Linguagem C

Estruturas de Controle: Repetições

MsC. Douglas Santiago Kridi

Programação I - 2018.2

Bacharelado em Ciência da Computação

Universidade Estadual do Piauí

douglaskridi@gmail.com

Introdução

- Anteriormente, vimos estruturas condicionais como, *if...else* e *switch*, as quais permitem executar parte do código do programa somente sob determinadas condições.
- Agora, veremos estruturas que além de uma condição, atrelam a repetição de código.

Introdução

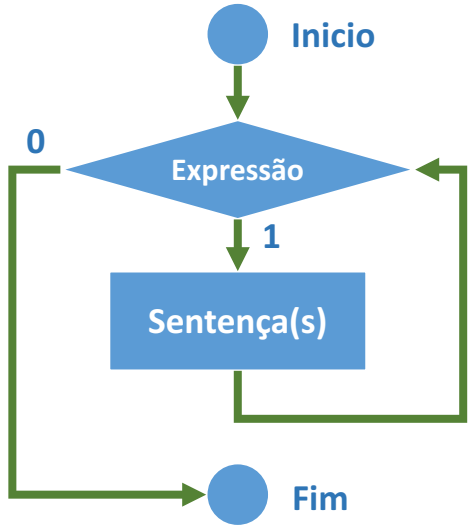
- As estruturas de repetição, permitem executar *mais de uma vez* um mesmo trecho de código.
- Trata-se de uma forma de executar blocos de comandos somente sob determinadas *condições*, mas com a opção de ***repetir*** o mesmo bloco quantas vezes for necessário.

Introdução

- As estruturas de repetição são úteis, por exemplo, para repetir uma série de *operações semelhantes* que são executadas para todos os elementos de uma lista ou de uma tabela de dados.
  - Ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.

Estrutura *While*

- O ***while*** (enquanto) é a estrutura de repetição mais simples.
- Ele repete a execução de um bloco de sentenças enquanto uma condição permanecer verdadeira.




```
while (expressão) {
        sentença;
        sentença;
        ...
}
```

- Na primeira vez que a condição se tornar falsa, o *while* não repetirá a execução do bloco, e continuará com a sentença ou comando que vem logo após o bloco do while.

Estrutura *While*

- A *expressão* é uma condição que controla o while.
- Ela utiliza os mesmos operadores relacionais e lógicos vistos em estruturas condicionais.

```
while (expressão) {
        sentença;
        sentença;
        ...
}
```

- Primeiro, o programa avalia a expressão.
- Caso o resultado da expressão seja não nulo (verdadeiro), então todo o bloco de sentenças será executado.
- Em seguida, o programa volta a avaliar a expressão e o processo se repete até que a expressão retorne como zero (falso).

- A expressão é sempre avaliada antes da decisão de se executar o bloco de sentenças novamente.

Estrutura *While*

- Para o uso correto do *while*, o bloco de sentenças precisa *modificar o estado do sistema* de forma a afetar justamente as características testadas na expressão.
  - Se isto não ocorrer, então o *while* executará eternamente.
    - Um loop infinito.

Estrutura *While*

- Exemplo 1: Imprimir os números de 1 a 10:

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

1. A expressão do while verifica se o número está dentro do limite desejado (menor ou igual a 10).
2. No início, o valor da variável numero é 1 e portanto a condição do while é satisfeita.
3. Nas próximas repetições, a condição será verdadeira para os valores de numero 2, 3, 4, ... 9 e 10.
4. Quando numero armazenar o valor 11, condição será falsa. Nesse ponto, o while termina, encerrando as repetições.

Estrutura *While*

- **Exemplo 2:** O programa imprime todos os divisores de um número inteiro positivo. Para o número n dado, o programa verifica se cada número de 1 até n é ou não um divisor de n.

```
int numero;
int divisor;
int resto;

printf("Digite o numero: ");
scanf("%d", &numero);

divisor = 1;
while (divisor <= numero) {
        resto = numero % divisor;
        if (resto == 0) {
                printf("Divisor encontrado: %d \n", divisor);
        }
        divisor = divisor + 1;
}
```

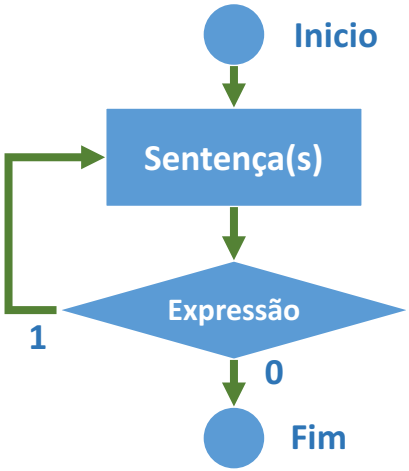
Observe que o bloco precisa modificar o valor da variável divisor para, em algum momento, parar as repetições.

