Programação Procedimental

Comandos condicionais, Laços e Repetições

Aula 02

Prof. Felipe A. Louza

Roteiro

Roteiro

Expressões relacionais

Operadores relacionais realizam **comparações** entre duas expressões e retornam:

1. **Verdadeiro**: valor não nulo `(1)`
2. **Falso**: valor zero `(0)`

Operadores Relacionais

Os operadores relacionais da linguagem C são:

Operador	Operação
`==`	`igualdade`
`!=`	`diferente`
`>`	`maior`
`<`	`menor`
`>=`	`maior ou igual`
`<=`	`menor ou igual`

- Cuidado: Não confundir atribuição (=) com igualdade (==).

Expressões relacionais

Alguns exemplos:

- *expressão* == *expressão*

```
1  9 == 9  //Resultado é 1, True
2
3  9 == 10 //Resultado é 0, False
```

- *expressão* != *expressão*

```
1  9 != 9 //Resultado é 0, False
2
3  9 != 10 //Resultado é 1, True
```

Expressões relacionais

Alguns exemplos:

- *expressão* $>$ *expressão*

```
1  9 > 5 //Resultado é 1, True
```

- *expressão* $<$ *expressão*

```
1  9 < 5 //Resultado é 0, False
```

Expressões relacionais

Alguns exemplos:

- *expressão* $>=$ *expressão*

```
1  9 >= 5 //Resultado é 1, True
```

- *expressão* $<=$ *expressão*

```
1  9 <= 5 //Resultado é 0, False
```

Expressões relacionais

Retorno do operador:

- Valor inteiro: 0 (falso) ou 1 (verdadeiro)

```c
#include <stdio.h>

int main(){

  int x = 10;
  int y = 2;

  printf("%d\n", x==y+8);

return 0;
}
```

Roteiro

Expressões lógicas

Operadores lógicos:

- Sistema algébrico definido por *George Boole* (álgebra de Boole).
- Retornam valores booleanos: verdadeiro (1) ou falso (0)

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x'
0	1
1	0

Operadores Lógicos

Na linguagem C temos os seguintes operadores lógicos:

Operador	Nome
`&&`	`Conjunção (AND)`
`||`	`Disjunção (OR)`
`!`	`Negação (NOT)`

Expressões lógicas

Alguns exemplos:

- *expressão && expressão*: Retorna verdadeiro quando ambas as expressões são verdadeiras.

AND

x	*y*	*xy*
0	*0*	*0*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

```
1 int a = 0;
2 int b = 0;
3 int c =  (a == 0 && b == 0);
```

Expressões lógicas

Alguns exemplos:

- *expressão* || *expressão* : Retorna verdadeiro quando pelo menos uma das expressões é verdadeira.

OR

x	*y*	*x+y*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*1*

```
1 int a = 0;
2 int b = 1;
3 int c = (a == 0 || b == 0);
```

Expressões lógicas

Alguns exemplos:

- ! *expressão*: Retorna verdadeiro quando a expressão é falsa e vice-versa.

NOT

x	*x'*
0	*1*
1	*0*

```
1 int a = 0;
2 int b = 1;
3 int c = !(a != b);
```

Expressões lógicas

Importante:

- Qual seria o resultado da expressão a seguir?

$$0.1 < 0.3 < 0.5$$

- Em linguagem C:

```
1 int a = 0.1 < 0.3 < 0.5 ;
```

  - Retorna 0 (falso), por que?
  - Operadores relacionais são associativos à esquerda.

Expressões lógicas

Ordem das operações:

- Primeiro:

```
1 int a = 0.1 < 0.3 < 0.5;
```

- Em seguida:

```
1 int a = 1 < 0.5;
```

Expressões lógicas

Como resolver?

- Podemos utilizar o operador &&:

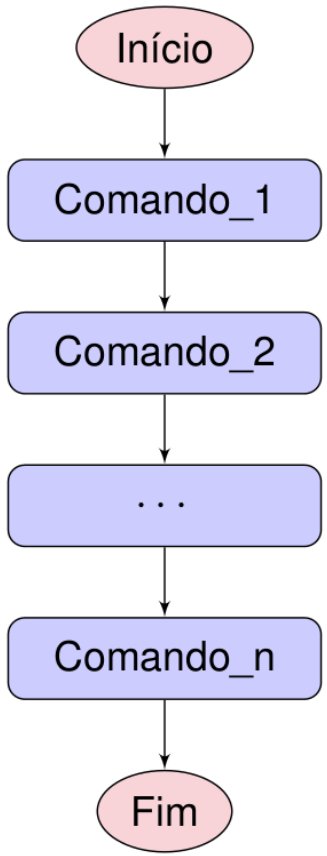
```
1 int a = (0.1 < 0.3) && (0.3 < 0.5);
```

