Programação Imperativa

Prof. Dr. Alcides Calsavara

Escola Politécnica

PUCPR

Controlando o fluxo de execução:
if, switch, while, do-while, for

Expressões Lógicas

- **Operadores relacionais**

  > maior que
  < menor que
  >= maior ou igual a
  <= menor ou igual a
  = = igual a
  ! = diferente de (não igual a)

- **Operadores Lógicos**

  && (E)
  || (OU)
  ! (NÃO)

Expressões Lógicas

```c
int a = 10;
int b = 30;
int c = 20;

bool resultado1 = a < b;
printf("resultado1: %d\n", resultado1);

bool resultado2 = (a < b) && (b < c);
printf("resultado2: %d\n", resultado2);

bool resultado3 = resultado1 || resultado2;
printf("resultado3: %d\n", resultado3);

bool resultado4 = (a < b || b < c) && a < c;
printf("resultado4: %d\n", resultado4);
```

expressao_logica.c

Comando if

```
if (expressão-lógica)
    comando-1
else
    comando-2
```

Se a avaliação da *expressão lógica* resultar em *verdadeiro*, **então** executa o *comando-1*,
senão executa o *comando-2*.

- A cláusula **else** (*senão*) e correspondente comando (*comando-2*) são opcionais.
- Os comandos (*comando-1* e *comando-2*) podem ser simples (apenas um comando) ou compostos (um bloco de comandos).
- A *expressão lógica* pode ser uma expressão aritmética: o resultado no valor 0 (zero) equivale a falso, enquanto outro resultado qualquer equivale a verdadeiro.

Exemplo

```c
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    printf("a < b\n");
else
    printf("a >= b\n");

if (a < b && b < c)
{
    printf("a < b < c");
}
else
{
    a = a * 2;
    b = c - a;
    c = c / 2;
}

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if.c

Exemplo

```c
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    printf("a < b\n");
else
    printf("a >= b\n");

if (a < b && b < c)
{
    printf("a < b < c");
}
else
{
    a = a * 2;
    b = c - a;
    c = c / 2;
}

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if.c

Exemplo

```
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    if (b < c)
        printf("a < b < c\n");
    else
        printf("a < b >= c\n");
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_2.c

Exemplo

```
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    if (b < c)
        printf("a < b < c\n");
    else
        printf("a < b >= c\n");
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_2.c

Exemplo

```
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    if (b < c)
        printf("a < b < c\n");
    else
        printf("a < b >= c\n");
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_2.c

Certo?

```
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    if (b < c)
        printf("a < b < c\n");
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_3.c

Exemplo

```c
int a = 10;
int b = 30;
int c = 20;

if (a < b)
    if (b < c)
        printf("a < b < c\n");
    else
        printf("a < b >= c\n");
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_2.c

Certo!

```c
int a = 10;
int b = 30;
int c = 20;

if (a < b)
{
    if (b < c)
        printf("a < b < c\n");
}
else
    printf("a >= b\n");

printf("a, b, c: %d %d %d\n", a, b, c);
```

comando_if_4.c

Exemplo

```
if (1)
    printf("verdadeiro\n");
else
    printf("falso");
```

```
int a = 10, b = 5;

if (a * b)
    printf("verdadeiro\n");
else
    printf("falso");
```

```
double x = 10.0, y = 5.0;

if (x - 2 * y)
    printf("verdadeiro\n");
else
    printf("falso");
```

comando_if_5.c

Exercícios

(5.1) Escreva um programa na linguagem C que, dados dois números inteiros distintos fornecidos pelo usuário (via teclado), imprima-os em ordem crescente.
Obs: o programa deve certificar-se de que os dois números sejam distintos.

(5.2) Escreva um programa na linguagem C que, dadas três letras distintas fornecidas pelo usuário (via teclado), imprima-as em ordem alfabética.
Obs: o programa deve certificar-se de que as três letras sejam distintas.

Comando switch

Sintaxe:

```
switch (expressão)
{
    case x: sequência de comandos
    case y: sequência de comandos
    ...
    default: sequência de comandos
}
```

Desvia a execução para o **case** correspondente ao resultado da *expressão*.

- A avaliação da expressão deve resultar num valor inteiro.
- Os rótulos (x, y, ...) nas cláusulas **case** devem ser expressões constantes.
- A cláusula **default** é opcional.
- Uma sequência de comandos é interrompida somente por meio do comando **break**, se existir.

Exemplo

