada queimada

```
Troca_Lâmpada_Queimada {
    Se (houver na dispensa lâmpada de mesma potência)
        Pegar a lâmpada;
    Senão
        Comprar lâmpada de mesma potência;
    Posicionar a escada abaixo do bocal da lâmpada;
    Subir na escada até alcançar a lâmpada;
    Girar a lâmpada do bocal no sentido anti-horário
        até soltá-la;
    Posicionar a lâmpada nova no bocal;
    Girá-la no sentido horário até prendê-la;
    Descer da escada.
}
```

- **Exemplo 2.3:** Raízes reais da equação $Ax^2 + Bx + C = 0$

```
Raizes_Eq_2_Grau {
    Ler (A, B, C);  Delta ← B^2 – 4AC;
    Se  (Delta ≥ 0) {
        X1 ← (-B + √Delta) / (2 * A);
        X2 ← (-B – √Delta) / (2 * A);
        Escrever ( “X1 = ”, X1, “ e X2 = ”, X2 );
    }
    Senão Escrever (“Não há raizes reais”);
}
```

- **Exemplo 2.4:** Soma de uma PA sem usar a fórmula

```
Soma_PA  {
        Ler  ( a1, r, n );
        soma ← 0;  aq ← a1;  i ← 1;
        Enquanto  (i ≤ n) {
                soma ← soma + aq;  aq ← aq + r;  i ← i + 1;
        }
        Escrever  (“Progressão aritmética: ”, <muda-linha>,
                “Primeiro termo: ”, a1, “Razão: ”, r,
                “Número de termos: ”, n, “Soma dos termos: ”,
                soma);
}
```

- **Programa:** implementação de um algoritmo numa linguagem de programação

- **Exemplo 2.5:** programa em C para o algoritmo do exemplo 2.3 (equação do 2.o grau)

```
#include<stdio.h>
#include<math.h>

void main() {
        float a, b, c, delta, x1, x2;

        scanf("%f%f%f", &a, &b, &c);
        delta = pow(b,2) - 4*a*c;
        if (delta >= 0) {
                x1 = (-b + sqrt(delta)) / (2*a);
                x2 = (-b - sqrt(delta)) / (2*a);
                printf("x1 = %f  e  x2 = %f", x1, x2);
        }
        else
                printf("nao ha raizes reais");
}
```

- **Exemplo 2.6:** programa em C para o algoritmo do exemplo 2.4 (cálculo da soma da PA)

```
#include <stdio.h>

void main () {
        int r, n, i;
        long a1, aq, soma;

        printf ("Progressao aritmetica\n\n");
        printf ("   Primeiro termo: ");
        scanf ("%ld", &a1);
        printf ("   Razao: ");
        scanf ("%d", &r);
        printf ("   Numero de termos: ");
        scanf ("%d", &n);

        soma = 0; aq = a1; i = 1;
        while (i<=n) {
                soma = soma + aq; aq = aq + r; i = i + 1;
        }
        printf ("\nSOMA DOS TERMOS: %ld", soma);
}
```

## 2.2 - Propriedades de Bons Algoritmos

- **1.a Propriedade:** o tempo de execução deve ser finito para qualquer entrada.

- **Exemplo 2.7:** tempo de execução infinito

**Ler** (n);
**Enquanto** n > 0 n ← n + 1;

Se o valor lido for maior que zero: tempo infinito

- **2.a Propriedade:** os comandos do algoritmo devem ser precisos.

- **Exemplo 2.8:** comandos precisos e imprecisos

  - **Se** ( | x - y | é pequeno) x ← y; → impreciso
  - **Se** ( | x - y | ≤ 0.001) x ← y; → preciso
  - Colocar uma pitada de sal; → impreciso
  - Colocar 5g de sal; → preciso

- **3.a Propriedade:** o algoritmo deve ter pelo menos uma saída de resultados, mas pode ter zero ou mais entradas de dados.

- **4.a Propriedade:** os comandos devem ser executáveis.

- **Exemplo 2.9:** comandos não executáveis
  - **Se** chover amanhã, hoje conserto meu guarda-chuva;
  - **Se** (n > 10) n ← n/2 (O valor de **n** é desconhecido)
- **5.a Propriedade:** o algoritmo deve ser suficientemente

detalhado.

- **Exemplo 2.10:** necessidade de detalhes

  - **Inserir** 'JOSE' na posição **i** de **Lista**; **(**Insuficiência de detalhes)

  - Correção:

**Para** (j ← n; j ≥ i; j ← j - 1)
    Lista [j + 1] ← Lista [j];
Lista [i] ← 'JOSE';

**Obs:** o detalhamento depende da linguagem escolhida

- **6.a Propriedade:** o algoritmo deve ser bem estruturado, legível e de fácil correção.

