Estruturas de Repetição

Disciplina: Algoritmos e Programação
Curso: Engenharia de Computação

Professora: Mariza Miola Dosciatti
mariza@utfpr.edu.br

Objetivos

- Entender o conceito de estruturas de repetição.

- Item da ementa (Plano de Ensino):
  - Estruturas de controle de fluxo: repetição

Estruturas de repetição

- Permitem que um conjunto de instruções seja repetido um determinado número de vezes, até que uma condição se estabeleça ou enquanto uma condição permanecer atendida.
- A finalização da execução pode ser previamente determinada ou ser decorrente de condições serem atendidas durante a execução do programa.
- O controle das execuções está vinculado a **condições de controle.**
- Uma **condição de controle** está relacionada a **operadores lógicos e relacionais**. Ela é resultado de um teste lógico.

Estruturas de repetição (cont.)

• Uma estrutura de repetição **(*loop)*** se refere a um conjunto de instruções que são executadas repetidamente enquanto alguma condição de continuidade dessa repetição permanecer verdadeira.

• A repetição pode:

- Ter o número de repetições conhecido antecipadamente.
- A condição de finalização das repetições é gerada pelo processamento das instruções que compõem a estrutura de repetição.

Estruturas de repetição (cont.)

- As três estruturas de repetição da **Linguagem C** são:
  1. Estrutura de repetição para um número definido de repetições (iterações):
     - **for**
  2. Estrutura de repetição para número indefinido de repetições (iterações):
     - **while** (com teste (condição) no início)
     - **do ... while** (com teste (condição) no final)

Estruturas de repetição (cont.)

- **for (PARA ... ATÉ ... FAÇA)**
  - A estrutura ***for*** é normalmente utilizada quando o número de vezes que o conjunto de instruções será executado é predefinido.
  - A estrutura ***for*** é conhecida como repetição controlada por contador.

Estruturas de repetição (cont.)

- **Sintaxe *for*:**

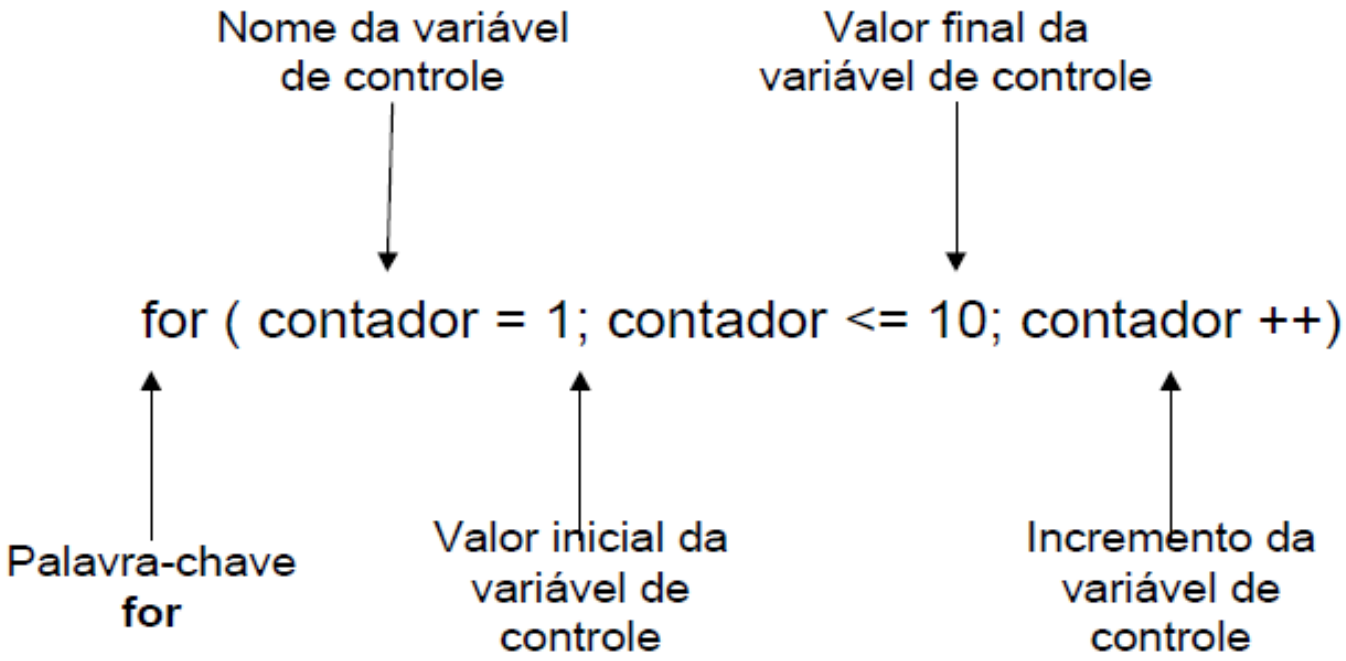
```
for(inicialização; condição; incremento/decremento)
{
        //instruções
}
```

onde:

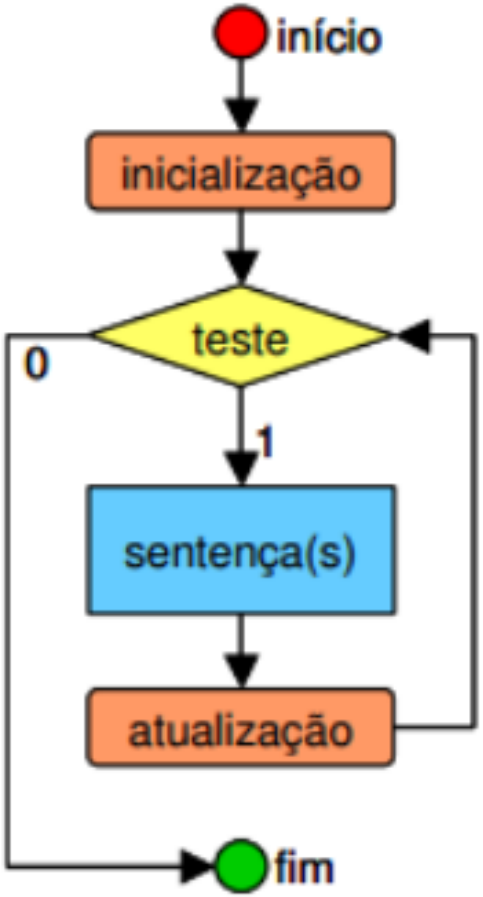
- ***inicialização*** é uma expressão que determina o início para a expressão de controle. Usa um contador de iterações, que é inicializado antes da primeira iteração.
- ***condição*** é uma expressão lógica que verifica se a condição de finalização foi alcançada. É o teste para determinar se o conjunto de instruções será executado novamente.
- ***incremento*** é uma expressão para armazenar a quantidade de iterações realizadas. Pode ser também **decremento**, que define quantas iterações faltam.
- ***conjunto de instruções*** são os comandos a serem executados.

Estruturas de repetição (cont.)

- **Componentes de um cabeçalho típico da estrutura *for*:**



Estruturas de repetição (cont.)



- Uma repetição controlada por contador necessita de:
  - Variável de controle (teste lógico).
  - O *valor inicial* da variável de controle.
  - O *incremento* (ou *decremento) que modifica* a variável a cada iteração.
  - A forma de verificação se a condição foi alcançada.

Estruturas de repetição (cont.)

- **Exemplo:** Imprime os números de 0 a 10.

```
#include <stdio.h>

int main(void)
{
      int i;

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

Estruturas de repetição (cont.)

- **Outro exemplo:** Números pares e ímpares.

```
#include <stdio.h>

int main(void)
{
        int num, i;

        for(i=1; i<=10; i++)
        {
                printf("Informe um numero: ");
                scanf("%d", &num);

                if(num%2 == 0)
                {
                        printf("O numero eh par\n");
                }
                else
                {
                        printf("O numero eh impar\n");
                }
        }

        return 0;
}
```

Estruturas de repetição (cont.)

- **Outro exemplo:** Tabuada.