Roteiro

Comandos condicionais

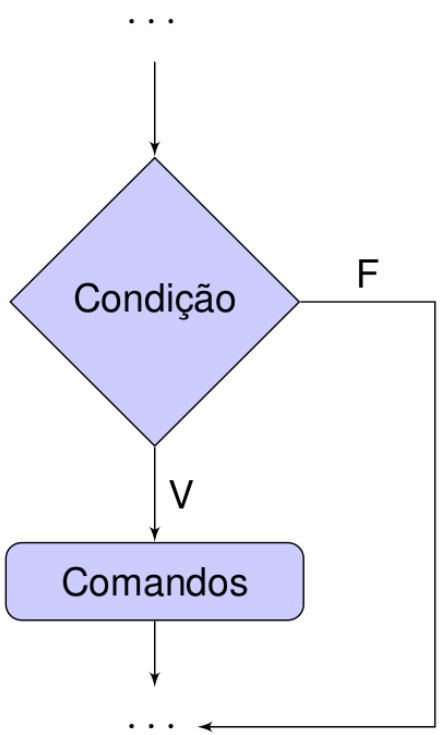
Até agora vimos apenas programas que executam operações sequencialmente:

```c
#include <stdio.h>

#define PI 3.1416

int main(){

  float raio;

  printf("Digite o valor do raio: ");
  scanf("%f", &raio);

  //calculando a área
  float area = PI * (raio*raio);
  printf("Área do circulo: %d\n", area);

return 0;
}
```



Comandos condicionais

Um comando condicional decide se um determinado bloco de comandos deve ser executado ou não, a partir do resultado de uma expressão relacional ou lógica.



Bloco de comandos

Bloco de comandos em C

- É um conjunto de instruções agrupadas limitado por { e }

```
1  #include <stdio.h>
2
3  #define PI 3.1416
4
5  int main(){ ⟵ Início de um bloco de comandos
6
7    float raio;
8    printf("Digite o valor do raio: ");
9    scanf("%f", &raio);
10
11   //calculando a área
12   float area = PI * (raio*raio);
13   printf("Área do circulo: %d\n", area);
14
15 return 0;
16 } ⟵ Fim de um bloco de comandos
```

Roteiro

Comandos condicionais

Comando `if`:

- O bloco de comandos é executado apenas se a expressão de teste for verdadeira:

```
1 if (<expressão de teste>) {
2   comando 1;
3   comando 2;
4   ...
5   comando n;
6 }
```

- Alternativamente:

```
1 if(<expressão de teste>)
2   um único comando;
```

Comandos condicionais

Podemos verificar se o raio fornecido pelo usuário é positivo:

```c
#include <stdio.h>

#define PI 3.1416

int main(){

  float raio;
  printf("Digite o valor do raio: ");
  scanf("%f", &raio);

  if(raio > 0.0){
    //calculando a área
    float area = PI * (raio*raio);
    printf("Área do circulo: %d\n", area);
  }

return 0;
}
```

Comandos condicionais

Exercício 1:

- Escreva um programa que lê do teclado um inteiro x e determina se x é par:

```c
#include <stdio.h>

int  main () {
    int x;
    scanf("%d", &x);
    if ((x % 2) == 0) {
       printf ("O valor é par.\n");
    }
}
```

Comandos condicionais

Exercício 2:

- Escreva um programa que lê do teclado um inteiro x e determina se x é par ou ímpar:

```c
#include <stdio.h>

int  main () {
    int x;
    scanf("%d", &x);
    if ((x % 2) == 0) {
       printf ("O valor é par.\n");
    }
    if ((x % 2) != 0) {
       printf ("O valor é ímpar.\n");
    }
}
```

Roteiro

Comandos condicionais

Uma variação do comando `if` é o `if/else`.

- Sintaxe:

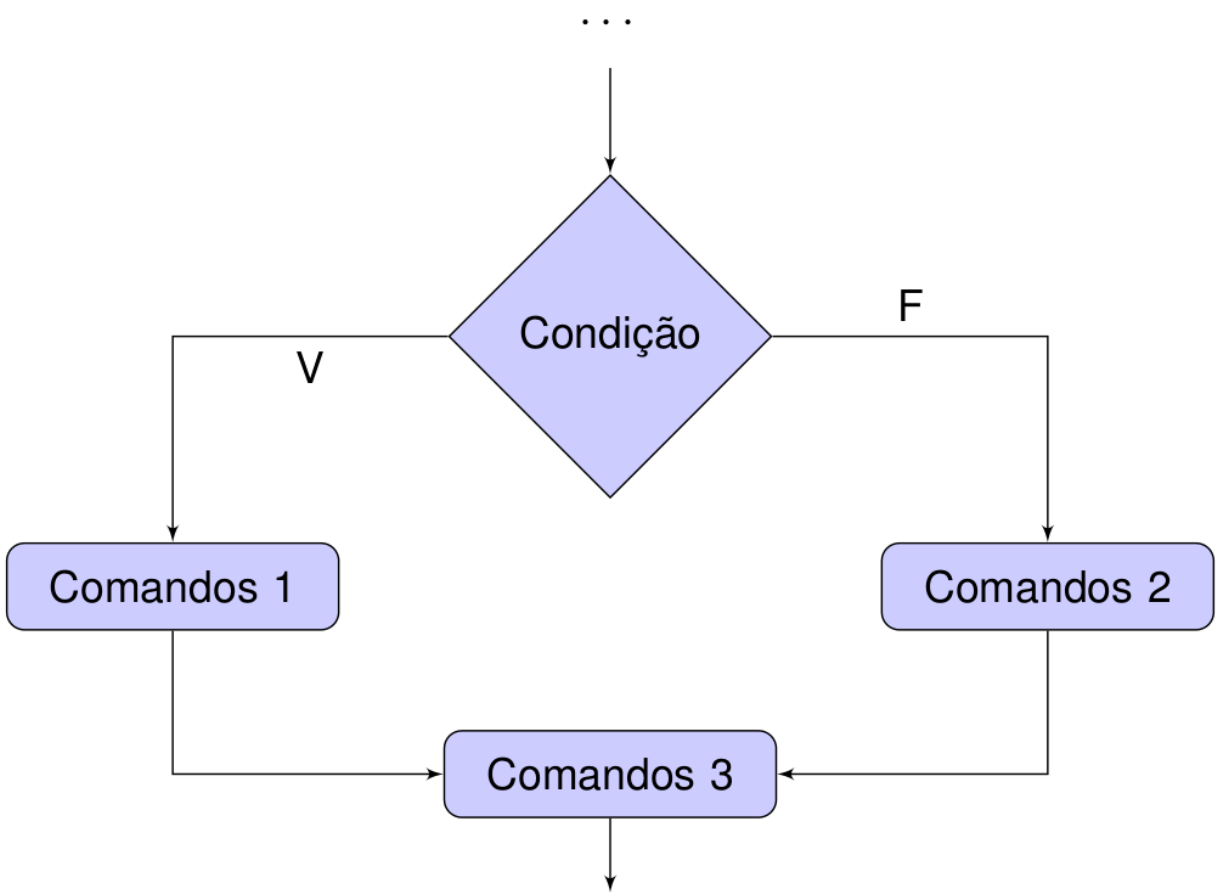
```
1 if(<expressão de teste>) {
2   //comandos executados se a expressão for verdadeira
3 }
4 else{
5   //comandos executados se a expressão for falsa
6 }
```

- Da mesma forma:

```
1 if(<expressão de teste>)
2   //comando executado se a expressão for verdadeira
3 else
4   //comando executado se a expressão for falsa
```

Comandos condicionais

Fluxograma:



Comandos condicionais

Exercício 3:

- Altere o programa que lê do teclado um inteiro x e determina se x é par ou ímpar:

```c
#include <stdio.h>

int  main () {
    int x;
    scanf("%d", &x);
    if ((x % 2) == 0) {
       printf ("O valor é par.\n");
    }
    else {
       printf ("O valor é ímpar.\n");
    }
}
```

Comandos condicionais

Cuidado com os operadores = (atribuição) e == (igualdade)

```c
#include <stdio.h>

int main(){
  int a = 2;

  if(a = 3){
    printf("Nota 0\n");
  }else{
    printf("Nota 10\n");
  }
}
```

- O programa acima imprime "Nota 0", pois na expressão dentro do `if`, temos uma atribuição, que sempre é verdadeiro.

Comandos condicionais

Comandos compostos `if/else`:

- Sintaxe:

```
1  if(<expressão de teste 1>) {
2    //comandos executados se a expressão 1 for verdadeira
3  }
4  else if(<expressão de teste 2>){
5    //comandos executados se a expressão 1 for falsa e 2 for verdadeira
6  }
7  else if(<expressão de teste 3>){
8    //comandos executados se 1 e 2 forem falsas e 3 for verdadeira
9  }
10 else{
11     //comando executado se 1, 2 e 3 forem falsas
12 }
```

Comandos condicionais

Comandos compostos `if/else`:

- Exemplo:

```c
#include <stdio.h>

int  main () {
    int x;
    scanf("%d", &x);
    if (x==0){
        printf ("O valor é zero.\n");
    }
    else if ((x % 2) == 0) {
       printf ("O valor é par.\n");
    }
    else {
       printf ("O valor é ímpar.\n");
    }
}
```

Roteiro

Comandos de decisão aninhados

Podemos ter comandos `if/else` dentro de outros blocos (aninhados).

```
1  if(<expressão de teste 1>) {
2    //comandos executados se a expressão 1 for verdadeira
3  }
4  else{
5    if(<expressão de teste 2>){
6      //comandos executados se a expressão 1 for falsa e 2 for verdadeira
7    }
8    else{
9      if(<expressão de teste 3>){
10       //comandos executados se 1 e 2 forem falsas e 3 for verdadeira
11     }
12     else{
13       //comando executado se 1, 2 e 3 forem falsas
14     }
15   }
16 }
```

- Obs. equivalente ao exemplo do Slide 16.

