

# Cap. III - ESTRUTURAS DE CONTROLE

## 3.1 - Comandos de Atribuição

### 3.1.1 - Forma geral

- Em algoritmos: *variável*  $\leftarrow$  *expressão* ;
- Em C: *variável* = *expressão* ;
- *Expressão* pode conter:  
constantes, variáveis, operadores e chamadas de funções.
- Constantes podem ser:
  - inteiras: 5, 2314, -801
  - reais: 13.47, 5.2E20, 0.03E-7
  - caracteres: 'a', 'B', ':', '+', '\n', '\0', '\\"', '\\', '\\'
  - octais: 014, 0712
  - hexadecimais: 0x182, 0XA27
- Variáveis podem ser declaradas como constantes:

**const int** a = 3;

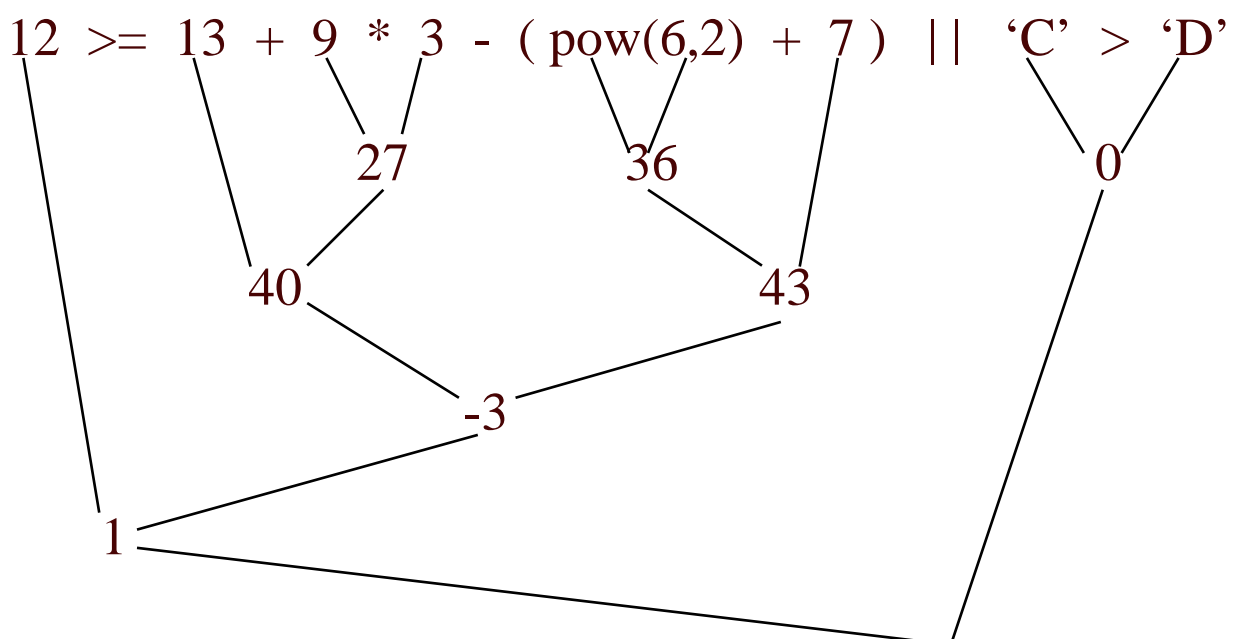
O valor de **a** não pode ser alterado no programa.

### 3.1.2 - Operadores e sua hierarquia

- Tabela da hierarquia de operadores:

Ordem	Classe	Operadores
1	Unários	+ - ! ( ) ++ --
2	Multiplicativos	* / %
3	Aditivos	+ -
4	Relacionais	< <= > >=
5	De Igualdade	== !=
6	Lógico Multiplicativo	&&
7	Lógico Aditivo	
8	De Atribuição	= += -= *= /= %=

- Exemplo 3.1:** cálculo de uma expressão:



### 3.1.3 - Funções de biblioteca de C

a) **Funções matemáticas:** pertencentes ao arquivo `<math.h>`

- **cos** ( *expressão* ), **sin** ( *expressão* ), **tan** ( *expressão* ): cosseno, seno e tangente de *expressão*.
- **acos** ( *expr* ), **asin** ( *expr* ), **atan** ( *expr* ): arco-cosseno, arco-seno e arco-tangente de *expr*.
- **cosh** ( *expr* ), **sinh** ( *expr* ), **tanh** ( *expr* ): cosseno, seno e tangente hiperbólicos de *expr*.
- **exp** ( *expr* ):  $e^{\text{expr}}$ .
- **log** ( *expr* ):  $\log_e \text{expr}$ .
- **log10** ( *expr* ):  $\log_{10} \text{expr}$ .
- **ceil** ( *expr* ): teto de *expr*.
- **floor** ( *expr* ): piso de *expr*.
- **abs** ( *expr* ):  $| \text{expr} |$ , inteiro.
- **fabs** ( *expr* ):  $| \text{expr} |$ , real.
- **pow** ( *x*, *y* ):  $x^y$ .