```c
#include <stdio.h>

int main(void)
{
      int num, i;

      printf("Informe um numero: ");
      scanf("%d", &num);

      for(i=0; i <= 10; i++)
      {
          printf("%2d * %2d = %2d\n", num, i, num*i);
      }
      return 0;
}
```

Estruturas de repetição (cont.)

- **Observações:**
  - A **inicialização**, a **condição de repetição** do loop e o **incremento** podem conter expressões aritméticas.

    Por exemplo, com x = 2 e y = 10

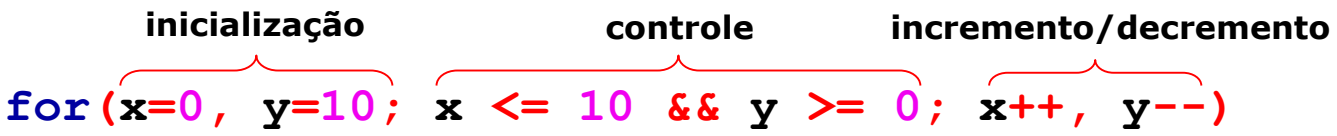
```
for(i=x; i<=4*x*y; i+=y/x)
```

  - O "incremento" pode ser negativo. É um decremento ou uma contagem regressiva.
  - Se a condição de repetição for inicialmente falsa, as instruções que compõem a estrutura não serão realizadas.
  - É comum que a variável de controle seja impressa e/ou usada em cálculos pelas instruções que compõem a estrutura. Contudo, essa variável pode ser utilizada apenas para contar e controlar a quantidade de iterações.

Estruturas de repetição (cont.)

- **Incremento e decremento no mesmo laço *for***
  - A inicialização, a condição de repetição do loop e o incremento/decremento podem ser compostos.

    Exemplo:



    - Possui duas variáveis inicializadas;
    - A condição de controle é composta;
    - Possui uma variável incrementada e outra decrementada.

Estruturas de repetição (cont.)

- **Exemplo:** Incremento e decremento.

```
#include <stdio.h>

int main(void)
{
    int x, y;

    for(x=0, y=10; x<=10 && y>=0; x++, y--)
    {
        printf("x = %d\t y = %d\n", x, y);
    }

    return 0;
}
```

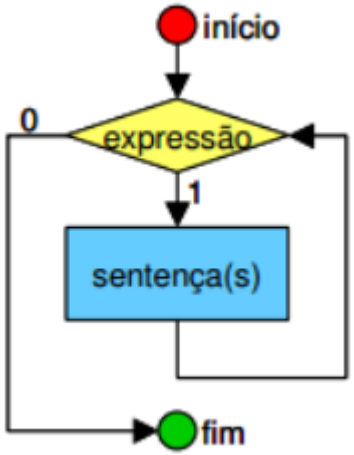
Estruturas de repetição (cont.)

- **Estrutura *while***
  - É a estrutura de repetição mais simples.
  - Repete a execução de um bloco de instruções enquanto uma condição for verdadeira.
  - Quando a condição se tornar falsa, o ***while*** não repetirá a execução do instruções.

Estruturas de repetição (cont.)

- **Sintaxe *while***

```
while(condição)
{
    //instruções
}
```



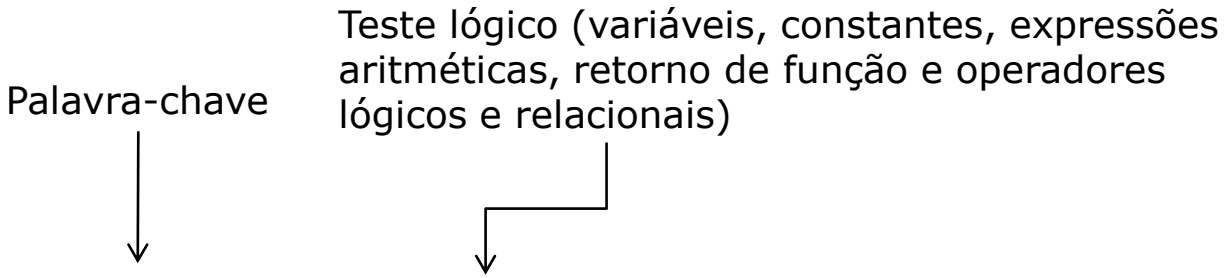
Onde:

- condição – teste lógico realizado. É o controle para indicar a finalização da repetição.
- instruções – comandos que são executados enquanto o resultado da condição for **verdadeiro**. Inclui a alteração da condição para que em algum momento ela se torne falsa e a repetição possa ser interrompida.

- Esta estrutura faz com que a condição seja avaliada e, se **verdadeira,** o conjunto de instruções é executado. Esse ciclo se repete até a condição ser **falsa** e, neste caso, a repetição é terminada sem executar o conjunto de instruções.

Estruturas de repetição (cont.)

- ***while***




```
while(Condição)
{
      /*instruções, incluindo forma de
        atingir a condição.*/
}
```

Estruturas de repetição (cont.)

- **Exemplo:** Imprime números de 0 a 10