```c
#include <stdio.h>

int main()
{
    int k;
    scanf("%d", &k);
    switch ( k )
    {
        case 1: printf("UM"); break;
        case 2: printf("DOIS"); break;
        case 3: printf("TRÊS"); break;
        default: printf("DESCONHECIDO");
    }

    return 0;
}
```

switch_int.c

Exemplo

```
bool vogal;

char letra;
printf("Digite um letra: ");
letra = getchar();

switch ( letra )
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u': vogal = true; break;
    default:  vogal = false;
}

if (vogal)
    printf("vogal\n");
else
    printf("consoante");
```

switch_char.c

Exercício 5.3

Elabore um programa em C para simular uma calculadora com as quatro operações básicas (adição, subtração, multiplicação e divisão), usando a estrutura *switch.* O programa deve receber, como entrada, dois operandos (*double*) e um operador (*char*). Como saída, o programa deve exibir o resultado da aplicação do operador sobre os dois operandos.

Comando while

```
while (expressão-lógica)

    comando
```

Enquanto a avaliação da *expressão lógica* resulta em *verdadeiro*,
executa o *comando*.

- O *comando* pode ser simples ou composto.
- A *expressão lógica* pode ser uma expressão aritmética.

Exemplos

```c
#include <stdio.h>

int main()
{
    int x = 0;

    while (x < 10)
        x++; // x = x + 1

    printf("x: %d\n", x);

    return 0;
}
```

comando_while_1.c

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

int main()
{
    int x = 0;
    bool rodando = true;

    while (x < 10 && rodando)
    {
        if (x == 7) rodando = false;
        x++; // x = x + 1
    }

    printf("x: %d\n", x);

    return 0;
}
```

comando_while_2.c

Exemplos

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

int main()
{
    int x = 0;
    bool rodando = true;

    while (rodando)
    {
        x++;
        printf("x: %d\n", x);
    }

    return 0;
}
```

comando_while_3.c

```c
#include <stdio.h>


int main()
{
    int x = 0;


    while ( 1 )
    {
        x++;
        printf("x: %d\n", x);
    }

    return 0;
}
```

comando_while_4.c

Exercício 5.4

Escreva um programa em C que seja equivalente ao programa em Python ao lado.

```python
# Python
# Cálculo do volume de uma esfera de raio R

PI = 3.1416
R = 0
while R <= 6:
    VOLUME = 4/3 * PI * (R**3)
    print(R, VOLUME)
    R =  R + 2
```

Sugestão: use a função **pow** da biblioteca **math** para fazer potenciação.

```c
#include <math.h>

pow(R, 3) // calcula R ao cubo
```

Atenção: em C, 4/3 é diferente de 4.0/3

Compare os valores de x e y em:

```c
double x = (4/3) * 10.0
double y = (4.0/3) * 10.0
```

Comando *for*

```
for (expressão-inicial ; expressão-lógica ; expressão-incremento)
  comando
```

Repete a execução do *comando* **enquanto** a *expressão-lógica* for verdadeira.

`expressão-inicial` – executada apenas uma vez, no início do comando **for**. Pode ser uma série de expressões separadas por vírgulas.

`expressão-lógica` – expressão lógica avaliada a cada iteração, **antes** da execução do *comando*, para decidir se a repetição continua: no caso de ser avaliado "falso", indica o fim da repetição.

`expressão-incremento` – executada a cada iteração, **após** a execução do *comando*. Pode ser uma série de expressões separadas por vírgulas.

Exemplo do comando *for*

```c
for  (int i = 0; i < 10; i++)
{
    printf("%d\n", i);
}
```

for_basico.c

Exemplo do comando *for*

```c
for  (int i = 0; i < 10; i++)
{
    printf("%d\n", i);
}
```

for_basico.c

```python
for i in range(0, 10):
  print( i )
```

Equivalente em Python

Exemplo do comando *for*

```c
for  (int i = 0, j = 10; i != j; i++, j--)
{
    printf("%d %d\n", i, j);
}
```

for_multi.c

Exemplo do comando *for*

```c
for  (int i = 0, j = 10; i != j; )
{
    printf("%d %d\n", i, j);
    i++;
    j--;
}
```

for_vazio.c

Exemplo do comando *for*