- Existem metodologias para se tentar garantir essa propriedade:

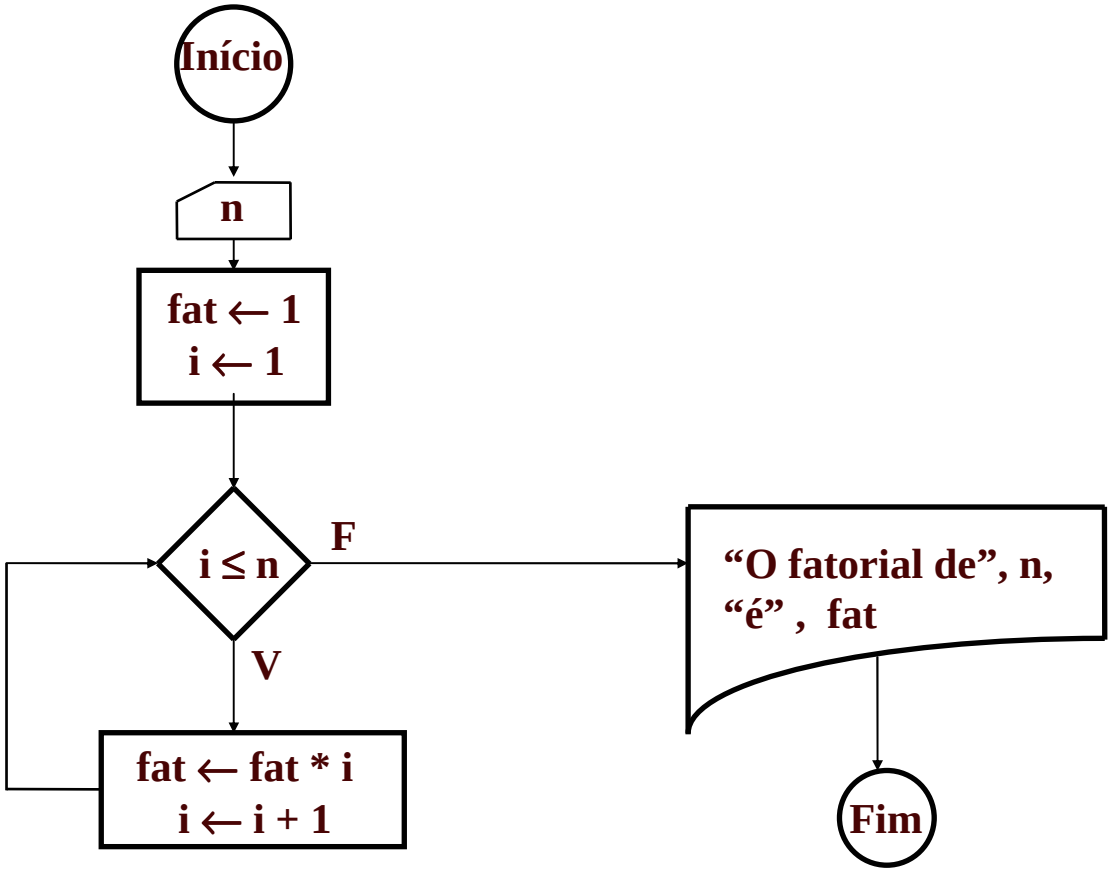
  - programação ***top-down***
  - programação orientada a objetos
  - programação modular
  - programação estruturada

- Nesta matéria, apenas o segundo item acima não será abordado.

