FICR
Faculdade Católica Imaculada Conceição

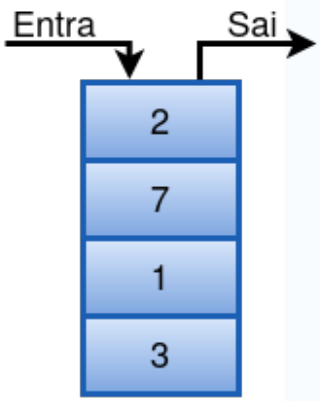
Estruturas de Dados

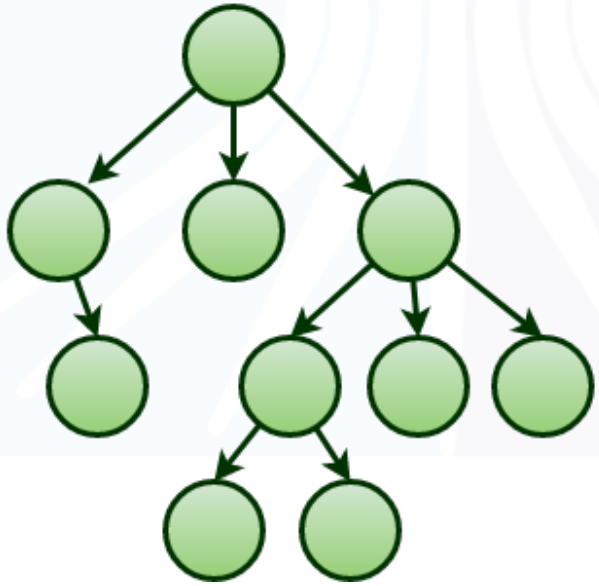
Mayrton Dias
mayrton.queiroz@p.ficr.edu.br

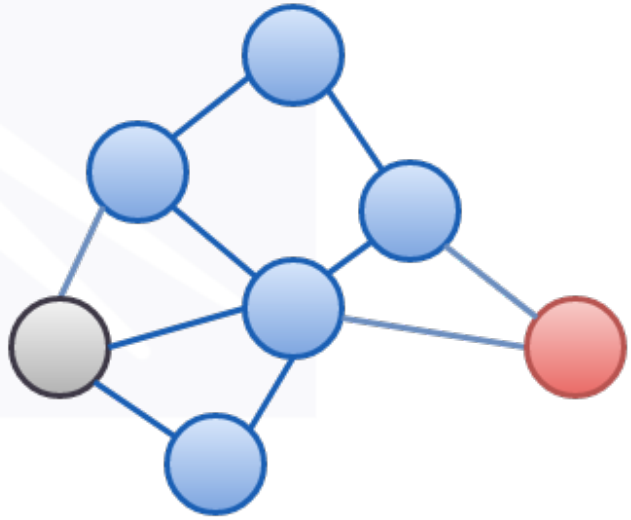
Apresentação

Apresentação

Estrutura de Dados: É o ramo da Computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento.







Ementa da disciplina

FICR
Faculdade Católica Imaculada Conceição

Disciplina: Estrutura de Dados.

Ementa: Tipos de dados, estrutura de dados e tipos abstratos de dados. Lista simplesmente encadeada com vetores e apontadores. Lista duplamente encadeada com vetores e apontadores. Lista circular com vetores e apontadores. Fila e Pilha com vetores e apontadores. Árvores utilizando alocação dinâmica, recursividade de árvores binárias de busca, buscas em profundidade (pré-fixado, pós fixado e em ordem) e em profundidade em largura, métodos de pesquisa: exaustiva, sequencial e binária, métodos de classificação interna: bolha, inserção, seleção e quicksort.

Ementa da disciplina

FICR
Faculdade Católica Imaculada Conceição

Disciplina: Estrutura de Dados.

Bibliografia:

- CELES, Waldemar, Renato Cerqueira, and José Rangel. **Introdução a estruturas de dados: com técnicas de programação em C.** GEN LTC, 2021.
- TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. **Estruturas de dados usando C.** Pearson Makron Books, 2004.
- CORMEN, Thomas H. et al. **Algoritmos: teoria e prática.** Editora Campus, v. 2, p. 296, 2002.