- **sqrt** ( *expr* ):  $\sqrt{expr}$ .

**b) Testes de caracteres:** pertencentes ao arquivo *<ctype.h>*

- **isalnum** ( *c* ): *c* é alfanumérico
- **isalpha** ( *c* ): *c* é letra
- **isdigit** ( *c* ): *c* é dígito decimal
- **islower** ( *c* ): *c* é letra minúscula
- **isupper** ( *c* ): *c* é letra maiúscula
- **isxdigit** ( *c* ): *c* é dígito hexadecimal
- **isspace** ( *c* ): *c* é espaço em branco
- **ispunct** ( *c* ): *c* é caractere de pontuação
- **isprint** ( *c* ): *c* é caractere *imprimível*
- **isctrl** ( *c* ): *c* é caractere de controle

- **Exemplo 3.2:** impressão de letras maiúsculas lidas:

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

```
void main () {
    char c; FILE *file1;
    file1 = fopen ("myfile", "w");
    do {
        scanf ("%c", &c);
        if (isupper (c)) fprintf (file1, "%c", c);
    } while (! isctrl (c));
    fclose (file1);
}
```

Se a entrada for: abcd +=\*& EFGH 123456 %faIJ enter

a saída no arquivo *myfile* será: EFGHIJ

### c) Outras funções:

- Manipulação de cadeias de caracteres
- Entrada e saída de dados
- Funções especiais para ponteiros e alocação de memória

Serão vistas em tópicos específicos.

#### 3.1.4 - Operações lógicas

a	b	!a	a && b	a    b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

- Em operações lógicas, todo operando diferente de 0 (zero) torna-se 1 (um).
- **Exemplo 3.3:** operações lógicas com números quaisquer:

$$\begin{aligned}452 \ || \ 0 &= 1 \ || \ 0 = 1 \\36 \ \&\& \ 2 &= 1 \ \&\& \ 1 = 1 \\17502 \ \&\& \ 0 &= 1 \ \&\& \ 0 = 0 \\! \ 57 &= ! \ 1 = 0\end{aligned}$$

## 3.2 - Comandos Compostos

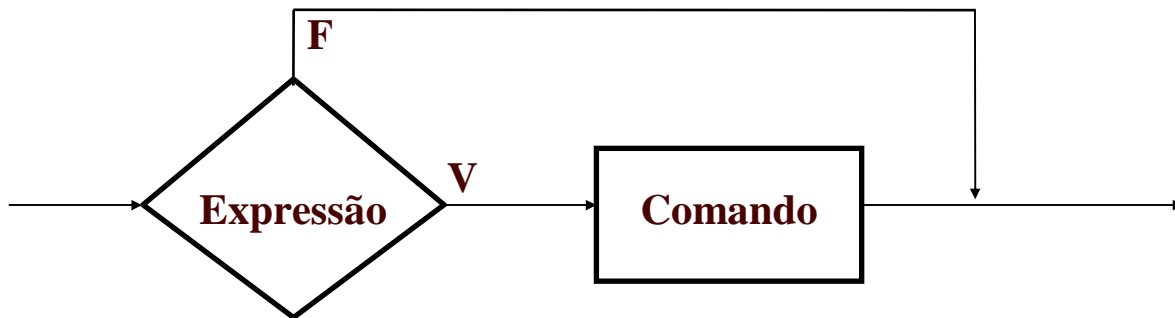
- ***Comando\_composto*** := { *Comando* *Comando* . . . . .  
*Comando* }
- ***Comando*** := *Comando\_simples* | *Comando\_composto* |  
*Comando\_condicional* | *Comando\_iterativo* |  
*Comando\_de\_seleção*
- ***Comando\_simples*** := *Comando\_de\_atribuição* |  
*Comando\_de\_entrada* | *Comando\_de\_saída* |  
*Chamada\_de\_subprograma* | *Comando\_vazio*
- ***Comando\_vazio*** := ;

## 3.3 - Comandos Condicionais

- ***Comando\_condicional*** := **if** ( *Expressão* ) *Comando\_1* |  
**if** ( *Expressão* ) *Comando\_1* **else** *Comando\_2*
- ***Comando\_1*** é executado caso ***Expressão*** ≠ 0; caso contrário, ***Comando\_2*** será executado.