Estrutura *While*

- Pratique:
  - Escreva um aplicativo em C que mostra todos os números ímpares de 1 até 100.
  - Escreva um aplicativo em C que recebe um inteiro e mostra os números pares e ímpares (separados, em duas colunas), de 1 até esse inteiro.

Estrutura: *do ... while*

- O do ... while, tem um comportamento muito semelhante ao while, com uma diferença crucial: *a condição é verificada **após** executar o bloco de instruções correspondente*.




```
do {
        sentença;
        sentença;
        ...
} while (expressão);
```

- Executa-se o bloco de sentenças, independentemente da condição. Somente então a expressão é avaliada.
- Caso ela seja não nula (***verdadeira***), então todo o bloco de sentenças será executado novamente. Este processo se repete até que a expressão resulte em zero (***falso***). A expressão é sempre avaliada depois da execução do bloco de sentenças.

Estrutura: *do ... while*

- Exemplo 1: Imprimir números de 1 a 10:

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

1. A expressão do *do...while* verifica se o número está dentro do limite desejado (menor ou igual a 10).
2. O bloco de sentenças é executado, imprimindo o valor de *numero* e aumentado seu valor em uma unidade.
3. Após executar o bloco, a expressão verifica se a variável *numero* continua dentro do limite permitido. Caso afirmativo, o bloco é executado novamente.
4. No final da execução, o valor da variável *numero* será 11, impedindo uma nova execução do bloco do...while.

Estrutura: *do ... while*

- Pratique:
  - Construa um programa que imprime a soma de todos os valores positivos digitados pelo usuário até que ele digite um número negativo.

Operadores de incremento

- As estruturas de repetição utilizam variáveis para controlar o número de repetições.
  - No exemplo de imprimir números de 1 até 10, no final de cada iteração temos:

$$numero = numero + 1;$$

- Como este tipo de atribuição é muito frequente, a linguagem C oferece atalhos que podem ser práticos em estruturas de repetição:

Objetivo	Atalho	Forma original
Somar uma unidade ao valor da variável	++numero;	numero = numero + 1;
Subtrair uma unidade do valor da variável	--numero;	numero = numero – 1;

Operadores de incremento

- Exemplos: imprimindo números de 1 até 10:

while

```
int numero = 1;
While (numero <= 10){
        printf("%d\n" , numero);
        ++numero;
}
```

do ... while

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

Operadores de incremento

- Existem comandos semelhantes aos anteriores, mas que retornam os valores que estavam armazenados nas variáveis ***antes*** de se realizar as *operações de incremento ou de decremento.*
  - *Logo, temos situações de pré/pós incremento/decremento.*

Objetivo	Atalho	Forma original
Somar um ao valor da variável, retornando o valor original	numero++;	(retorne numero) numero = numero + 1;
Subtrair um do valor da variável, retornando o valor original	numero--;	(retorne numero) numero = numero – 1;

Operadores de incremento

Exemplo: Considere o código apresentado a seguir.

```
int var1 = 10;
int var2 = 20;
int res = 0;
res = var1 + var2;
printf("res: %d”, res);
res = var1++ + var2;
printf("res: %d”,res);
res = var1 + var2;
printf("res: %d”, res);
res = var1 + --var2;
printf("res: %d”, res);
```