Ementa da disciplina

Av1: Avaliação 1

Av2: Avaliação 2

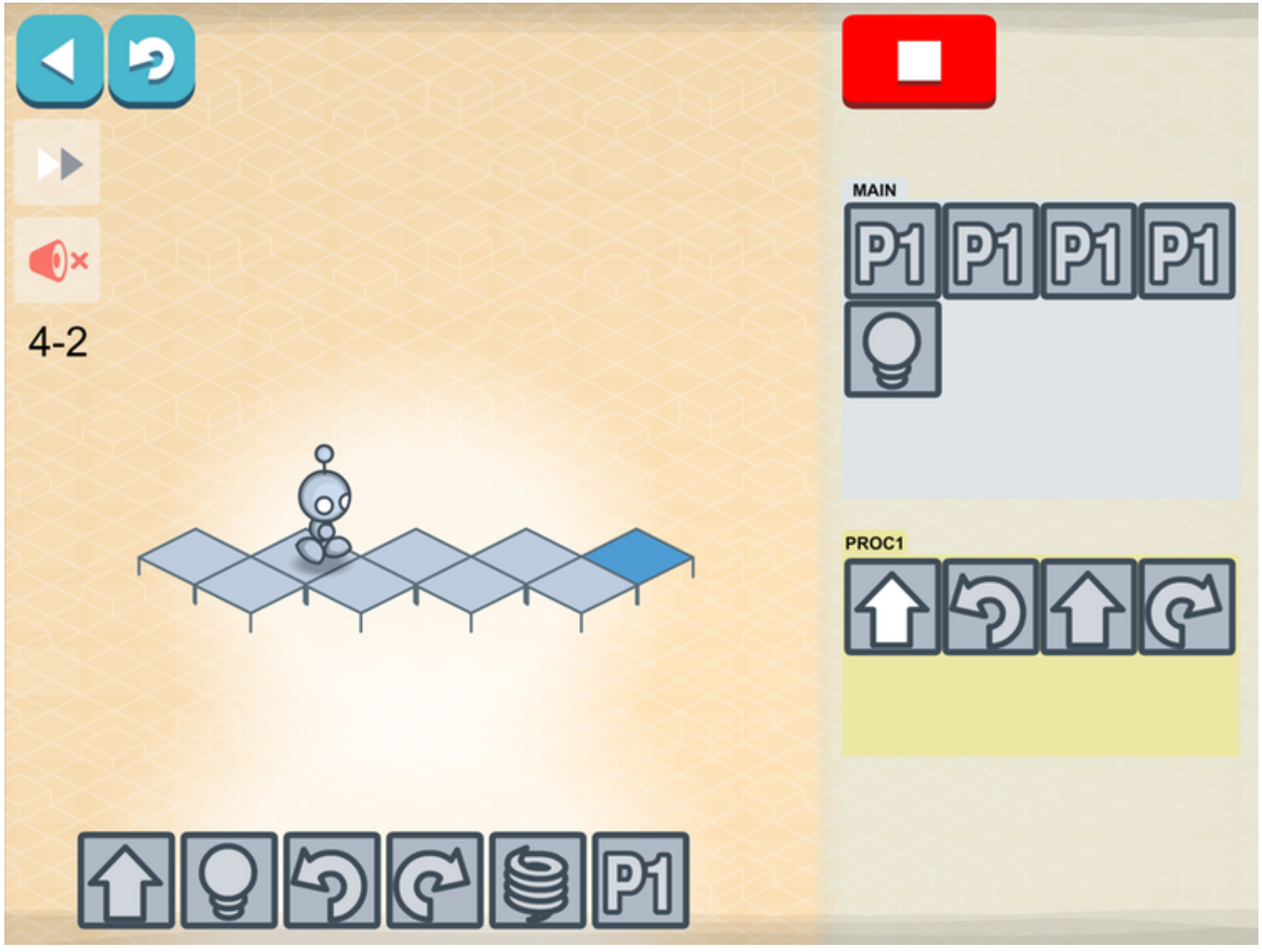
Av3: Avaliação 3 (peso 2) + AVA (peso 1) + ATV (peso 2)

Média:

$$\frac{(Av1 * 2) + (Av2 * 3) + (Av3 * 5)}{10}$$

Introdução

Introdução



https://www.gameflare.com/online-game/light-bot/

Introdução

Um **algoritmo** é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como **entrada** e produz algum valor ou conjunto de valores como **saída** (CORMEM, 2001).