Comandos condicionais

Comandos `if/else` aninhados:

- Exemplo:

```c
#include <stdio.h>

int  main () {
    int x;
    scanf("%d", &x);
    if (x==0){
        printf ("O valor é zero.\n");
    }
    else{
      if ((x % 2) == 0) {
        printf ("O valor é par.\n");
      }
      else {
        printf ("O valor é ímpar.\n");
      }
    }
}
```

Roteiro

O comando `switch`

Podemos simplificar construções `if/else` aninhados com o comando `switch`:

```
1   switch(variável) {
2     case valor :
3         //comandos
4         break;
5     case valor :
6         //comandos
7         break;
8     case valor :
9         //comandos
10        break;
11    default:
12        //comandos
13  }
```

O comando `switch`

Exemplo:

```
1   char a = 'A';
2
3   switch(a){
4     case 'A': printf("Aprovado\n"); // if(a == 'A')
5               break;
6     case 'B': printf("Aprovado\n"); // else if(a == 'B')
7               break;
8     case 'C': printf("Aprovado\n"); // else if(a == 'C')
9               break;
10    default: printf("Reprovado\n"); // else
11              break;
12  }
```

- `default` é executado se todos os testes falharem.

O comando `switch`

Algumas restrições:

```
1   float a = 10.0;
2
3   switch(a){
4     case 10.0: printf("Aprovado\n");
5                break;
6     case  9.0: printf("Aprovado\n");
7                break;
8     case  8.0: printf("Aprovado\n");
9                break;
10    default: printf("Reprovado\n");
11               break;
12  }
```

- O `switch` só pode ser utilizado em condições que envolvam uma variável inteira ou caractere.

O comando `switch`

Algumas restrições:

```
1   int a = 10, b = 9;
2
3   switch(a){
4     case 10: printf("Aprovado\n");
5              break;
6     case  b: printf("Aprovado\n");
7              break;
8     case  8: printf("Aprovado\n");
9              break;
10    default: printf("Reprovado\n");
11             break;
12  }
```

- O `switch` só permite comparar expressões com constantes.

O comando switch

Exemplo:

```c
#include <stdio.h>

int main(){

  scanf("%d", &a);
  switch(a) {
    case 10129:
        printf("Maria Cândida Moreira Telles\n");
        break;
    case 33860:
        printf("Larissa Garcia Alfonsi\n");
        break;
    default:
        printf("O aluno não está matriculado\n");
  }

return 0;
}
```

Roteiro

Função sqrt()

A função `sqrt()` é utilizada para calcular a raiz quadrada:

- Faz parte da biblioteca `<math.h>`.

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

```c
double sqrt(double x);
```

  - Um "domain error" ocorre se o argumento for negativo.

Função sqrt()

Exemplo:

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

//função principal
int main(){

  double x = 16.0;

  printf("%lf\n", sqrt(x));

return 0;
}
```

Função sqrt()

Para compilar um programa utilizando a biblioteca `math.h`:

```
1 gcc raiz.c -o raiz -lm
```

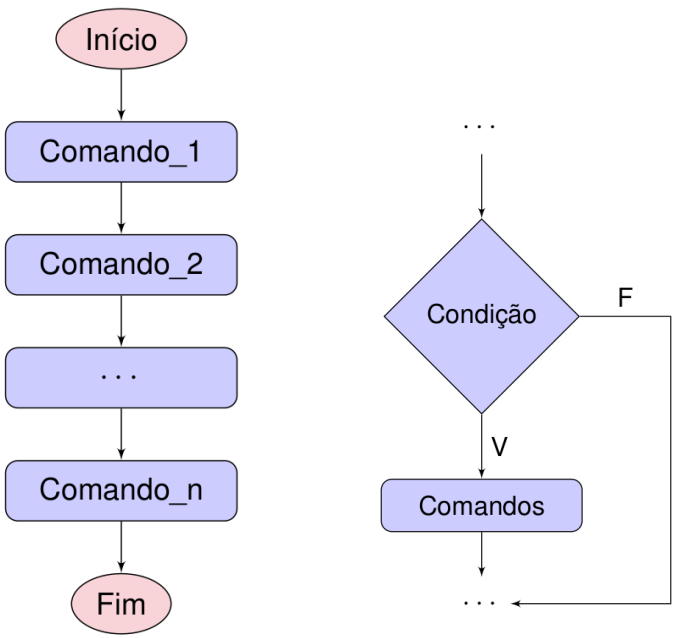
- Existem outras funções: `pow()`, `floor()`, `ceil()`, etc[1]...

[1] https://www.ime.usp.br/~pf/algoritmos/apend/math.h.html

Roteiro

Comandos de Repetição

Até agora vimos como escrever programas que executam comandos de forma linear, e, se necessário, tomam decisões com relação a executar ou não um bloco de comandos.



Comandos Repetitivos

Entretanto, muitas vezes é necessário executar um bloco de comandos várias vezes:

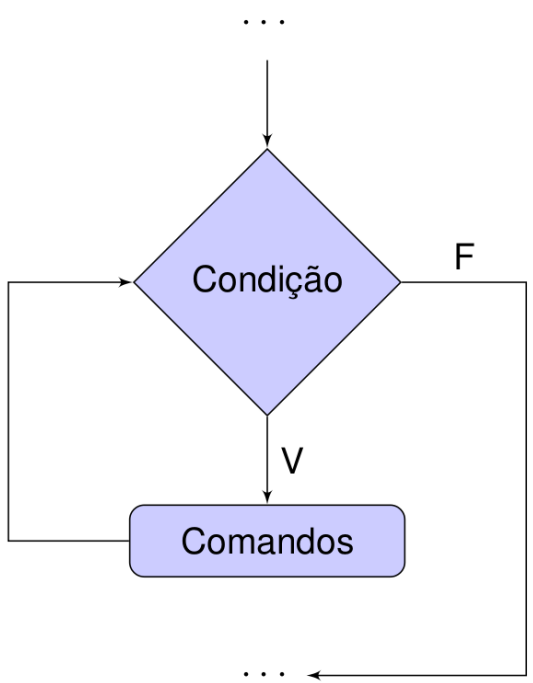
- Exemplo:

```
1  int i = 0;
2  print("%d*7 = %d\n", i, i*7); // 0*7 = 0
3  i++;
4  print("%d*7 = %d\n", i, i*7); // 1*7 = 7
5  i++;
6  print("%d*7 = %d\n", i, i*7); // 2*7 = 14
7  i++;
8  print("%d*7 = %d\n", i, i*7); // 3*7 = 21
9  i++;
10 ...
11 print("%d*7 = %d\n", i, i*7); // 10*7 = 70
```

Comandos Repetitivos

Comandos repetitivos (laços):

- Executam um bloco de instruções enquanto um certa condição for satisfeita.
- A execução repetida de um bloco é chamada de iteração.



Comandos Repetitivos

Laços em C:

- `while`
- `do-while`
- `for`

Roteiro

Comando `while`

Comando `while`:

- Sintaxe:

```
1  while ( condição ){
2      comandos;
3  }
```

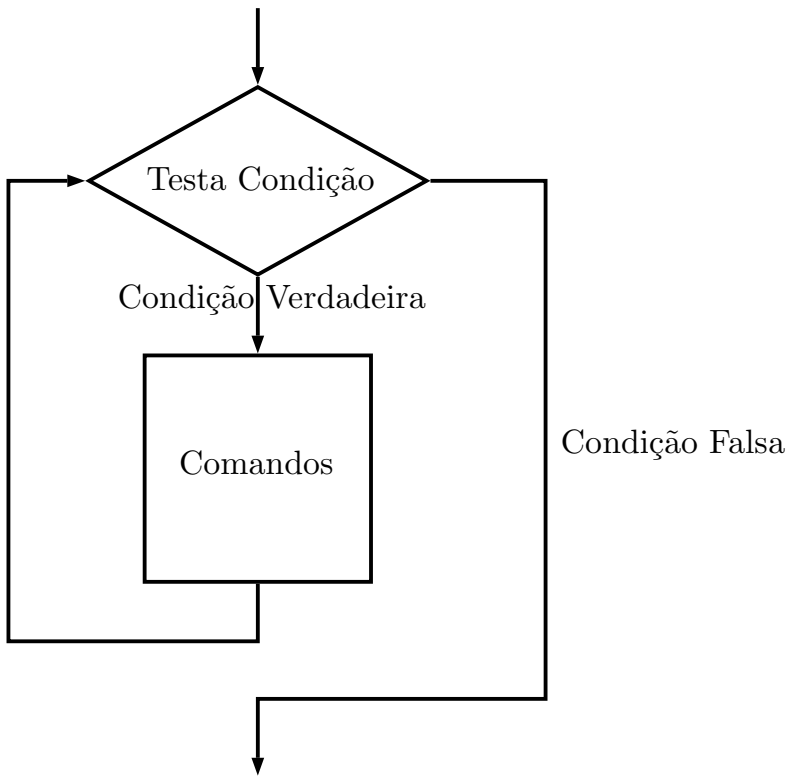
ou:

```
1  while ( condição )
2      comando;
```

  - Enquanto a condição for verdadeira (`!=0`), ele executa o(s) comando(s).

Comando `while`

- Passo 1: Testa a condição. Se for verdadeira vai para o Passo 2.
- Passo 2.1: Executa os comandos.
- Passo 2.2: Volta para o Passo 1.



Comando `while`

Exemplo:

- Imprimindo a tabuada do 7:

```c
#include <stdio.h>

int main(){

  int i = 0;
  while(i <= 10){
    printf("%d\n", i*7);
    i++;
  }

}
```

Comando while

Importante:

- Sempre inicialize a variável de controle:

```
1  #include <stdio.h>
2
3  int main(){
4
5    int i; //pode conter lixo de memória.
6    while(i <= 10){
7      printf("%d*7 = %d\n", i*7);
8      i++;
9    }
10
11 }
```

Comando `while`

O que acontece se a condição for falsa na primeira vez?