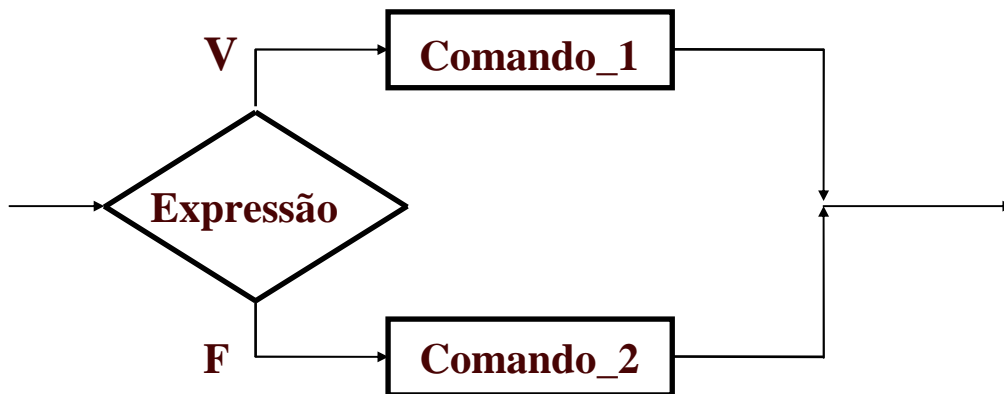
- **if** ( *Expressão* ) *Comando*

Fluxograma:



- **if** ( *Expressão* ) *Comando\_1* **else** *Comando\_2*

Fluxograma:



- **Obs:** no trecho de programa

**if** ( *Expr\_1* ) **if** ( *Expr\_2* ) *Comando\_1* **else** *Comando\_2*

o **else** corresponde ao segundo **if**.



- Para que ele corresponda ao primeiro **if**, deve-se usar:

```
if ( Expr_1 ) {  
    if ( Expr_2 ) Comando_1  
}  
else Comando_2
```

### 3.4 - Comandos iterativos

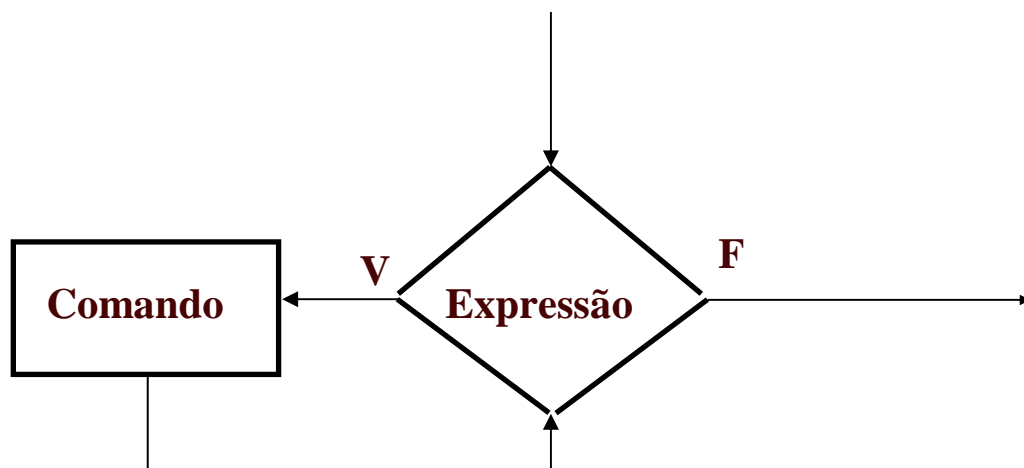
- *Comando\_iterativo* := *Comando\_while* | *Comando\_do* | *Comando\_for*

#### 3.4.1 - Comando while

- *Comando\_while* := **while** ( *Expressão* ) *Comando*

Enquanto *Expressão*  $\neq$  0, *Comando* é executado repetidamente.

Fluxograma:

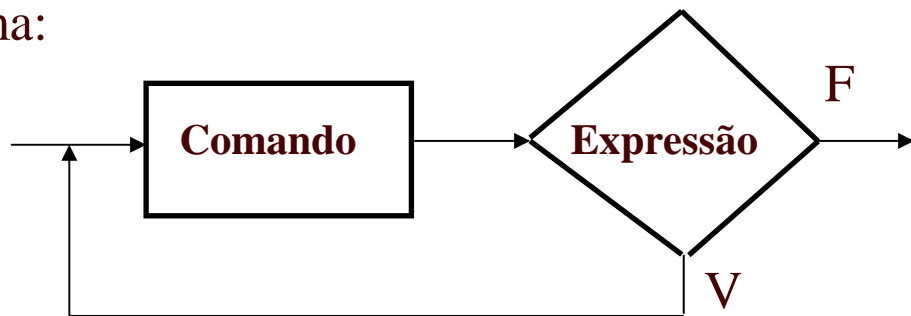


### 3.4.2 - Comando do

- *Comando\_do* := **do** *Comando* **while** ( *Expressão* ) ;

*Comando* é executado repetidamente enquanto *Expressão*  $\neq 0$ .