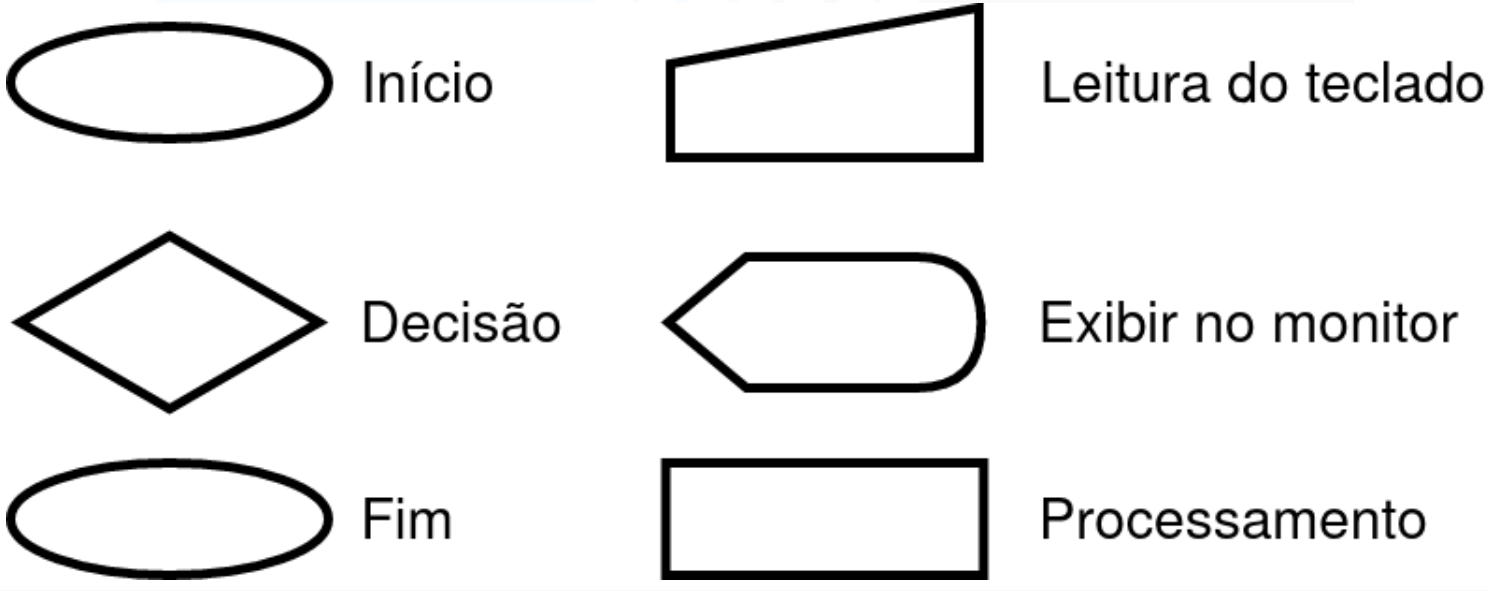
- Representações:
  - Narrativa
  - Fluxograma
  - Pseudocódigo

Descrição Narrativa

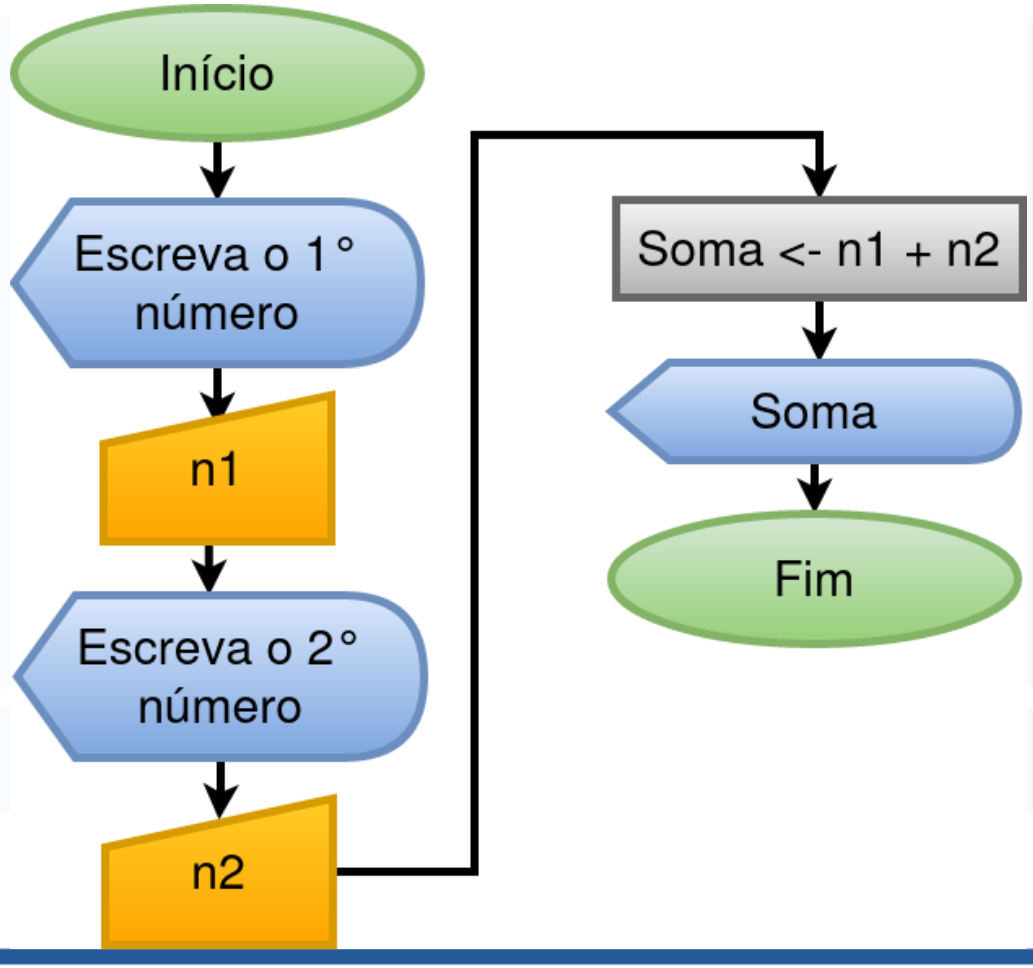
- **Vantagem:** Utiliza a linguagem natural, ou seja, o português.
- **Desvantagem:** Ambiguidade, ou seja, permite diversas interpretações.
- **Exemplo:**
  - Passo 1 - Obter o primeiro número.
  - Passo 2 - Obter o segundo número.
  - Passo 3 - Somar os dois números.
  - Passo 4 - Mostrar o resultado obtido na soma.