```
1  int i = 11;
2  while (i <= 10){
3    printf("%d\n", i*7);
4    i++;
5  }
```

Comando `while`

O que acontece se a condição for sempre verdadeira?

```
1  int i = 11;
2  while (i >= 10){
3    printf("%d\n", i*7);
4    i++;
5  }
```

Roteiro

Comando do-while

Comando do-while:

- Sintaxe:

```
1  do{
2    comandos;
3  } while ( condição );
```

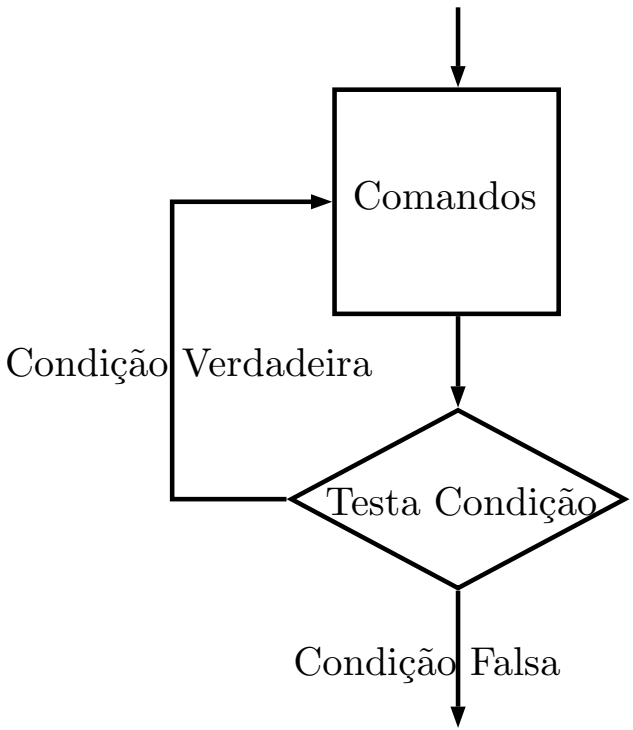
ou:

```
1  do
2    comando;
3  while ( condição );
```

  - Diferença do while: sempre executa comandos na primeira vez. Teste condicional está no final.

Comando `do-while`

- Passo 1: Executa comandos.
- Passo 2: Testa condição. Se for verdadeira vai para Passo 1.



Comando do-while

Exemplo:

- Imprimindo a tabuada do 7:

```c
#include <stdio.h>

int main(){

  int i = 0;
  do{
    printf("%d\n", i*7);
    i++;
  } while(i <= 10) ;

}
```

Roteiro

O comando `for`

O comando `for`

- Sintaxe:

```
1  for (início ; condição ; passo) {
2    comandos;
3  }
```

ou:

```
1 for (início ; condição ; passo)
2     comando;
```

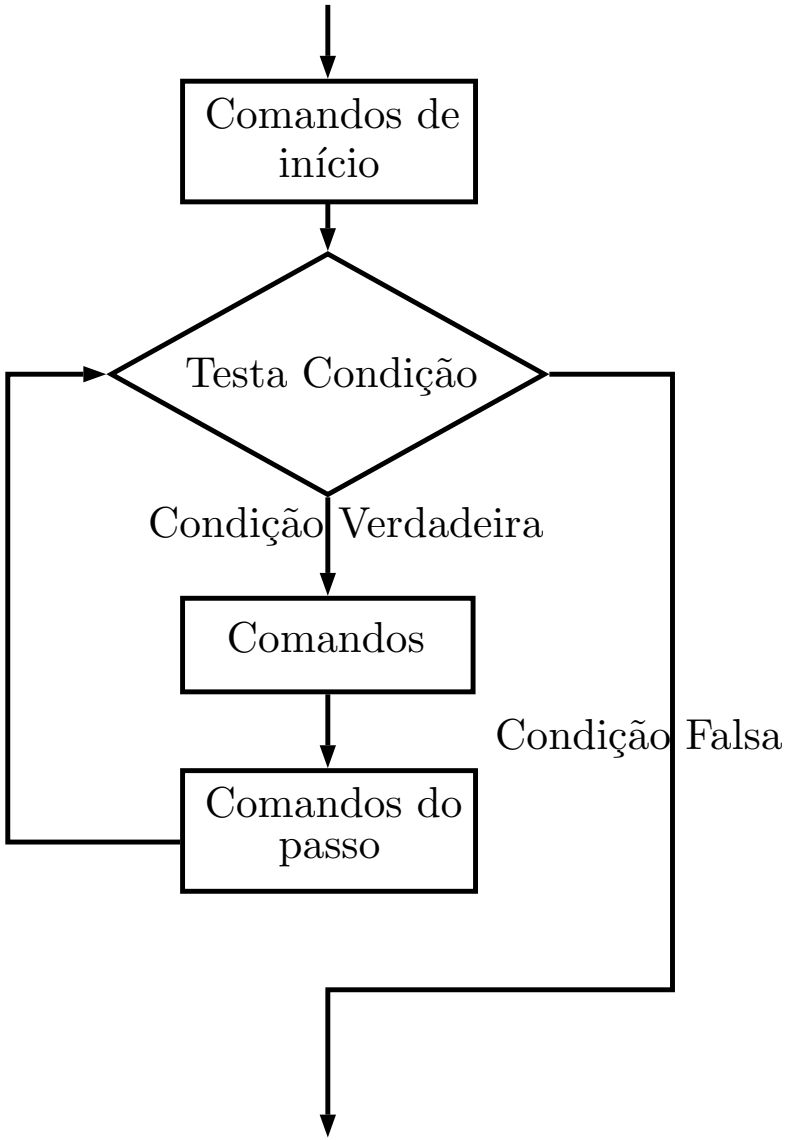
  - **Início**: Uma ou mais atribuições, separadas por ",".
  - **Condição**: Laço executado enquanto a condição for verdadeira.
  - **Passo**: Um ou mais comandos separados por ",".
    Os comandos do passo são executados após os comandos do bloco.

O comando `for`