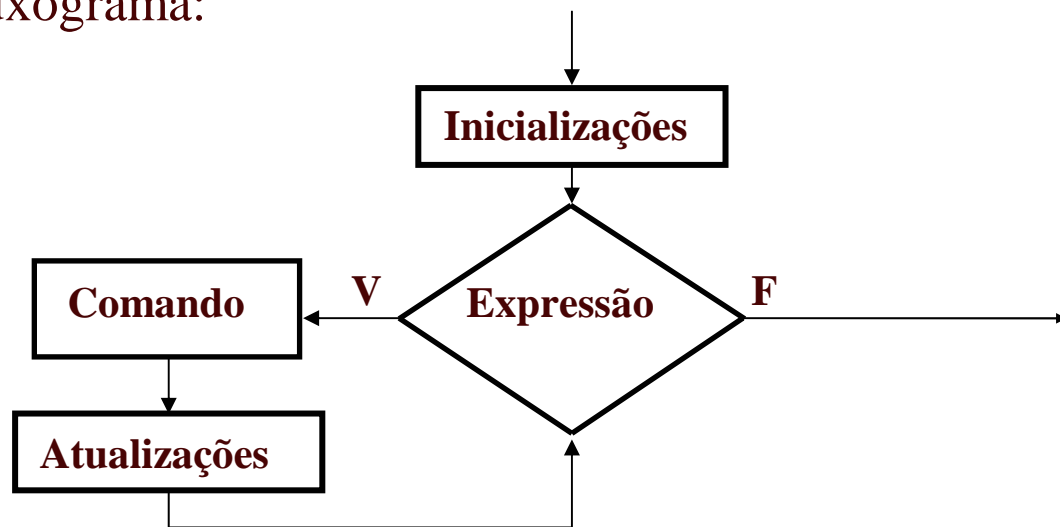
Fluxograma:



### 3.4.3 - Comando for

- *Comando\_for* := **for** ( *Inicializações* ; *Expressão* ; *Atualizações* ) *Comando*
- *Inicializações* e *Atualizações*: lista de zero ou mais *Comandos simples* separados por **vírgula** e não por **ponto e vírgula**.
- O *Comando\_for* acima equivale a  
*Inicializações*  
**while** ( *Expressão* ) {  
    *Comando*  
    *Atualizações*  
}

Fluxograma:



## 3.5 – Exemplos com Comandos Condicionais e Iterativos

- **Exemplo 3.4:** Soma da PA usando comando *for*

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

- **Obs:** o trecho marcado pode ser substituído por:

```
for (soma=0, aq=a1, i=1; i<=n; soma=soma+aq, aq=aq+r, i=i+1) ;
```

- **Exemplo 3.5:** Cálculo do MDC de vários pares de números

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

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

```
void main ( ) {
```

```
    int a, b, aux; char c;
```

```
    do {
```

```
        printf ("Calculo de MDC ? (s/n) ");
```

```
        do scanf ("%c", &c); while (c != 's' && c != 'n');
```

```
        if (c == 's') {
```

```
            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\n", a);
```

```
        }
```

```
    } while (c == 's');
```

```
}
```

- **Obs:** O trecho marcado poderia ser substituído por:

```
printf ("Par de numeros:"); scanf ("%d%d",&a,&b);
```

```
for (a = abs(a), b = abs(b); b>0; b = aux % b) {
```

```
    aux = a; a = b;
```

}

ou simplesmente por:

```
for (printf ("Par de numeros:"), scanf ("%d%d",&a,&b),  
     a = abs(a), b = abs(b); b>0; aux = a, a = b, b = aux % b);
```

- **Exemplo 3.6:** Inversão dos dígitos de um número decimal

```
#include <stdio.h>  
#include <stdlib.h>  
void main ( )  
{  
    long num, inv; char c;  
    do {  
        printf ("Inverter um numero ? (s/n) ");  
        do scanf ("%c", &c); while (c != 's' && c != 'n');  
        if (c == 's') {  
            printf ("  Numero:");  
            scanf ("%ld",&num); num = labs (num);  
            printf ("  %ld invertido torna-se ", num);  
            inv = 0;  
            while (num > 0) {  
                inv = 10 * inv + num % 10; num = num/10;  
            }  
            printf ("%ld\n\n", inv);  
        }  
    } while (c == 's');  
}
```

**Obs:** o trecho marcado  
pode ser substituído por:

```
for (inv = 0; num > 0; num = num/10)  
    inv = 10 * inv + num % 10;
```

### 3.6 – Metodologia *Top-Down* para Desenvolvimento de Algoritmos e Programas

- Um problema pode ser decomposto em mais de um subproblemas menores.
- Cada subproblema também pode ser decomposto da mesma maneira.
- Os subproblemas são subdivididos até que se obtenha uma coletânea de problemas triviais.
- O algoritmo do problema inicial é uma coletânea de comandos, um comando para resolver cada um desses problemas triviais.
- **Exemplo 3.7:** Desenvolvimento de um algoritmo para encontrar os  $n$  primeiros números primos:

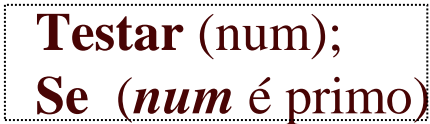
#### 1.a Etapa:

```
Primeiros_Números_Primos {  
    Escrever o título do relatório;  
    Pedir e Ler o valor de  $n$ ;  
    Se ( $n \leq 0$ )  
        Escrever mensagem de erro;  
    Senão  
        Encontrar e Escrever os  $n$  1.os números primos  
}
```