Fluxograma

- **Vantagem:** Os elementos gráficos permitem uma melhor compreensão, quando comparados aos textos.
- **Desvantagem:** É preciso aprender os elementos do fluxograma.
- **Elementos gráficos:**



Fluxograma


Início
Escreva o 1° número
n1
Escreva o 2° número
n2
Soma <- n1 + n2
Soma
Fim

Pseudocódigo

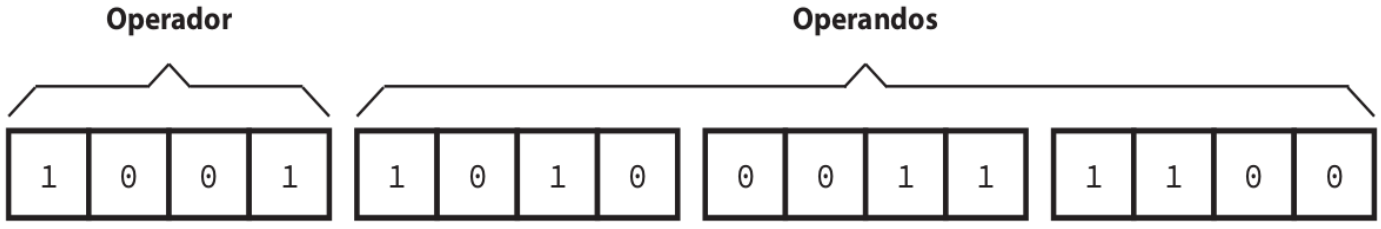
- **Vantagem:** Facilidade para implementar em uma Linguagem de Programação.
- **Desvantagem:** É preciso entender do Pseudocódigo.
- **Exemplo:**

```
Algoritmo Soma
        Inteiro n1, n2, soma
    Inicio
        Escreva("Digite o primeiro número")
        Leia(n1)
        Escreva("Digite o segundo número")
        Leia(n2)
        Soma <- n1 + n2
        Escreva("Soma = ", Soma)
    FimAlgoritmo
```

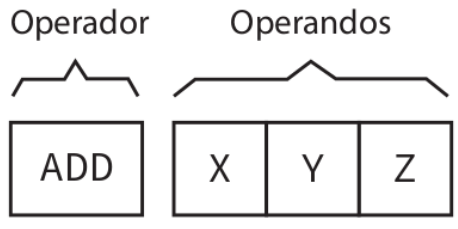
FICR
Faculdade Católica Imaculada Conceição

Linguagens de Programação

- Exemplo de uma instrução em Linguagem de Máquina



- Exemplo de uma instrução em Assembly



- Uso de palavras mnemônicas
- Uso de variáveis simbólicas

Linguagens de Programação

- **Tradutor** é um programa de computador que aceita como entrada um programa escrito em assembly ou numa linguagem de alto nível e produz como saída um programa escrito em linguagem de máquina.
- Há três tipos de tradutores que são:
  - **Assembler** (ou montador), que traduz programas escritos em assembly em programas escritos em linguagem de máquina.
  - **Compilador** é um programa que traduz um programa escrito numa linguagem de alto nível (programa-fonte) num programa escrito numa linguagem de máquina (programa-objeto)
  - **Interpretador**: cada instrução do programa-fonte é traduzida exatamente antes de ser executada e de acordo com o fluxo de execução do programa.

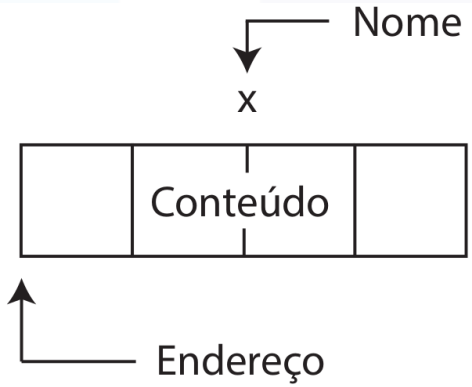
Editores de Texto

https://ubecedu-my.sharepoint.com/:u:/g/personal/mayrton_queiroz_p_ficr_edu_br/EcBCnkdTGldOrnFILS90dbcBC4u60TnNWIgQ_pknDHqksQ?e=i0OqoG