```
1  for (início ; condição ; passo) {
2    comandos;
3  }
```

- Passo 1: Executa comandos em "início".
- Passo 2: Testa condição. Se for verdadeira vai para passo 3.
- Passo 3.1: Executa comandos do bloco.
- Passo 3.2: Executa comandos em "passo".
- Passo 3.2: Volta ao Passo 2.

O comando `for`



O comando `for`

O `for` é equivalente a seguinte construção utilizando o `while`:

```
1  início;
2  while(condição){
3        comandos;
4        passo;
5  }
```

```
1  for (início ; condição ; passo) {
2    comandos;
3  }
```

Comando `for`

Exemplo:

- Imprimindo a tabuada do 7:

```c
#include <stdio.h>

int main(){

  for(i = 0 ; i <= 10 ; i++){
    printf("%d\n", i*7);
  }

}
```

Roteiro

Soma de números

Exemplo 1:

- Dado uma sequência de entrada $a_1, a_2, \ldots, a_n$, calcule:

$$\sum_{i=1}^{n} a_i$$

Soma de números

- Primeiro, vamos ler quantos números o usuário deseja somar.

```c
#include <stdio.h>

int main(){

  int n;
  printf("Digite a quantidade de números: ");
  scanf("%d", &n);

...
```

- Em seguida, vamos ler os *n* números da entrada com um laço:

```c
  int i, aux;
  for(i=0; i<n; i++){
    scanf("%d", &aux);
    ...
  }
```

Soma de números

- Para somar, vamos criar uma variável acumuladora que a cada iteração acumula a soma de todos os números lidos até então.

```
1  int i, soma=0;
2  for(i=0; i<n; i++){
3    int aux;
4    scanf("%d", &aux);
5    soma = soma + aux;
6  }
```

  - Refaça esse laço com o comando `while`.

Soma de números

```c
#include <stdio.h>

int main(){

  int n;

  printf("Digite a quantidade de números: ");
  scanf("%d", &n);

  int i, soma=0 ;
  for(i=0; i<n; i++){
    int aux;
    scanf("%d", &aux);
    soma = soma + aux;
  }

  printf("Soma é: %d\n", soma);

return 0;
}
```

Calculando potências de 2

Exemplo 2:

- Leia um inteiro positivo `n`, e imprima as potências:

$$2^0, 2^1, \ldots, 2^n$$

Calculando potências de 2

- Primeiro, vamos ler o valor de n.

```c
#include <stdio.h>

int main(){

  int n;
  printf("Digite n: ");
  scanf("%d", &n);

...
```

- Em seguida, vamos realizar um laço com $n + 1$ iterações:

```c
  int i;
  for(i=0; i<=n; i++){ //2^0, 2^1, ..., 2^n
    ...
  }
```

Calculando potências de 2

- Para calcular as potências de 2, vamos criar uma variável acumuladora `pot` que a cada iteração acumula o valor de 2^i, para $i = 0, 1, \ldots, n$.

```
1  int i=0; pot = 1 ; // corresponde a 2^0
2  for(i=0; i<=n; i++){
3    printf("%d\n", pot);
4    pot = pot * 2; // 2^1,...,2^n
5  }
6  ...
```

  - Qual o valor em `pot` depois do laço (linha 6)?

Calculando potências de 2

```
1  #include <stdio.h>
2
3  int main(){
4
5    int n;
6    printf("Digite n: ");
7    scanf("%d", &n);
8
9    int i=0;  pot = 1 ; // corresponde a 2^0
10   for(i=0; i<=n; i++){
11     printf("%d\n", pot);
12     pot = pot * 2; // 2^1,...,2^n
13   }
14
15 return 0;
16 }
```

Calculando o valor de $n!$

Exemplo 3:

- Leia um inteiro positivo `n`, e calcule o fatorial de s:

$$n! = n * (n-1) * (n-2) * \ldots 2 * 1$$

Calculando potências de 2

- Primeiro, vamos ler o valor de n.