## 2.a Etapa:

```
Primeiros_Números_Primos {  
    Escrever (“Relação dos n primeiros números primos”);  
    Escrever (“Fornecer o valor de n: ”);  
    Ler (n);  
    Se (n ≤ 0)  
        Escrever (“Relação vazia”);  
    Senão {  
        num ← 0; cont ← 0;  
        Enquanto (cont < n) {  
            num ← num + 1;  
            Testar (num);  
            Se (num é primo) {  
                Escrever (num); cont ← cont + 1;  
            }  
        }  
    }  
}
```

Faltam detalhes



## 3.a Etapa: Desenvolvimento do trecho marcado acima

**Procurar** divisor para *num* no intervalo  $[2, \lfloor \sqrt{num} \rfloor ]$ ;  
**Se** (não encontrar divisor)



#### 4.a Etapa:

$\text{div} \leftarrow 2;$

**Enquanto** ((não encontrar divisor) e  $(\text{div}^2 \leq \text{num}))$  {

**Testar** (div);

**Se** (*div* é divisor de *num*)

        Encontrou divisor;

**Senão**

$\text{div} \leftarrow \text{div} + 1;$

}

**Se** (não encontrar divisor)

#### 5.a e Última Etapa:

$\text{div} \leftarrow 2;$  achou  $\leftarrow$  **Falso**;

**Enquanto** ((achou = **Falso**) e  $(\text{div}^2 \leq \text{num}))$  {

    Resto  $\leftarrow$  num % div;

**Se** (Resto = 0)

        Achou  $\leftarrow$  **Verdade**;

**Senão**

$\text{div} \leftarrow \text{div} + 1;$

}

**Se** (achou = **Falso**)

## Algoritmo final:

```
Primeiros_Números_Primos {  
    Escrever (“Relação dos n primeiros números primos”);  
    Escrever (“Fornecer o valor de n: ”);  
    Ler (n);  
    Se (n ≤ 0)  
        Escrever (“Relação vazia”);  
    Senão {  
        num ← 0; cont ← 0;  
        Enquanto (cont < n) {  
            num ← num + 1; div ← 2; achou ← Falso;  
            Enquanto ((achou = Falso) e (div2 ≤ num)) {  
                Resto ← num % div;  
                Se (Resto = 0)  
                    Achou ← Verdade;  
                Senão  
                    div ← div + 1;  
            }  
            Se (achou = Falso) {  
                Escrever (num); cont ← cont + 1;  
            }  
        }  
    }  
}
```

**Obs:** todos os comandos resolvem problemas triviais e já podem ser traduzidos para a Linguagem C

## Programa em C:

```
#include <stdio.h>
#include <math.h>
void main ( )
{
    short achou; int n, div, resto, cont; long num;

    printf ("Relacao dos n primeiros numeros primos\n\n");
    printf ("Fornecer o valor de n:"); scanf ("%d", &n);
    if (n <= 0)
        printf ("Relacao vazia");
    else {
        num = 0; cont = 0;
        while (cont < n) {
            num = num + 1; div = 2; achou = 0;
            while ((achou == 0) && (pow (div,2) <= num)) {
                resto = num % div;
                if (resto == 0)
                    achou = 1;
                else
                    div = div + 1;
            }
            if (achou == 0) {
                printf ("%6ld", num); cont = cont + 1;
                if (cont % 5 == 0) printf ("\n");
            }
        }
    }
}
```

- **Exemplo 3.8:** Fatores primos de números inteiros

Obter na tela relatórios do tipo:

Fatorar número? (s/n) s

Número: 504

Fatores primos de 504:

2 elevado a 3

3 elevado a 2

7 elevado a 1

Fatorar número? (s/n) s

Número: 348

Fatores primos de 348:

2 elevado a 2

3 elevado a 1

29 elevado a 1

Fatorar número? (s/n) s

Número: 750

Fatores primos de 750

2 elevado a 1

3 elevado a 1

5 elevado a 3

Fatorar número? (s/n) n

Será aplicada a  
metodologia  
***Top-Down***

### 1.a Etapa:

```
Fatores_Primos {  
    Repetir {  
        Perguntar se quer fatorar número;  
        Aguardar a resposta;  
        Se (resposta for sim) {  
            Perguntar qual o número;  
            Aguardar o número;  
            Fatorar o número;  
        }  
    } Enquanto (resposta for sim);  
}
```

### 2.a Etapa:

```
Fatores_Primos {  
    Repetir {  
        Escrever (“Fatorar número? (s/n) ”);  
        Obter resposta ‘s’ ou ‘n’;  
        Se (caractere lido for ‘s’) {  
            Escrever (“Número: ”);  
            Ler (num);  
            Escrever (“Fatores primos de ”, num, “:”);  
            Fatorar num, escrevendo cada fator  
                ao lado de seu expoente;  
        }  
    } Enquanto (caractere lido for ‘s’);  
}
```

### 3.a Etapa:

```

Fatores_Primos {
    Repetir {
        Escrever (“Fatorar número? (s/n) ”);
        Repetir
            Ler (c);
        Enquanto (c ≠ ‘s’ e c ≠ ‘n’);
        Se (c = ‘s’) {
            Escrever (“Número: ”);
            Ler (num);
            Escrever (“Fatores primos de ”, num, “: ”);
            Fatorar num, escrevendo cada fator
                ao lado de seu expoente;
        }
    } Enquanto (c = ‘s’);
}

```

↑  
Detalhar

#### 4.a Etapa: Desenvolvimento do trecho marcado acima

*sobra* ← num;

**Procurar** fatores para *sobra* no intervalo  $[2, \lfloor \sqrt{sobra} \rfloor]$   
 atualizando o valor de *sobra*, dividindo-o pelos  
 fatores encontrados, imprimindo cada fator ao lado  
 de seu expoente.