Conceitos Iniciais

Conceitos Iniciais

- Uma **variável** (simbólica) em programação representa por meio de um nome o conteúdo de um espaço contínuo em memória



- Um **identificador** serve para nomear um componente utilizado num programa escrito numa dada linguagem de programação
  - Um identificador em C deve ser constituído apenas de letras, dígitos e _ (subtraço)
  - O primeiro caractere de um identificador não pode ser um dígito
  - O ISO C90: 32 caracteres. O C99 e o C11: 63 caracteres

Conceitos Iniciais

- **Palavras-chaves:** `break`, `for`, `while`, ...
- **Palavras Reservadas:** `printf`, `math`, `scanf`, ...
- Tipos básicos (primitivos)

Tipo	Tamanho	Representatividade
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
short int	2 bytes	-32768 a 32767
unsigned short int	2 bytes	0 a 65535
long int	4 bytes	-2147483648 a 2147483647
unsigned long int	4 bytes	0 a 4294967295

Tabela: Tipos inteiros e suas representatividades.

Tipo	Tamanho	Representatividade
float	4 bytes	10^{-38} a 10^{38}
double	8 bytes	10^{-308} a 10^{308}

Tabela: Tipos para representação de números reais.

Propriedades dos Operadores:

- **Resultado:** é o valor resultante da aplicação do operador sobre seus operandos.
- **Aridade:** é o número de operandos sobre os quais o operador atua, ou seja, unário, binário e ternário.
- **Precedência:** determina sua ordem de aplicação com relação a outros operadores.
- **Associatividade:** é utilizada com operadores de mesma precedência, ou seja, associatividade à esquerda e à direita.

Operadores e Expressões

- **Operadores Aritméticos**

Simbolo	Operador
-	Menos unário (inversão de sinal)
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto de divisão inteira

Grupo de Operadores	Precedência
- (unário)	Mais alta
*, /, %	↓
+, - (binários)	Mais baixa

Operadores e Expressões

- **Operadores Relacionais**

Símbolo	Operador	Aplicação	Resultado
>	maior que	a >b	1 se a é maior do b ; 0 , caso contrário
>=	maior ou igual	a >= b	1 se a é maior do que ou igual a b ; 0 , c.c.
<	menor do que	a <b	1 se a é menor do b ; 0 , caso contrário
<=	menor ou igual	a <= b	1 se a é menor do que ou igual a b ; 0 , c. c.
==	igualdade	a == b	1 se a é igual a b ; 0 , caso contrário
!=	diferente	a != b	1 se a é diferente de b ; 0 , caso contrário

Grupo de Operadores	Precedência
>, >=, <, <=	Mais alta
==, !=	Mais baixa

Operadores e Expressões

- **Operadores lógicos**

Operador	Símbolo	Precedência
Negação	!	Mais alta
Conjunção	&&	↓
Disjunção	||	Mais baixa

X	!X
0	1
diferente de 0	0

X	Y	X && Y	X || Y
0	0	0	0
0	diferente de 0	0	1
diferente de 0	0	0	1
diferente de 0	diferente de 0	1	1

Conceitos Iniciais

- **Operadores e Expressões:**

Grupo de Operadores	Precedência
!, - (unário)	Mais alta
*, /, %	↓
+, - (binários)	↓
>, >=, <, <=	↓
==, !=	↓
&&	↓
||	Mais baixa

- **Operador de Atribuição:**

```
variável = expressão;
```

- **Incremento e Decremento:**

```
++i;, i++;, --i; e i--;
```

- **Comentários:**

```
/* Comentário em várias linhas */
// Comentário em uma linha
```