## 2.3 - Fluxogramas

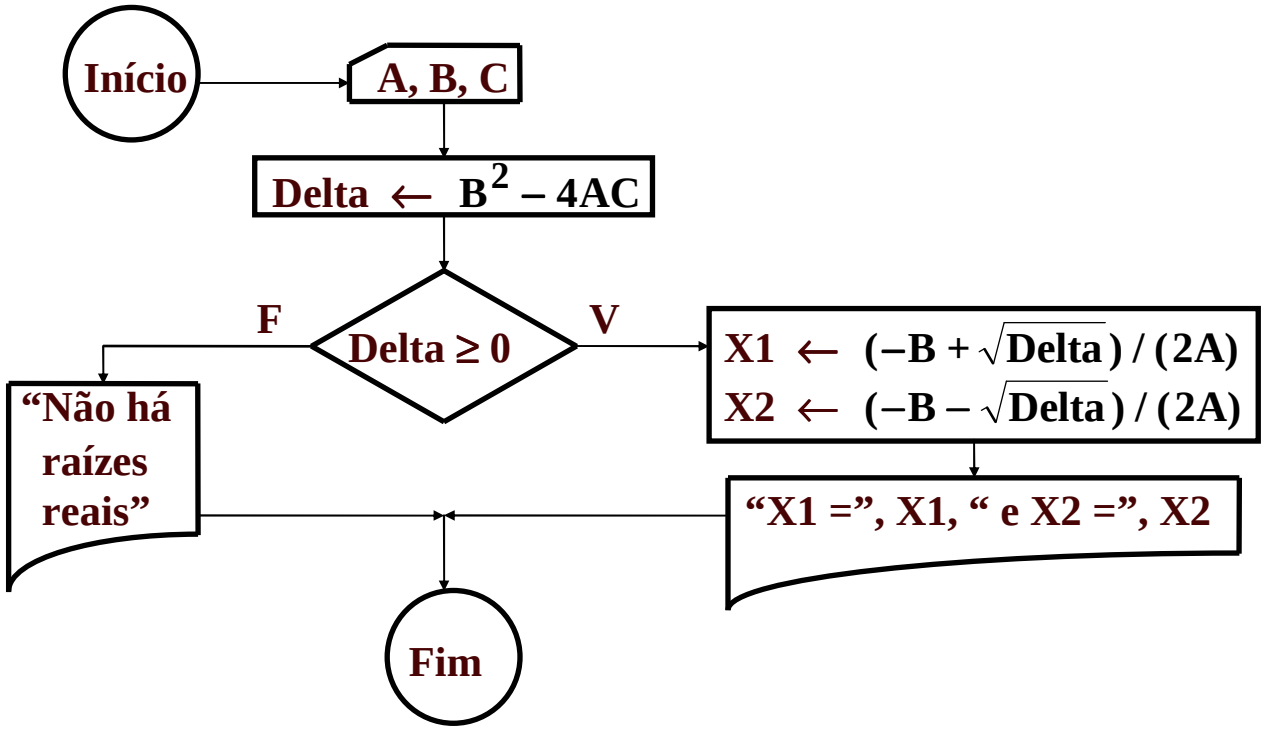
- Algoritmos também podem ser expressos por diagramas de blocos chamados **fluxogramas** (evidenciam o fluxo de execução dos comandos).