**Testar** *sobra* para coletar resíduo;

#### 5.a Etapa:

```

sobra ← num; fat ← 2;
Enquanto ( $\text{fat}^2 \leq \text{sobra}$ ) {
    Calcular expo de fat atualizando sobra;
    Se ( $\text{expo} > 0$ )
        Escrever (fat, “ elevado a ”, expo);
    fat ← fat + 1;
}
Se ((sobra ≠ 1) ou (num = 1))
    Escrever (sobra, “ elevado a 1”);

```

↑  
detalhar

## 6.a e Última Etapa:

```

sobra ← num; fat ← 2;
Enquanto ( $\text{fat}^2 \leq \text{sobra}$ ) {
    expo ← 0;
    Enquanto (sobra % fat = 0) {
        expo ← expo + 1; sobra ← sobra / fat;
    }
    Se ( $\text{expo} > 0$ )
        Escrever (fat, “ elevado a ”, expo);
    fat ← fat + 1;
}
Se ((sobra ≠ 1) ou (num = 1))
    Escrever (sobra, “ elevado a 1”)

```

## Algoritmo final:

```

Fatores_Primos {
  Repetir {
    Escrever (“Fatorar número? (s/n) ”);
    Repetir
      Ler (c);
    Enquanto (c ≠ ‘s’ e c ≠ ‘n’);
    Se (c = ‘s’) {
      Escrever (“Número: ”);
      Ler (num);
      Escrever (“Fatores primos de ”, num, “: ”);
      sobra ← num; fat ← 2;
      Enquanto (fat2 ≤ sobra) {
        expo ← 0;
        Enquanto (sobra % fat = 0) {
          expo ← expo + 1;
          sobra ← sobra / fat;
        }
        Se (expo > 0)
          Escrever(fat, “ elevado a ”, expo);
        fat ← fat + 1;
      }
      Se ((sobra ≠ 1) ou (num = 1))
        Escrever (sobra, “ elevado a 1”)
    }
  } Enquanto (c = ‘s’);
}

```

**Programa:** transformação dos *while*’s em *for*’s



```

#include <stdio.h>
#include <math.h>
void main ( )
{
    long num, sobra; int fat, expo; char c;
    do {
        printf ("Fatorar numero? (s/n) ");
        do scanf ("%c", &c);
        while (c != 's' && c != 'n');
        if (c == 's') {
            printf ("\tNumero: "); scanf("%ld",&num);
            num = labs(num);
            printf ("\n\tFatores primos de %ld:\n\n",
num);
            for (fat=2,sobra=num;pow(fat,2)<=sobra;fat=fat+1) {
                for (expo=0; sobra%fat == 0; sobra = sobra/fat)
                    expo = expo + 1;
                if (expo > 0)
                    printf ("\t\t%d elevado a %d\n", fat, expo);
            }
            if (sobra != 1 || num == 1)
                printf ("\t\t\t%ld elevado a 1\n", sobra);
            printf ("\n\n");
        }
    } while (c == 's');
}

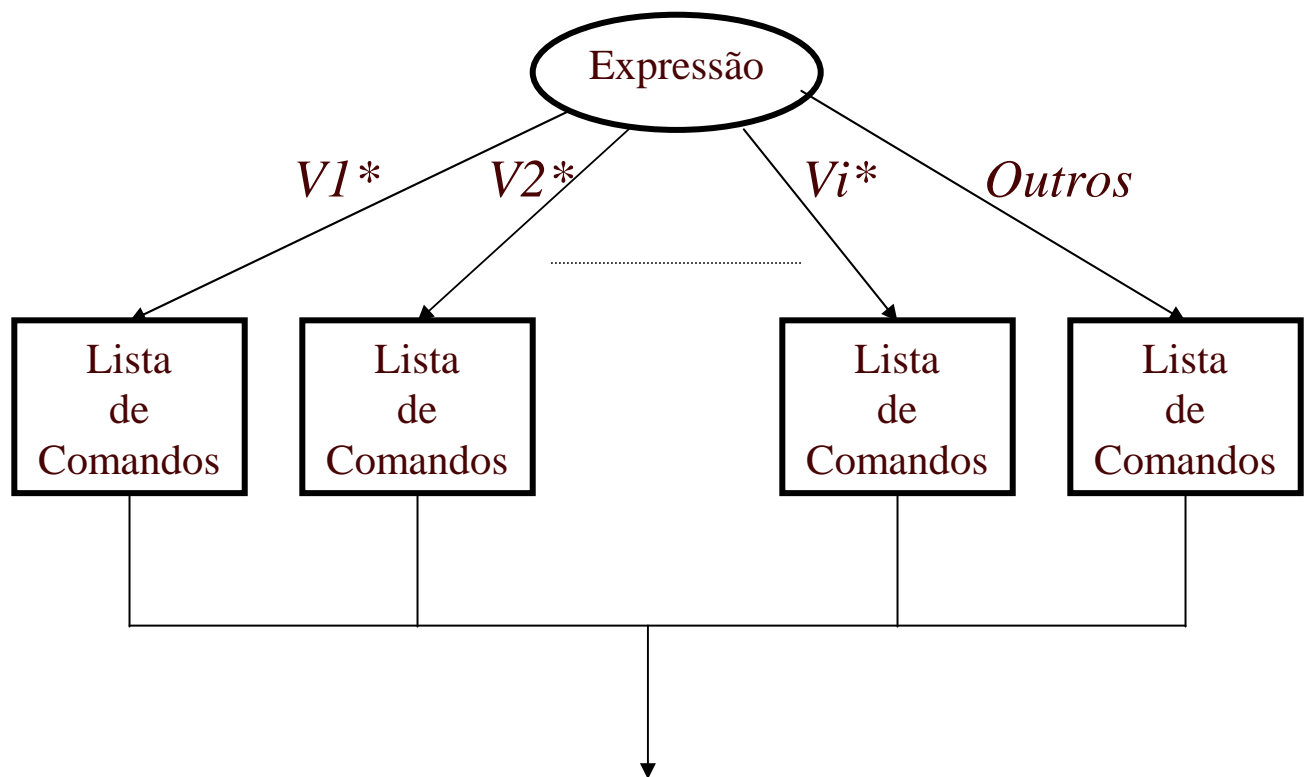
```