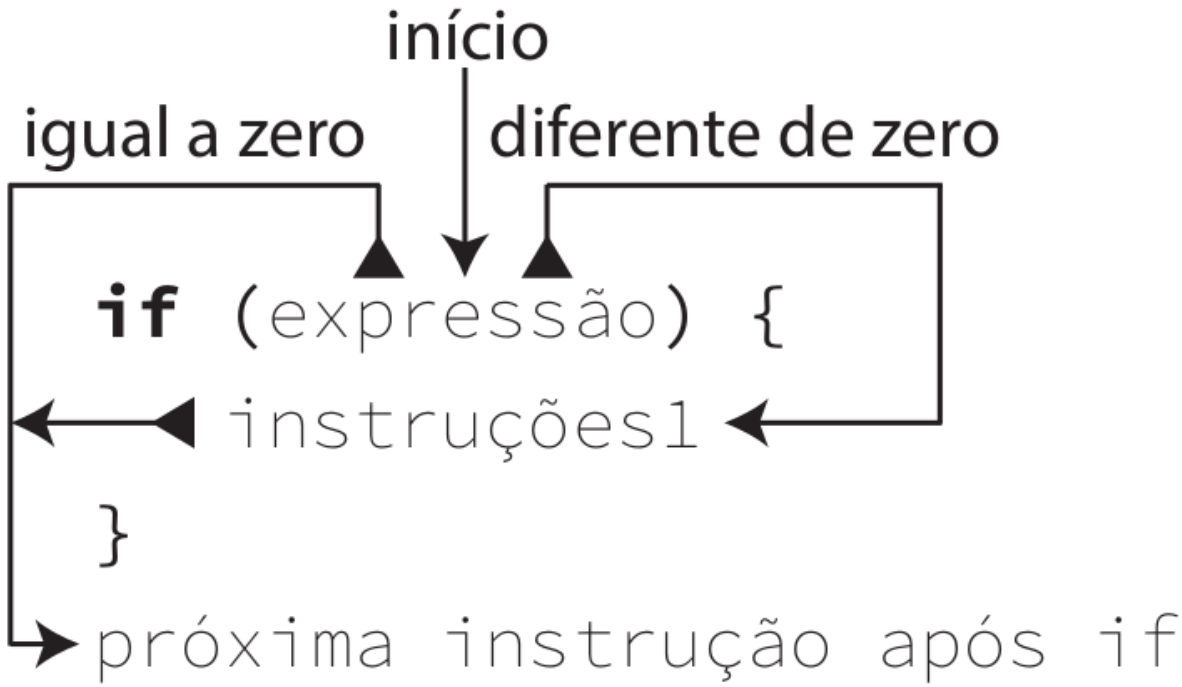
- **Biblioteca de Entrada e Saída de dados**
- **Função de Saída de dados via tela**:
  `printf(string-de-formatação, e1, e2, ..., en);`
- **Função de Entrada de dados pelo teclado**:
  `scanf(string-de-formatação, e1, e2, ..., en);`
- Adicionar a biblioteca: `#include <stdio.h>`

Especificador de formato	O item será apresentado como
%c	Caractere
%s	Cadeia de caracteres (string)
%d ou %i	Inteiro em base decimal
%f	Número real em notação convencional

Comandos condicionais

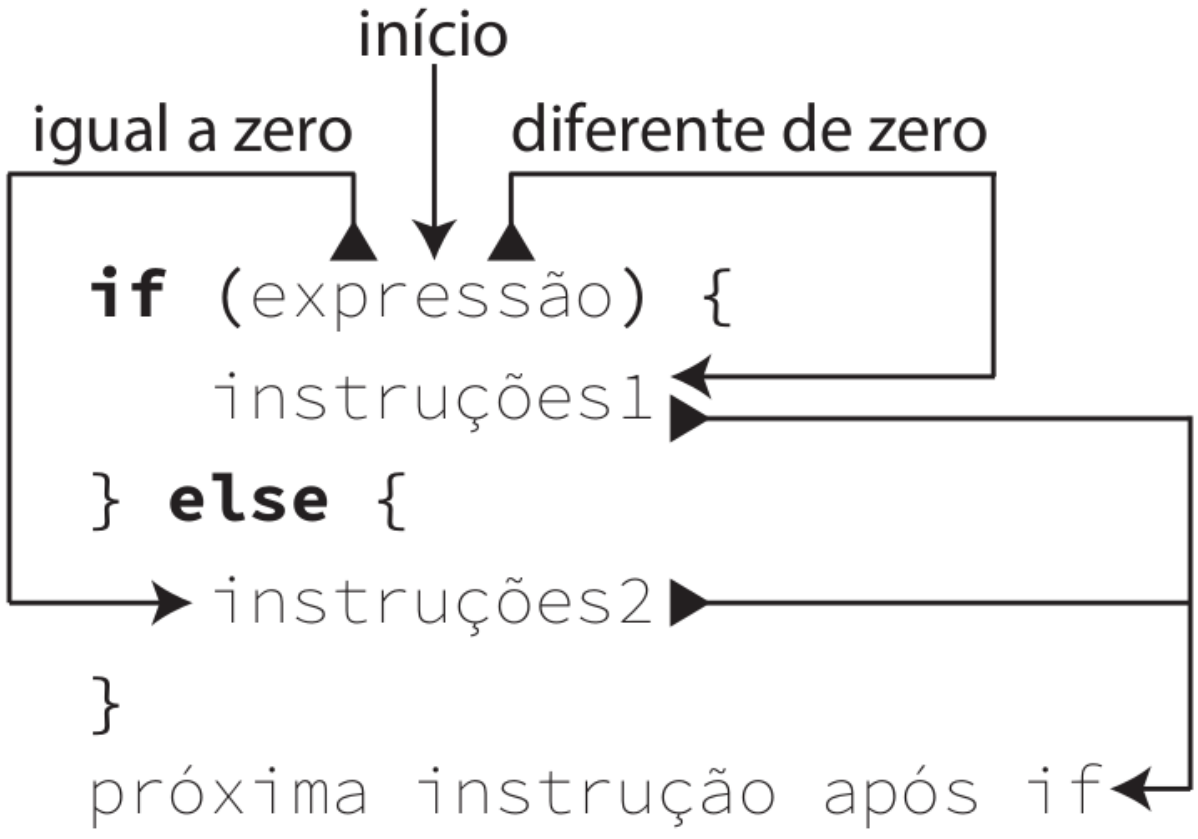
Comandos condicionais

- **Desvios condicionais** permitem desviar o fluxo de execução de um programa dependendo do resultado da avaliação de uma expressão (condição)