- **Exemplo 2.11:** fluxograma para o cálculo do fatorial

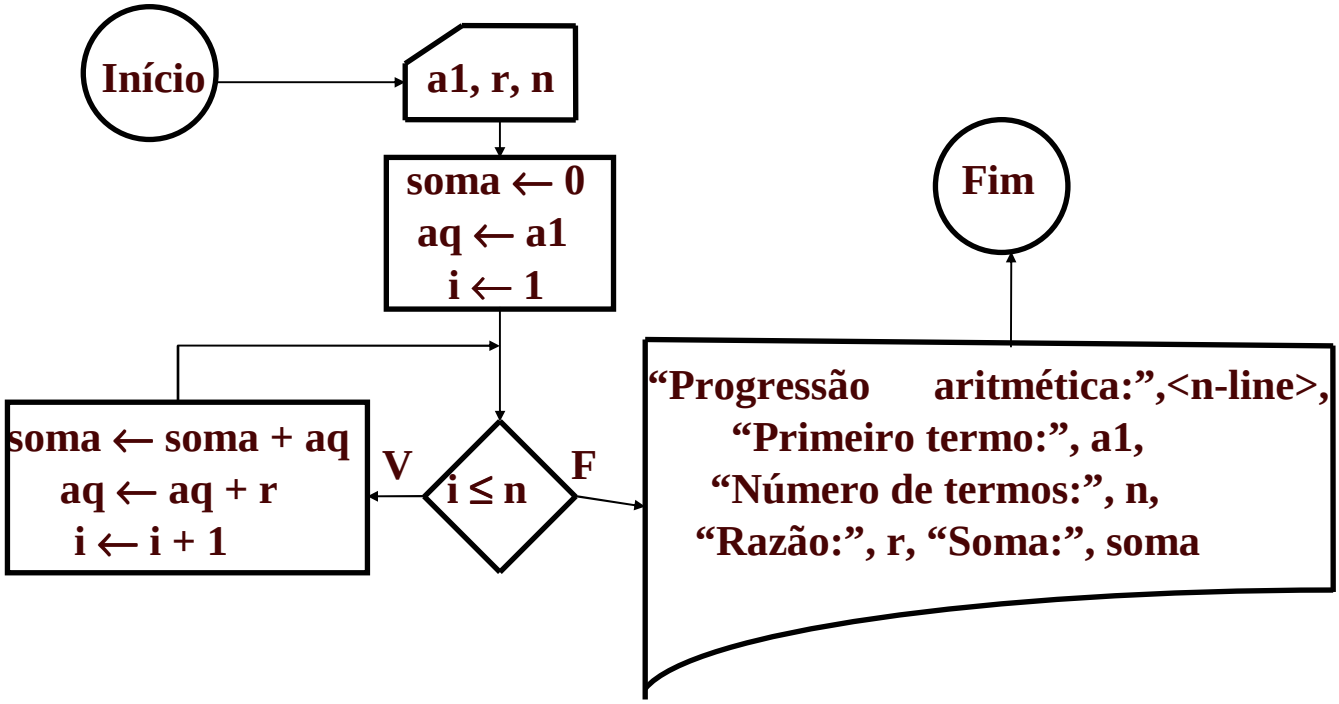



- Os comandos são guardados em blocos e o fluxo de controle é expresso por setas.

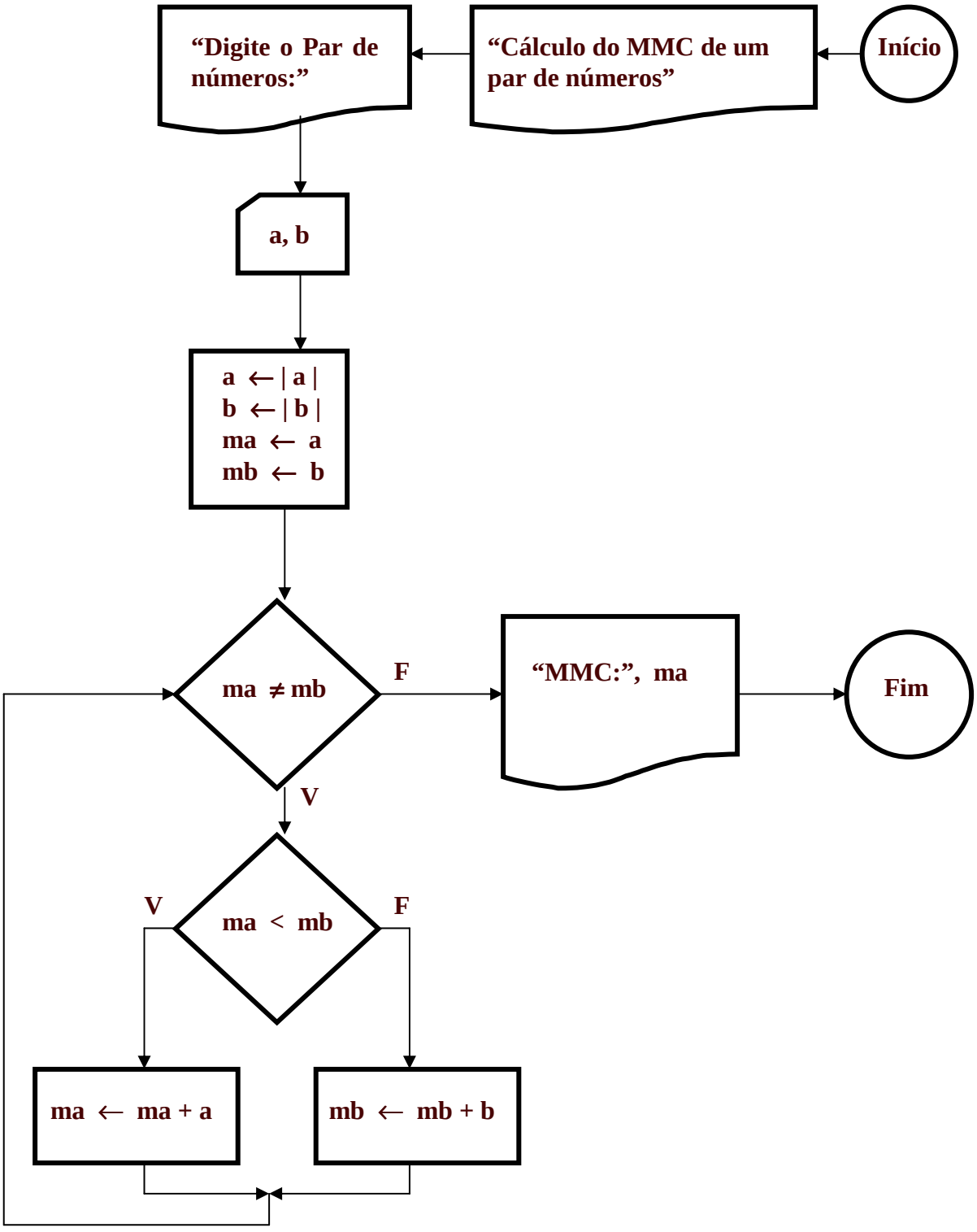
- Neste exemplo: retângulo é conjunto de comandos de atribuição; losango é decisão; cartão é entrada; folha é escrita; círculo é início e final.
- **Exemplo 2.12:** fluxograma da equação do 2.o grau:




- **Exemplo 2.13:** fluxograma da soma da PA:



- **Exercício 2.1:** Escrever um programa em C para o seguinte fluxograma destinado a calcular o MMC de um par de números:



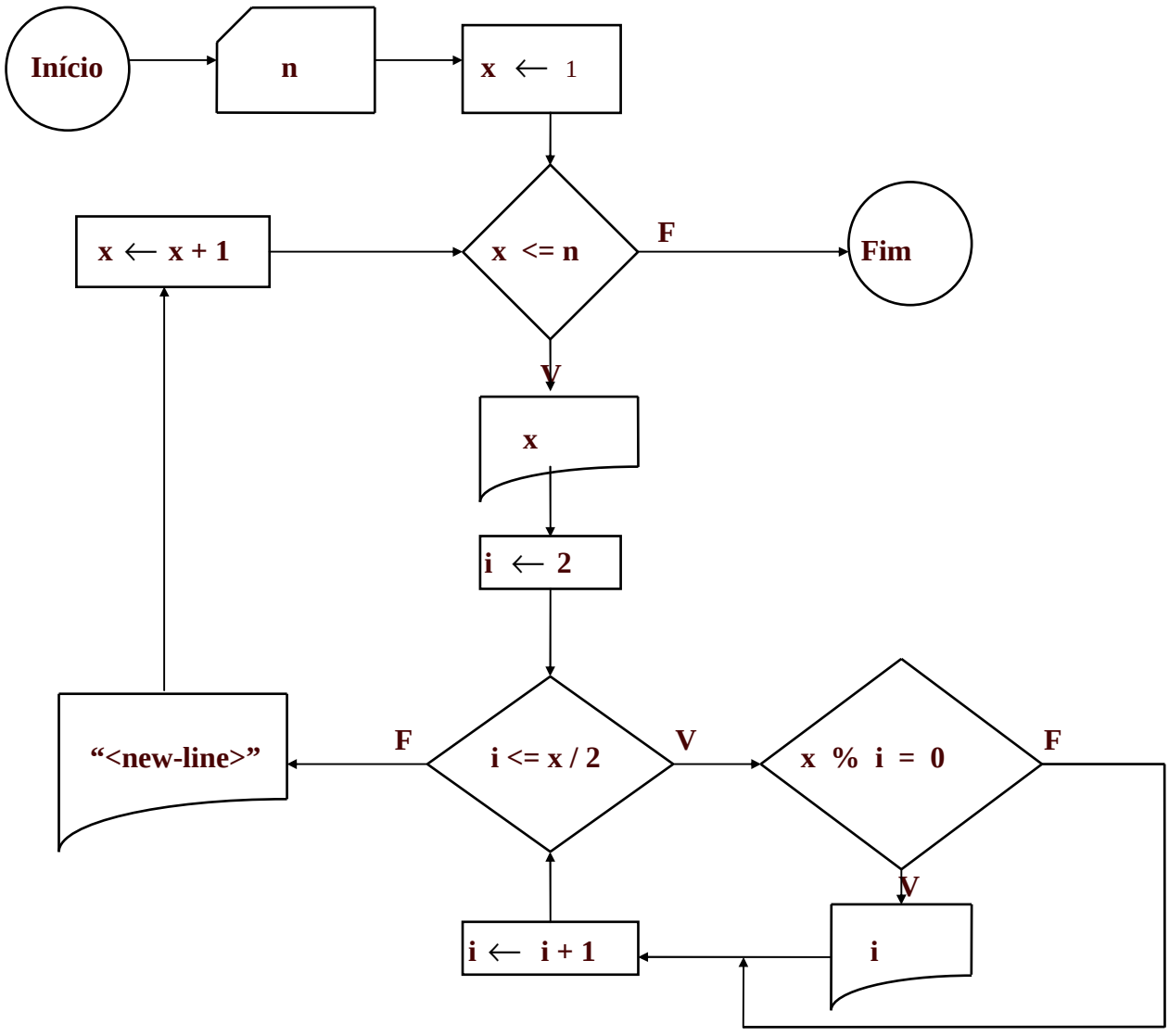
- **Exercício 2.2:** Desenhar o fluxograma para o seguinte programa para calcular o MDC de um par de números:

```
#include <stdio.h>
#include <math.h>

void main ()
{
        int a, b, aux;

        printf ("Calculo de MDC\n");
        printf ("Par de numeros: ");
        scanf("%d%d",&a,&b);
        a = abs(a);  b = abs(b);
        while  (b>0) {
                aux=a;
                a=b;
                b=aux%b;
        }
        printf ("MDC: %d\n", a);
}
```