## 3.7 – Comandos de Seleção

- Algoritmos usam muito a seguinte estrutura:

**Caso** *expressão\_inteira* {  
 $V11, V12, \dots, V1m : lista\_comandos;$   
 $V21, V22, \dots, V2n : lista\_comandos;$   
 .  
 .  
 $Vi1, Vi2, \dots, Vip : lista\_comandos;$   
**Em outros casos:** *lista\_comandos;*  
}

- Em fluxogramas:



- Implementação em C:

```

switch ( expressão ) {
    case V11:
    case V12:
        .
        .
    case V1m: lista de comandos; break;

    case V21:
    case V22:
        .
        .
    case V2n: lista de comandos; break;

        .
        .
        .
        .
        .

    case Vi1:
    case Vi2:
        .
        .
    case Vip: lista de comandos; break;

    default: lista de comandos;
}

```

- **Exemplo 3.9:** Contagem de caracteres de vários tipos:

```

#include <stdio.h>

```

```

void main ()
{
    char c;
    int ct_a, ct_e, ct_i, ct_o, ct_u, ct_w, ct_y,
        ct_cons, ct_dig, ct_outros;

    /* Zerar todos os contadores */

    ct_a = 0; ct_e = 0; ct_i = 0; ct_o = 0; ct_u = 0; ct_w = 0;
    ct_y = 0; ct_cons = 0; ct_dig = 0; ct_outros = 0;

    /* Ler e classificar cada caractere */

    printf (“Digite uma frase:\n”);
    do {
        scanf ("%c", &c);
        if (c != '\n') {
            switch (c) {
                case 'a': case 'A': ct_a = ct_a+1; break;
                case 'e': case 'E': ct_e = ct_e+1; break;
                case 'i': case 'I': ct_i = ct_i+1; break;
                case 'o': case 'O': ct_o = ct_o+1; break;
                case 'u': case 'U': ct_u = ct_u+1; break;
                case 'w': case 'W': ct_w = ct_w+1; break;
                case 'y': case 'Y': ct_y = ct_y+1; break;

                default:
                    if (c >= '0' && c <= '9') ct_dig = ct_dig+1;
                    else if (c >= 'B' && c <= 'Z' ||

```

```

        c >= 'b' && c <= 'z')
        ct_cons = ct_cons+1;
    else ct_outros = ct_outros+1;
    }
}
} while (c != '\n');

/* Imprimir os contadores */

printf ("Letra A: %d\nLetra E: %d\nLetra I: %d\nLetra O:%d\n", ct_a, ct_e, ct_i, ct_o);
printf ("Letra U: %d\nLetra W: %d\nLetra Y: %d\nConsoantes: %d\n", ct_u, ct_w, ct_y, ct_cons);
printf ("Digitos: %d\nOutros: %d\n", ct_dig, ct_outros);
}

```

## 3.8 – Comandos Especiais da Linguagem C

### 3.8.1 – Operações de acréscimo e decréscimo unitário

- Operadores: ++ e --

São operadores de um só operando  
O operando deve ser uma variável

- Expressões: ++a a++ --a a—

Podem ser subexpressões de expressões maiores

- Efeitos:

**a++** : o valor da expressão é o valor *antigo* de **a**;  
a variável **a** recebe um acréscimo unitário;  
**++a** : a variável **a** recebe um acréscimo unitário;  
o valor da expressão é o valor *novo* de **a**.