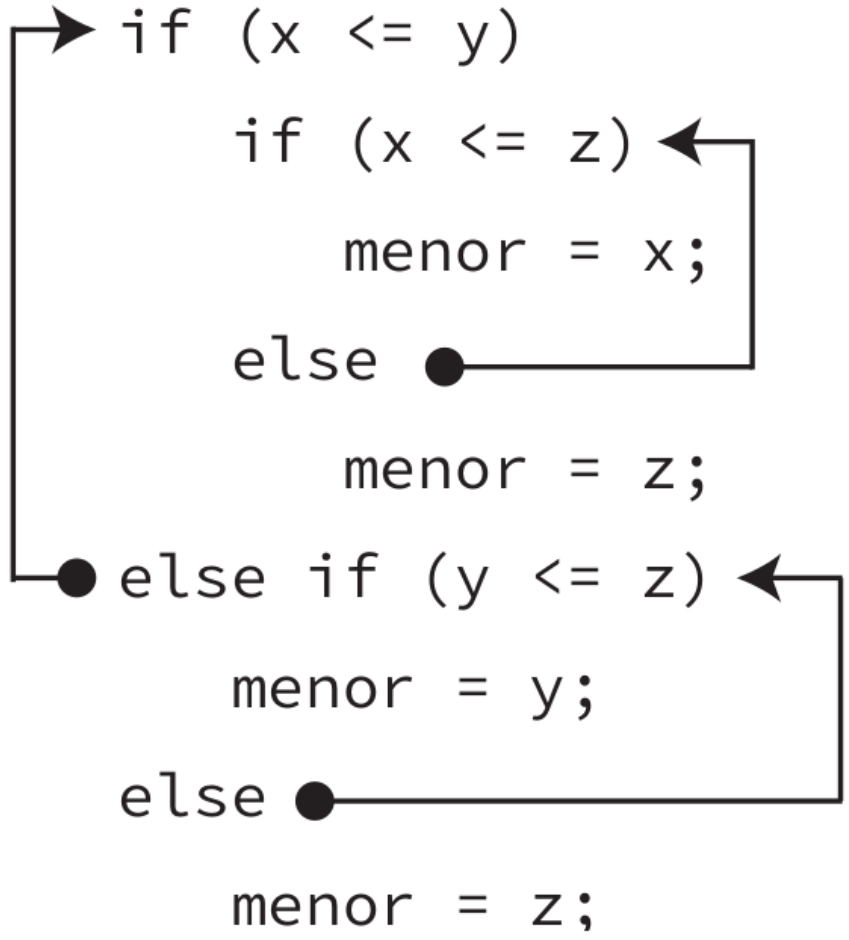
Comandos condicionais

- **if-else**



Comandos condicionais

- **if-else aninhados**



Comandos condicionais

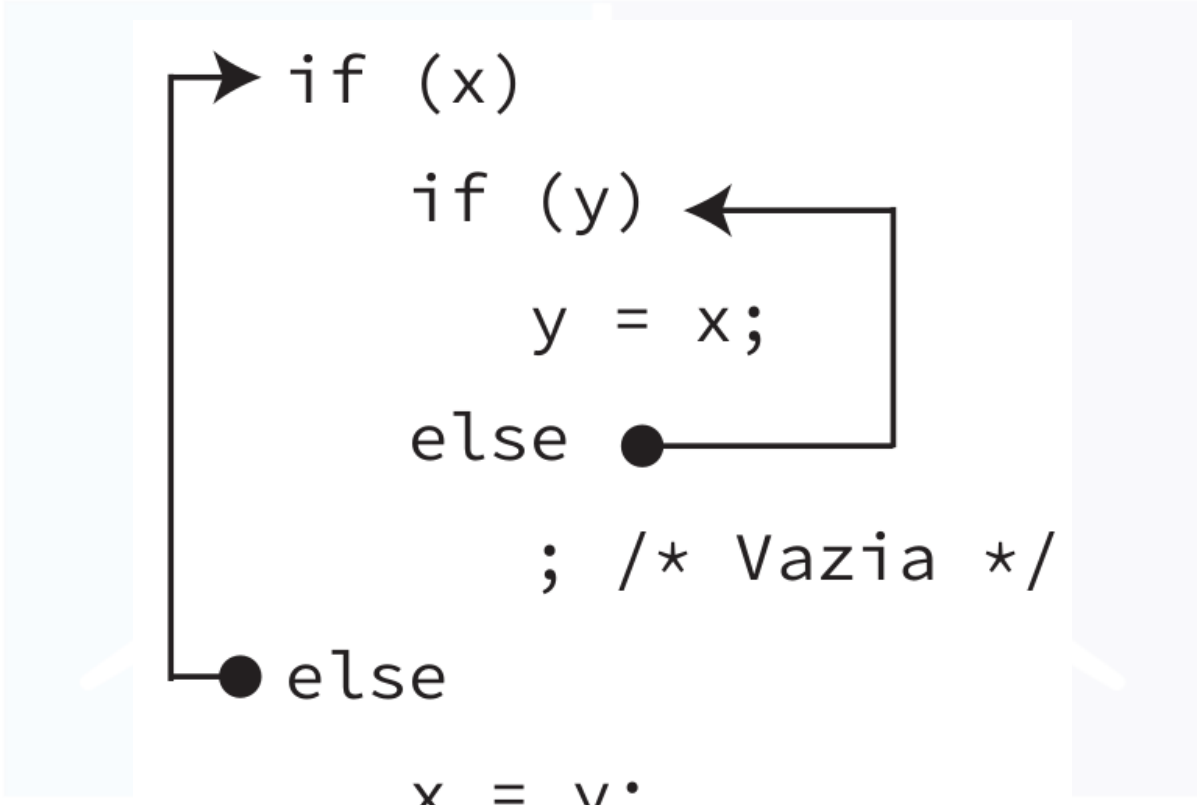
- **if-else aninhados**

```
if (x)
    if (y)
        y = x;
    else
        x = y;
```

```
if (x) {
    if (y)
        y = x;
} else
    x = y;
```

Comandos condicionais