- **Exercício 2.3:** Escrever um programa em C para o seguinte fluxograma destinado a calcular os divisores dos números entre 1 e **n**:



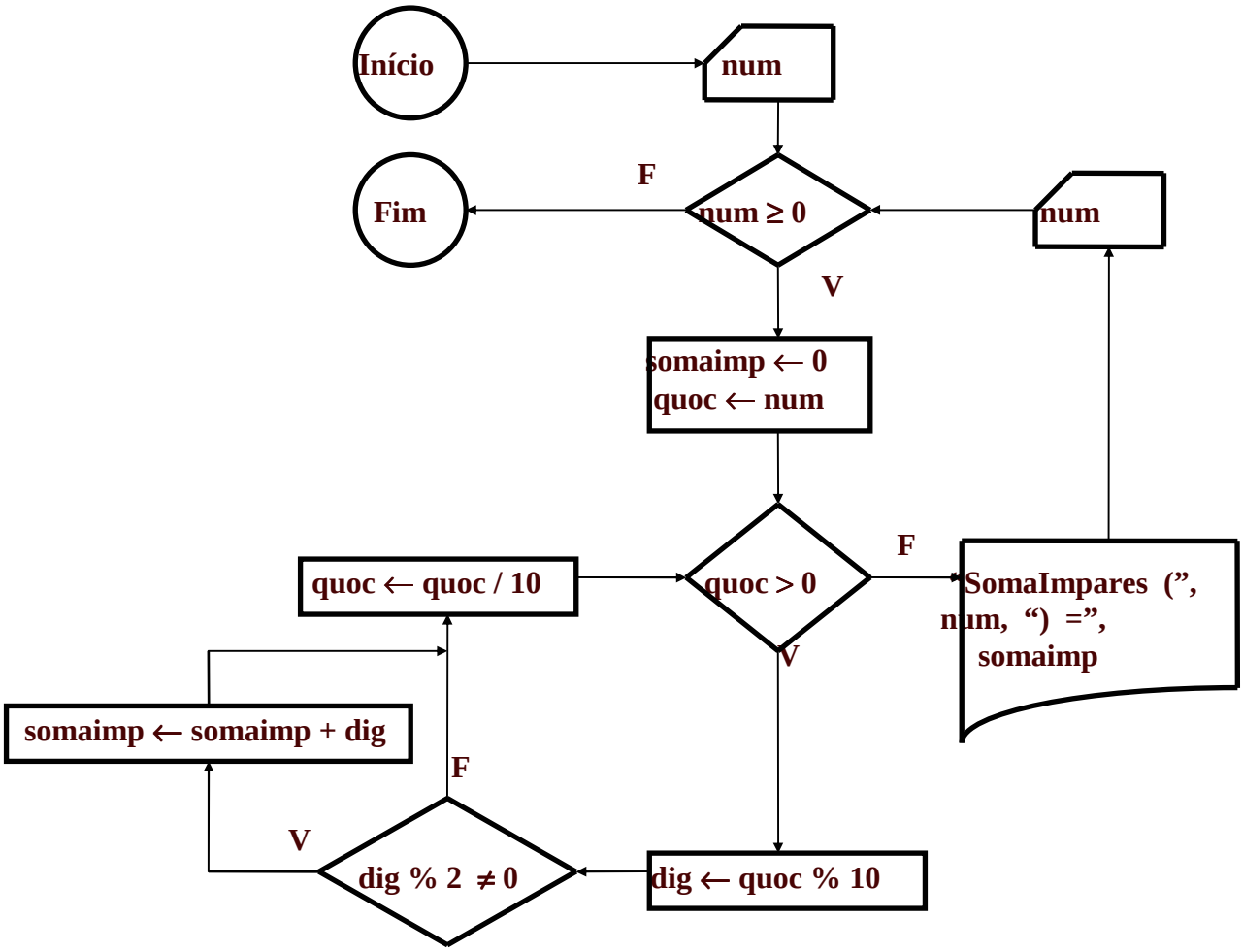
- **Exercício 2.4:** Desenhar o fluxograma para o seguinte programa para encontrar os números perfeitos entre 1 e **n** (número perfeito é aquele cuja soma de seus divisores próprios é igual a si):

```
#include <stdio.h>

void main ( )
{
      long n, i, div, soma;

      printf ("Digite um numero inteiro positivo: ");
      scanf ("%ld", &n);
      printf ("\nNumeros perfeitos entre 1 e %ld:\n\n", n);
      i = 1;
      while (i<=n) {
            soma = 0; div = 1;
            while (div * 2 <= i) {
                  if (i % div == 0)
                        soma = soma + div;
                  div = div + 1;
            }
            if (soma == i) printf ("%12ld", i);
            i = i + 1;
      }
}
```

- **Exercício 2.5:** Escrever um programa em C para o seguinte fluxograma destinado a calcular a soma dos dígitos ímpares de vários números positivos lidos:



## 2.4 - Estrutura de um Programa em C

- Programas em C têm estrutura flexível:
  - É possível escrever programas desorganizados que funcionam.
  - Esses programas são ilegíveis, de difícil correção.