a)res: 30
res: 29
res: 30
res: 29

b)res: 30
res: 32
res: 33
res: 32

c)res: 30
res: 28
res: 29
res: 28

d)res: 30
res: 30
res: 31
res: 30

e)res: 30
res: 31
res: 30
res: 31

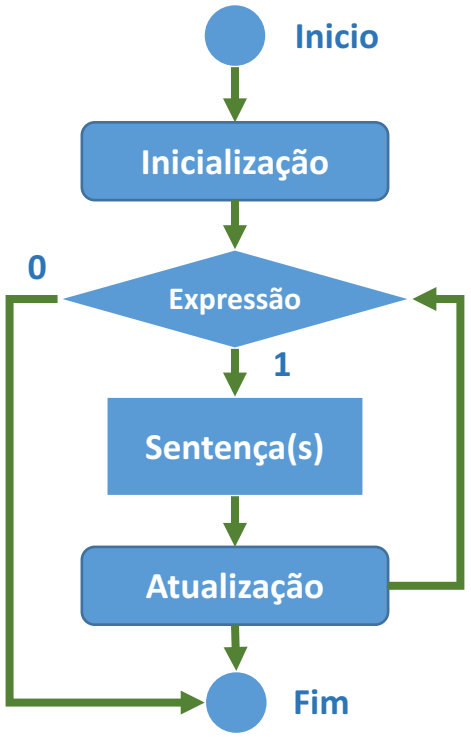
Operadores de incremento

- Casos possíveis com operadores aritméticos:

Objetivo	Atalho	Forma original
Somar k unidades ao valor da variável	numero += k;	numero = numero + k;
Subtrair k unidades do valor da variável	numero -= k;	numero = numero - k;
Multiplicar o valor da variável por k	numero *= k;	numero = numero * k;
Dividir o valor da variável por k	numero /= k;	numero = numero / k;

Estrutura: *for*

- As estruturas que vimos anteriormente, sempre apresentam uma variável contadora e as quatro etapas seguintes: Inicialização, Teste, Execução e Atualização:
  - Um *for*, apresenta de forma compacta, as etapas de inicialização, teste e atualização:




```
for (inicialização; teste; atualização) {
        sentença;
        sentença;
        ...
}
```

Estrutura: *for*

- Exemplo 1: imprimindo números de 1 a 10:

```
int numero;
for (numero = 1; numero <= 10; numero++) {
        printf("%d ", numero);
}
```

- Exemplo 2: imprimindo números de 1 ate 20 de 2 em 2:

```
int numero;
for (numero = 1; numero <= 20; numero += 2) {
        printf("%d ", numero);
}
```

- Exemplo 3: imprimindo números de 10 até 1:

```
int numero;
for (numero = 10; numero >= 1; numero--) {
        printf("%d ", numero);
}
```

Break e Continue

- O comando ***break*** termina imediatamente a execução de um bloco controlado por uma repetição, sem esperar a próxima avaliação da condição.
- O comando ***continue*** reinicia imediatamente a execução de um bloco de uma estrutura de repetição. O continue não espera o término da execução do restante do bloco.
  - O uso dos comandos continue e break, costuma estar associado com uma estrutura condicional if para que a repetição seja reiniciada, ou interrompida, somente sob determinadas condições.

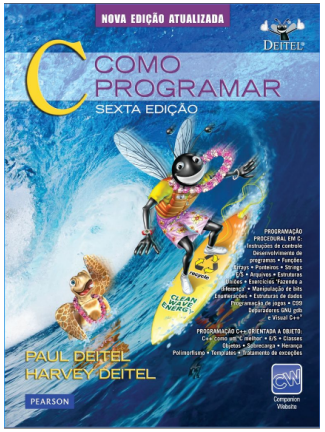
Estrutura: *for*

- Pratique:
  - Escreva um programa em C que solicita 10 números ao usuário, através de um laço for, e ao final mostre qual destes números é o maior.
  - Escreva um programa que lê o tamanho do lado de um quadrado e imprime um quadrado daquele tamanho com asteriscos. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20.

    Por exemplo, para lado igual a 5:

```
*****
*****
*****
*****
*****
```

Referências Bibliográficas



C: COMO PROGRAMAR

DEITEL, Harvey M.; DEITEL, Paul J. Editora Pearson - 6ª ed. 2011

Fundamentos da Programação de Computadores

Ascencio, Ana F. G., Campos, Edilene A. V. de, - Editora Pearson 2012

Lógica de Programação e Estrutura de Dados

Puga, Sandra. Risseti, Gerson. – Ed. Pearson - 2016

Lógica de Programação Algorítmica (Apostila)

Guedes, Sergio. - Editora Pearson/Ser - 2014