```c
#include <stdio.h>

int main(){

  int n;
  printf("Digite n: ");
  scanf("%d", &n);

...
```

- Em seguida, vamos realizar um laço com n iterações:

```c
  int i = 1;
  while(i <= n){ //1,2,...,n
    ...
  }
```

Calculando o valor de $n!$

- Vamos criar uma variável acumuladora que no início da i-ésima iteração de um laço armazena o valor de $(i)!$

```
1  int i = 1, fat = 1;
2
3  while(i <= n){
4    fat = fat * i; //1 * 2 * 3 * ... * (n - 2) * (n - 1) * n
5    i++;
6  }
```

Soma de números

```c
#include <stdio.h>

int main(){

  int n;

  printf("Digite a quantidade de números: ");
  scanf("%d", &n);

  int i = 1, fat = 1;

  while(i <= n){
    fat = fat * i; //1 * 2 * 3 * ... * (n - 2) * (n - 1) * n
    i++;
  }

  printf("Fatorial de %d é: %d\n", n, fat);

return 0;
}
```

Roteiro

O comando `break`

Podemos encerrar a execução de um laço imediatamente:

- Comando `break`:

```
1  for(i = 1; i<= 10; i++){
2    // comando
3    // comando
4    break;
5    // comando
6  }
7  ...
```

O comando break

Exemplo 1:

```
1  int i;
2  for(i = 1; i<= 10; i++){
3    if(i >= 5)
4      break;
5    printf("%d\n",i);
6  }
7  printf("Terminou o laço");
```

- O que será impresso?

O comando break

Exemplo 2:

- Encontrar o primeiro inteiro divisível por 11 e 13:

```
1  int i = 1;
2  while(i < 10000 ){
3    if((i%11 == 0) && (i%13 == 0))
4      break;
5    i++;
6  }
7  printf("%d\n", i);
```

  - Reescreva esse código acima utilizando o comando `for` sem utilizar o `break`.

Roteiro

O comando continue

Podemos alterar o fluxo de execução dentro de um laço para o final.

- Comando continue:

```
1   while(i <= 10){
2     i++;
3     // comando
4     continue;
5     // comando
6     // comando
7   }
8   ...
```

O comando continue

No caso do laço com o comando for:

- Executa o passo, depois testa a condição:

```
1  for(i=0; i <= 10; i++ ){
2    // comando
3    continue;
4    // comando
5    // comando
6  }
7  ...
```

O comando continue

Exemplo:

```
1  int i;
2  for(i = 1; i<= 10; i++){
3    if(i == 5)
4      continue;
5    printf("%d\n",i);
6  }
7  printf("Terminou o laço");
```

- O que será impresso?

O comando continue

Exemplo:

- Imprimir todos os números entre 1 e 100, exceto aqueles divisíveis por 7:

```
1  int i;
2  for(i = 1; i <= 10000; i++){
3    if(i%7 == 0)
4      continue;
5    printf("%d\n",i);
6  }
```

Roteiro

Laços Encaixados

Em alguns casos, é necessário implementar laços dentro de outros laços.

- Estes são conhecidos como laços encaixados/aninhados.

```
1  for(i = 1; i <= 10; i++){
2    // comando A
3    ...
4    for(j = 1; j <= 10; j++){
5      // comando B
6      ...
7    }
8    ...
9  }
```

Laços Encaixados

Importante:

- Cuidado com as variáveis de controle:

```
1  for(i = 1; i <= 10; i++){
2    // comando A
3    ...
4    for(i = 1; i <= 10; i++){
5      // comando B
6      ...
7    }
8    ...
9  }
```

Laços Encaixados

Exemplo 1:

```
1 int main(){
2   int i,j;
3
4   for(i=1; i<=4; i++){
5     for(j=1; j<=3; j++){
6        printf("%d %d\n", i, j);
7     }
8   }
9 }
```

- O que será impresso por este programa?

Laços Encaixados

Exemplo 2:

- Implemente um programa que imprime as tabuadas do $0, 1, \ldots, 10$:

```
1  int main(){
2    int i,j;
3
4    for(i=0; i<=10; i++){
5      for(j=0; j<=10; j++){
6         printf("%d x %d = %d\n", i, j, i*j);
7      }
8      printf("\n");
9    }
10 }
```

Fim

Dúvidas?

Roteiro

Referências

1. Feofiloff, Paulo. Algoritmos em linguagem C. Elsevier Brasil, 2009.
2. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.