- Nesta matéria, sugere-se uma organização.

- Sugestão para a estrutura geral:

  - Declarações globais
  - Subprogramas
  - Programa principal

  Funções

- Sugestão para a estrutura de uma função:

```
Cabeçalho  {
      Declarações locais
      Comandos
}
```

- Ver como exemplos os programas das páginas 10, 22, 25 e 26.

- Neste tópico será visto o básico; aspectos mais avançados serão vistos em capítulos específicos.

### 2.4.1 - Cabeçalho de função

- Todo subprograma em C é uma **função**; o programa principal também é uma função.

- Toda função tem um **tipo** e pode ter uma lista de **parâmetros**.

- Neste capítulo será abordado apenas o **programa principal**; subprogramas serão vistos em capítulo próprio.

- Até agora, o cabeçalho do programa principal dos programas vistos é:

```
main ( )
```

  O tipo não está explícito, nem há parâmetros entre os parêntesis.

- Quando não se especifica o tipo de uma função, por *default* ele é inteiro (**int**); quando se deseja que a função não tenha tipo, seu tipo deve ser explicitado como **void**.

```
void  main  ( )
```

### 2.4.2 - Declarações globais e locais

- Uma declaração deve ser **global** quando for usada por várias funções do programa.

- Quando a declaração for usada só em uma função, ela pode ser **local** a essa função.

## 2.4.3 - Diretivas de pré-processamento

- Toda declaração do programa iniciada por um '#' recebe o nome de **diretiva de pré-processamento**.

- **Pré-processamento** é uma fase da compilação anterior à compilação propriamente dita.

- Aqui serão vistas duas dessas diretivas:

**#define** e **#include**

### a) Diretiva #include:

- Um programa pode incorporar um ou mais arquivos

- **#include** incorpora outros arquivos ao programa.

- Arquivos a serem incorporados:

  - Arquivos da biblioteca da Linguagem C (seu nome deve estar entre < > );
  - Arquivos auxiliares do usuário (seu nome deve estar entre " ").

- **Exemplo 2.14:** diretivas #include:

```
#include <stdio.h>
#include "auxprog.h"
```

- Os arquivos devem ter a extensão **.c** ou **.h**.

**b) Diretiva #define:**

- Insere constantes simbólicas no programa.
- **Exemplo 2.15:** diretivas #define:

```
#define PI 3.1416
#define LIMITE 100
#define EQ ==
#define ENQUANTO while
#define SE if
#define SENAO else
```

- O **pré-processador** troca no programa todas as ocorrências de PI, LIMITE, EQ, ENQUANTO, SE e SENAO por 3.1416, 100, ==, while, if e else, respectivamente.
- É comum o programador trocar " == " por " = " ; usando EQ, ele pode evitar essa distração.
- O programador pode ser *nacionalista* e querer escrever programas só em Português.
- Podem ocorrer várias diretivas **#define** para um mesmo nome, num mesmo programa:
  - Isso não é aconselhável; pode contribuir para confusão.

- **Exemplo 2.16:** sejam os arquivos ***defines.h*** e ***preproc.h***:

defines.h:

```
#define LIMITE 100
#define   EQ   ==
#define   SE   if
#define SENAO else
```

preproc.c:

```
#include <stdio.h>
#include "defines.h"`
void main () {
      int i;

      printf ("LIMITE_1: %d\n", LIMITE);
      i = 100;
      #define LIMITE 200
      SE (i EQ LIMITE)
            printf ("i: %d", i);
      SENAO
        printf ("LIMITE_2: %d", LIMITE);
}
```

Arquivo ***preproc.c*** já pré-processado:

```
biblioteca stdio.h
void main ( )  {
      int i;  printf (“LIMITE_1: %d\n”, 100);  i = 100;
      if  (i == 200)  printf (“i: %d”, i);
      else  printf (“LIMITE_2: %d”, 200);
}
```

Saída no vídeo:

```
LIMITE_1: 100
LIMITE_2: 200
```

O pré-processador não altera LIMITE dentro de “ ”.