- **Exemplo 3.10:**

```
int i, a, b, c, d, e;  
i = 0; a = i; b = i++ - 5; c = i; d = ++i + 5; e = i;  
printf ("a:%d b:%d c:%d d:%d e:%d", a, b, c, d, e);
```

Resultado: a:0 b:-5 c:1 d:7 e:2

- As expressões **++a** e **a++** isoladas equivalem ao comando **a = a+1**
- Os efeitos do operador **--** nas expressões **--a** e **a--** e sua equivalência com o comando **a = a - 1** são análogos.

### 3.8.2 – Atribuição em expressões

- O comando *Variável* = *Expressão* (sem o ‘;’) é  
considere-rado uma expressão:  
Seu valor é o valor de *Expressão*;  
*Variável* recebe o valor de *Expressão*;  
Pode ser uma subexpressão de outra expressão

- **Exemplo 3.11:**

```
int a, b, c;  
c = (a=2) + (b=3);  
printf ("a:%d b:%d c:%d", a, b, c);
```

Resultado: a:2 b:3 c:5

- Esta propriedade pode ser usada para compactar vários comandos de atribuição; por exemplo, os comandos

```
a = b + c;  
d = e + f;  
g = a + d + h;
```

poderiam ser compactados em

```
g = (a = b+c) + (d = e+f) + h;
```

### 3.8.3 – Outros operadores de atribuição

- Operadores: += -= \*= /= %=  
• Seja **op** um operador entre [+ , - , \* , / , %]; o comando

*Variável* **op=** *Expressão*

equiivale a *Variável* = *Variável* **op** *Expressão*

- **Exemplo 3.12:**

```
int a, b, c;  
a = 2; b = 3; c = 41;  
a += 5; b *= 3; c %= a+b;  
printf("a:%d b:%d c:%d", a, b, c);
```

Resultado: a:7 b:9 c:9

### 3.8.4 – O operador *vírgula*

- Seja a expressão:

*Expr1, Expr2, Expr3, Expr4*

- As subexpressões *Expr1*, *Expr2*, *Expr3* e *Expr4* são calculadas da esquerda para a direita; o valor da expressão é o da última subexpressão, ou seja, é o valor de *Expr4*.

- **Exemplo 3.13:** sejam os comandos:

```
int a, b, c, x;  
x = (a = 2, b = 3, c = 41);  
printf("a:%d b:%d c:%d x:%d", a, b, c, x);
```

Resultado: a:2 b:3 c:41 x:41



### 3.8.5 – Expressões condicionais

- Forma:  $Expr1 \ ? \ Expr2 \ : \ Expr3$ 
  - Calcula-se  $Expr1$ ;
  - Se o valor for **diferente de zero (Verdadeiro)**, calcula-se o valor de  $Expr2$ , que será o valor da expressão condicional;
  - Se o valor for **igual a zero (Falso)**, calcula-se o valor de  $Expr3$ , que será o valor da expressão condicional.
- **Exemplo 3.14:** sejam os comandos:

```
int a, b, x;  
a = 2; b = 3;  
x = a > b ? a + b : b - a;  
printf ("a:%d b:%d x:%d", a, b, x);
```

Resultado: a:2 b:3 x:1

- Alguns comandos *if-else* simples podem ser substituídos por expressões condicionais.

- **Exemplo 3.15:** os comandos

```
int a, b;  
a = 2; b = 3;  
a > b ? (a = 5, b = 15) : (a = 50, b = 100);  
printf ("a:%d b:%d\n", a, b);
```

podem substituir os seguintes comandos:

```
a = 2; b = 3;  
if (a > b)  
    {a = 5; b = 15;}  
else  
    {a = 50; b = 100;}  
printf ("a:%d b:%d\n", a, b);
```

Em ambos, o resultado será: a:50 b:100

### 3.8.6 – Os comandos *break* e *continue*

- **Comando *break*:**

Saída anormal de um comando iterativo ou comando de seleção mais interno.

- **Exemplo 3.16:**

```

for ( ..... ; ..... ; ..... ) {
    .....
    .....
    while ( ..... ) {
        .....
        if ( ..... ) break;
        .....
        .....
    }
    ..... /* Proximo comando a ser executado depois do break */
    .....
}

```

- **Comando *continue*:** encerra a iteração corrente e inicia a iteração seguinte.

- **Exemplo 3.17:** o comando

```

for (Expr1; Expr2; Expr3) {
    Comandos 1
    if ( ..... ) continue;
    Comandos 2
}

```

equivale a

```

Expr1;
While (Expr2) {
    Comandos 1
    if ( ..... ) goto next;
    Comandos 2
next:
    Expr3;
}

```