```
#include <stdio.h>

int main(void)
{
      int i = 0;

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

      return 0;
}
```

Estruturas de repetição (cont.)

- **Exemplo:** Pares e ímpares

```c
#include <stdio.h>
int main(void)
{
    int num, i=0;

    while(i < 10)
    {
        printf("Informe um numero: ");
        scanf("%d", &num);

        if(num%2 == 0)
        {
            printf("O numero eh par\n");
        }
        else
        {
            printf("O numero eh impar\n");
        }
        i++;
    }
    return 0;
}
```

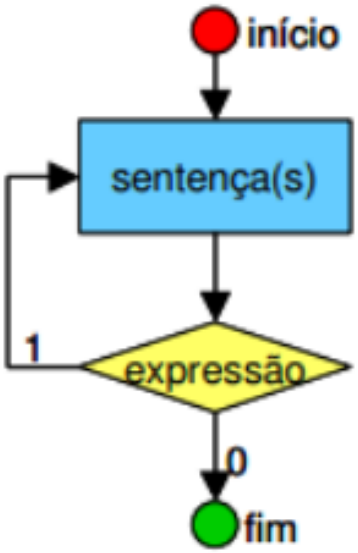
Estruturas de repetição (cont.)

- **Exemplo:** Tabuada

```
#include <stdio.h>

int main(void)
{
    int num, i=0;

    printf("Informe um numero: ");
    scanf("%d", &num);

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

    return 0;

}
```

Estruturas de repetição (cont.)

- **Estrutura *do - while***
  - Esta estrutura tem um comportamento muito semelhante ao ***while***, com uma diferença crucial: a condição é verificada **<u>após</u>** executar o bloco de instruções.



Estruturas de repetição (cont.)

- **Sintaxe *do – while***

```
do
{
      //instruções;

}while(condições);
```

- Onde:
  - condição – teste lógico realizado - é o controle para indicar a finalização da repetição;
  - instruções – comandos que são executados enquanto o resultado da condição for **verdadeiro**.
  - Instruções inclui a alteração da condição para que em algum momento ela se torne falsa e a repetição (loop) possa ser interrompida.

- Esta estrutura faz com que o conjunto de instruções seja executado pelo menos uma vez. Após a execução desse conjunto, a condição é avaliada. Se for **verdadeira,** o conjunto de instruções é executado outra vez, caso contrário a repetição é finalizada.

Estruturas de repetição (cont.)

- **Sintaxe *do – while***

Palavra-chave

```
do
{
     /*instruções, incluindo forma de atingir a
       condição.*/
}while(Condição);
```

Palavra-chave Teste lógico (variáveis, constantes, expressões aritméticas, retorno de função e operadores lógicos e relacionais)

Estruturas de repetição (cont.)

- **Exemplo:** Imprime números de 0 a 10

```
#include <stdio.h>

int main(void)
{
    int i=0;

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

    return 0;
}
```

Estruturas de repetição (cont.)

- **Outro exemplo :** Lê número positivo.

```c
#include <stdio.h>

int main(void)
{
    int num;
    do
    {
        printf("Digite um numero positivo (0 para sair): ");
        scanf("%d", &num);

        if(num<=0)
        {
            printf("Numero invalido\n");
        }
    }while(num <= 0);

    return 0;
}
```

Estruturas de repetição (cont.)

- **Exemplo:** Pares e ímpares

```
#include <stdio.h>

int main(void)
{
    int num, i=0;

    do
    {
       printf("Informe um numero: ");
       scanf("%d", &num);

       if (num%2 == 0)
       {
           printf("O numero eh par\n");
       }
       else
       {
           printf("O numero eh impar\n");
       }
       i++;
   }while(i < 10);

   return 0;
}
```

Estruturas de repetição (cont.)

- **Exemplo:** Tabuada

```
#include <stdio.h>

int main(void)
{
      int num, i=0;

      printf("Informe um numero: ");
      scanf("%d", &num);

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

      return 0;
}
```

Estruturas de repetição (cont.)