## 2.4.4 - Declaração de variáveis

- O **tipo** de toda variável de um programa deve ser **declarado** antes dela ser usada.

- Principais tipos primitivos:

  **char** : para caracteres;
  **int**, **long**, **unsigned**, **unsigned long**, **short**: para inteiros;
  **float**, **double**: para reais.

**a)** O tipo **char:** guarda 1 caractere; ocupa 1 byte.

- Os caracteres têm cada um sua representação interna.

- Tabela ASCII para caracteres:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | | | | | | | bel | bs | ht |
| 10 | nl | | | cr | | | | | | |
| 20 | | | | | | | | esc | | |
| 30 | | | sp | ! | “ | # | $ | % | & | ’ |
| 40 | ( | ) | * | + | , | - | . | / | 0 | 1 |
| 50 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 60 | < | = | > | ? | @ | A | B | C | D | E |
| 70 | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y |
| 90 | Z | [ | \ | ] | ^ | _ | ‘ | a | b | c |
| 100 | d | e | f | g | h | i | j | k | l | m |
| 110 | n | o | p | q | r | s | t | u | v | w |
| 120 | x | y | z | { | \| | } | ~ | del | | |

- Observações:
  - Os dígitos 0 .. 9 têm representação 48 .. 57;
  - As letras maiúsculas têm representação 65 .. 90;
  - As letras minúsculas têm representação 97 .. 122;
  - O espaço em branco tem representação 32;
  - Representação de operadores, separadores e caracteres de pontuação: veja a tabela.

- Caracteres especiais que não são escritos:

| carac | em C | repres. interna | significado |
|---|---|---|---|
| nul | \0 | 0 | carac. nulo |
| bel | \a | 7 | campainha |
| bs | \b | 8 | volta um espaço |
| ht | \t | 9 | tabulação |
| nl | \n | 10 | próxima linha |
| cr | \r | 13 | início da linha |

- **Exemplo 2.17:** impressão de caracteres: os comandos

```
char c;
c = 'a';
printf ("%3c%3d%3c%3c%3c", c, c, c+1, c+2, c+3);
```

produzirão:

```
__a_97__b__c__d
```

- **Exemplo 2.18:** caracteres especiais: o comando:

  printf (“abcde\nabcde\rxxx\nabcde”);

  produzirá:

  abcde
  xxxde
  abcde

- **Exemplo 2.19:** acionamento da campainha: o comando:

  **for** ( i = 1; i <= 1000; i = i + 1 ) printf (“\a”);
  tocará a campainha por alguns segundos.

**b)** Os tipos inteiros **short**, **int**, **long** e **unsigned**:

| Tipo | N.o de bytes | Intervalo de valores |
|---|---|---|
| int | 2 | -32768 a +32767 |
| short | 2 | -32768 a +32767 |
| long | 4 | -2147483648 a +2147483647 |
| unsigned | 2 | 0 a 65535 |

**c)** Os tipos reais **float** e **double**:

- Há duas notações para números reais: a notação **decimal** e a notação **exponencial**.

- **Exemplo 2.20:** as notações decimal e exponencial:

| decimal | exponencial |
|---|---|
| 317.42 | $0.31742 * 10^{3}$ |
| 57325000000000 | $0.57325 * 10^{14}$ |
| 0.000000161 | $0.161 * 10^{-6}$ |

- Nos programas em C

$0.161 * 10^{-6}$ equivale a 0.161e-6

- A representação de números reais no computador tem duas características quantitativas:
  - número de dígitos significativos (**precisão**)
  - variação do expoente (**intervalo**)

| **tipo** | **precisão** | **intervalo** |
|---|---|---|
| float | 6 dígitos significativos | -38 a +38 |
| double | 15 dígitos significativos | -308 a +308 |

- Números reais não são representados com exatidão;

  O número 3426175.8390176294015 (20 dígitos significativos) é representado como:

  float: $0.342617 * 10^{7}$
  double: $0.342617583901762 * 10^{7}$

**d)** O tipo **lógico**:

- Não existe em C, mas pode ser simulado, como no exemplo a seguir.

- **Exemplo 2.21:** simulação do tipo lógico

```
#include  <stdio.h>
#define  logical  char
#define  TRUE  1
#define  FALSE  0

void main ( )  {
      logical log;
      log = TRUE;
      if  (log)  printf  (“TRUE”);
      else  printf  (“FALSE”);
}
```

**e) Outros tipos:** serão vistos em capítulos específicos:

matrizes, estruturas, ponteiros, tipos enumerativos.

## 2.4.5 - Comentários nos programas

- Usados para documentar e elucidar os programas.
- Tudo entre **/*** e ***/** é um comentário; o compilador reduz isso a um espaço em branco.