```c
for  (int i = 0, j = 10; ; )
{
    printf("%d %d\n", i, j);
    i++;
    j--;
}
```

for_infinito.c

Exemplo do comando *for*

```c
int i = 0, j = 10;

for  ( ; ; )
{
    printf("%d %d\n", i, j);
    i++;
    j--;
}
```

for_todo_vazio.c

Exercício 5.5

Converta o programa em Python ao lado para um programa em C, usando o comando *for*

```
# Python
PI = 3.1416
R  =  0
while R <= 6:
    VOLUME = 4/3 * PI * (R**3)
    print(R, VOLUME)
    R =  R + 2
```

Comando *do-while*

```
do
    comando
while (expressão-lógica);
```

A execução ocorre do seguinte modo:

1) O comando é executado.

2) A expressão lógica é avaliada:

   a) Se resultar “verdadeiro”, o fluxo de execução volta para o item 1.

   b) Se resultar “falso”, a iteração termina e o fluxo de execução segue para o primeiro comando após o do-while

Exemplo do comando *do-while*

```c
#include <stdio.h>

int main()
{
    int x = 0;
    do
        x++;
    while (x < 10);

    printf("%d\n", x);
}
```

comando_dowhile.c

Exemplo do comando *do-while*

```c
int i = 1;
do
{
    printf("i = %d\n", i);
    i++;
}
while(i < 10);
```

dowhile_basico.c

Exemplo do comando *do-while*

```c
// código usando do-while:
char letra = 'a';
do
{
    putchar(letra);
    letra++;
}
while (letra <= 'z');
```

```c
// código usando while:
char letra = 'a';
while (letra <= 'z')
{
    putchar(letra);
    letra++;
}
```

dowhile_alfabeto.c

Exemplo do comando *do-while*

```c
int k; // um valor entre 10 e 20, inclusive

do
{
    printf("Digite um valor inteiro entre 10 e 20: ");
    scanf("%d", &k);
}
while (k < 10 || k > 20);

printf("Valor digitado: %d\n", k);
```

dowhile_consistencia_leitura_int.c

Exercício 5.6

Converta o programa em Python ao lado para um programa em C, usando *do-while*.

```
# Python
PI = 3.1416
R  =  0
while R <= 6:
    VOLUME = 4/3 * PI * (R**3)
    print(R, VOLUME)
    R =  R + 2
```

Comandos *break* e *continue*

Pode ser conveniente fazer o controle de execução de uma repetição independentemente do teste lógico efetuado.

- O *break*, quando utilizado dentro do bloco de comandos de um “loop”, (*for*, *while* ou *do-while*) faz com que o “loop” seja imediatamente interrompido, transferindo o fluxo de execução para o próximo comando após o “loop”.

- O *continue* desvia o fluxo de execução para a próxima iteração, ou seja, para o teste de parada do “loop".

Exemplo do comando *break*

```c
int i = 0;
do
{
    i++;
    if  ( (i % 4) == 0 )
        break;
    printf("i = %d\n", i);
}
while (i < 10);
```

break_dowhile.c

Exemplo do comando *continue*

```c
int i = 0;
do
{
    i++;
    if  ( (i % 2) == 0 )
        continue;
    printf("i = %d\n", i);
}
while (i < 10);
```

continue_dowhile.c

Exercício 5.7

Converta o programa em Python ao lado para um programa em C, usando *while* e *break*.

```
# Python
i=10
while i > 0 :
      i = i - 0.25
      print (i)
      if (i**2) + 1 ≥ 1.45 :
            i = i + 0,20;
      else
            break
```

Exercícios

(5.8) Sendo H = 1 + 1/2 + 1/3 + ¼ + ... 1/N, elaborar um programa. em C para gerar o número H. O valor de N deverá ser fornecido pelo usuário. Use a estrutura **for** para somar a sequência de termos.

(5.9) Escreva um programa em C para calcular o fatorial de um número N fornecido pelo usuário. Use a estrutura **for** para gerar a sequência de termos.

(5.10) Fazer um programa em C para calcular o valor da série S abaixo. O valor de N deve ser fornecido pelo usuário. Use a estrutura **do-while** para somar a sequência de termos.

S = 1/N + 2/N-1 + 3/N-2 + ...+ N-1/2 + N/1

(5.11) Elabore um programa em C para imprimir os 30 primeiros números naturais pares. Use a estrutura **for** para gerar a sequência de termos.