```c
#include <stdio.h>

int main(void)
{
    int i;
    printf("Usando o comando \"for\"\n");
    for (i=1; i<=5; i++)
    {
        printf("%d\n", i);
    }

    printf("\nUsando o comando \"while\"\n");
    i=1;
    while (i<=5)
    {
        printf("%d\n", i);
        i++;
    }

    printf("\nUsando o comando \"do while\"\n");
    i=1;
    do
    {
        printf("%d\n", i);
        i++;
    }while (i<=5);
}
```

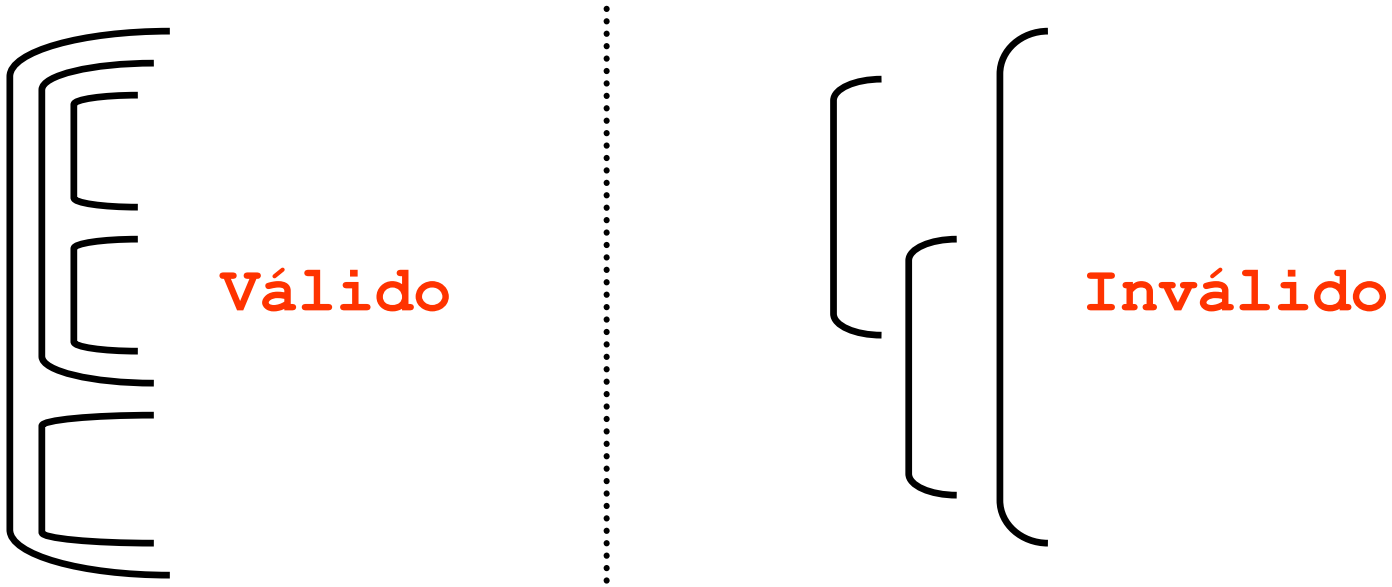
```
Usando o comando "for"
1
2
3
4
5

Usando o comando "while"
1
2
3
4
5

Usando o comando "do while"
1
2
3
4
5
```

Estruturas de repetição (cont.)

- Para as estruturas de repetição (for, while, do ... while) é:
  - Indispensável:
    - **Identificar as instruções que repetem**. Elas comporão o corpo da estrutura de repetição.
    - **Identificar a condição** de finalização, de controle, para saber quando as repetições serão finalizadas;
    - Definir a **forma de alcançar essa condição de controle**. Pode ser pelo **incremento/decremento** de uma variável, por resultado de processamento, por informação (entrada) do usuário.
  - Em loops contados, geralmente é necessário:
    - **Inicialização** da variável de controle para que as repetições possam ou não ser iniciadas;
    - **Incremento** ou **decremento** da variável de controle para alcançar a condição de finalização. Ou verificação dessa variável de controle.

Aninhamentos

- Aninhamentos ocorrem quando uma estrutura de controle (*if*, *switch*, *for*, *while* e *do while*) está inserida em uma outra estrutura de controle.
- Por exemplo: Uma **estrutura *if*** inserida em uma **estrutura for** e esta inserida em uma **estrutura while**.
- **Regra:** A estrutura mais interna deve estar inteiramente contida na estrutura imediatamente mais externa.



Variáveis contadoras e acumuladoras

- Em situações onde é necessário realizar contagens de ocorrências, ou somatórios e produtórios de valores dentro de um conjunto de dados, devemos utilizar variáveis específicas para fazer o armazenamento dos resultados.
- Chamamos de **contadoras** as variáveis que realizam a contagem de ocorrências de um determinado valor (ou situação) e de **acumuladoras** as variáveis responsáveis por armazenar os resultados de somatórios e produtórios de valores.

Variáveis contadoras

- As variáveis **contadoras** são normalmente inicializadas com valor 0 e incrementadas em 1 a cada iteração. Exemplo:

```c
#include <stdio.h>

int main(void)
{
    int i, cont=0;

    for(i=0; i <= 10; i++)
    {
        if(i%2 == 0)
        {
            printf("%d\n", i);
            cont++;
        }
    }
    printf("\nQuantidade de numeros pares: %d\n", cont);

    return 0;
}
```

Variáveis acumuladoras

- A variáveis **acumuladores** são utilizadas em dois tipos de situações:
  - Para a realização de somatórios;
  - Para a realização de produtórios.
- No caso dos **somatórios**, a variável acumuladora é normalmente inicializada com o valor 0 e incrementada no valor de um outro termo qualquer, dependendo do problema em questão.
- No caso dos **produtórios**, a variável acumuladora é normalmente inicializada com o valor 1 e incrementado no valor de um outro termo qualquer, dependendo do problema em questão.

Variáveis acumuladoras (cont.)

- **Exemplo de somatório:**

```c
#include <stdio.h>

int main(void)
{
    int i, soma=0;

    for(i=0; i <= 10; i++)
    {
        if(i%2 == 0)
        {
            printf("%d\n", i);
            soma = soma + i;
        }
    }
    printf("\nA soma dos numeros pares eh: %d\n", soma);

    return 0;
}
```

Variáveis acumuladoras (cont.)

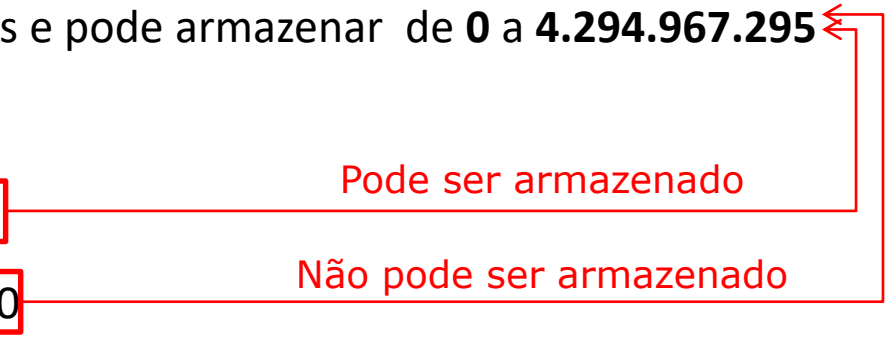
- **Exemplo de produtório:**

```c
#include <stdio.h>

int main(void)
{
    int i, produto=1;

    for(i=1; i <= 10; i++)
    {
        if(i%2 == 0)
        {
            printf("%d\n", i);
            produto = produto * i;
        }
    }
    printf("\nO produto dos numeros pares eh: %d\n", produto);

    return 0;
}
```

Variáveis acumuladoras (cont.)

Fatorial

- O cálculo do fatorial também é um exemplo de problema que deve ser resolvido usando um acumulador de produtório.
- Ao usar produtórios, deve-se levar em conta o tamanho máximo de um número que pode ser armazenado na memória:
  - int: ocupa 2 bytes e pode armazenar de **-32.768** a **32.767**
  - long int: ocupa 4 bytes e pode armazenar de **-2.147.483.648** a **2.147.483.648**
  - unsigned long int: ocupa 4 bytes e pode armazenar de **0** a **4.294.967.295**
- **Exemplo de fatorial:**
  - O fatorial de 12 é 479.001.600
  - O fatorial de 13 é 6.227.020.800



Referências

- DEITEL, P. J. DEITEL, H. M. **Como programar em C**. São Paulo: LTC, 1990.
- SCHILDT, H. **C Completo e total**, 3ª ed. São Paulo: Makron Books, 1996.
- MIZRAHI, V. V. **Treinamento em linguagem C: curso completo - módulo 1**. São Paulo: McGraw-Hill, 1990.
- Material baseado no conteúdo disponibilizado pela professora Beatriz Borsoi.

Dúvidas

- ???