- **if-else aninhados**



Comandos condicionais

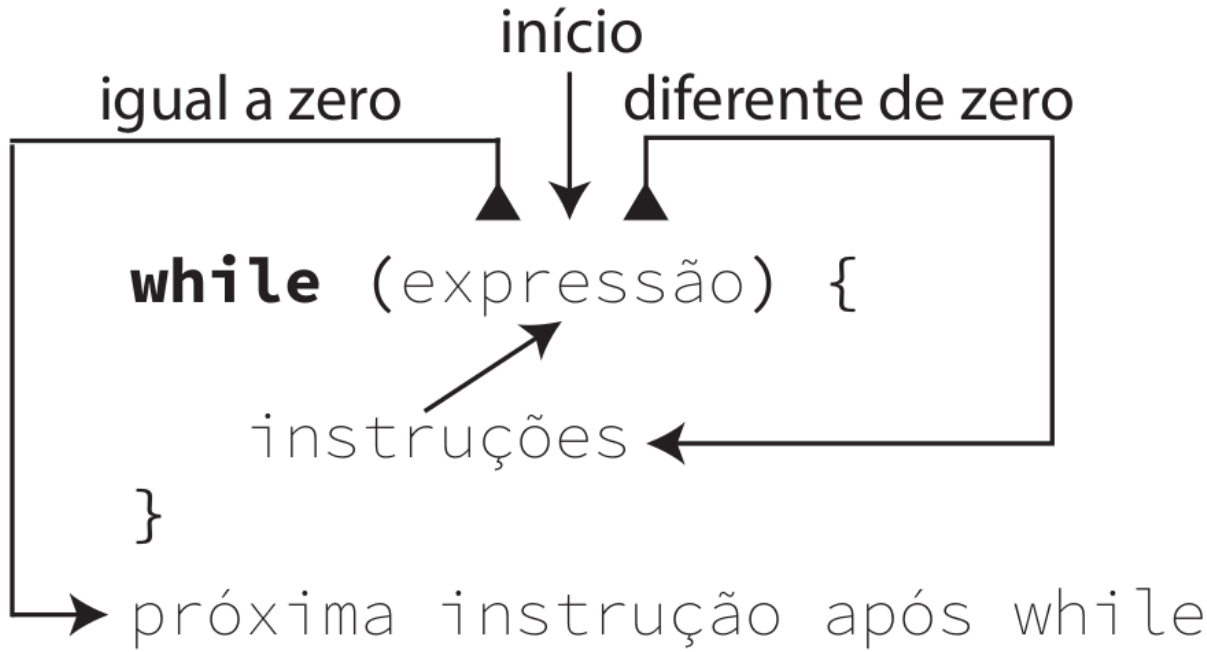
- **switch-case**

```
switch (expressão) {
        case expressão-constante1:
            instrução1;
        case expressão-constante2:
            instrução2;
        ...
        case expressão-constanten:
            instruçãon;
        default:
            instruçãod;
}
```

Comandos de repetição

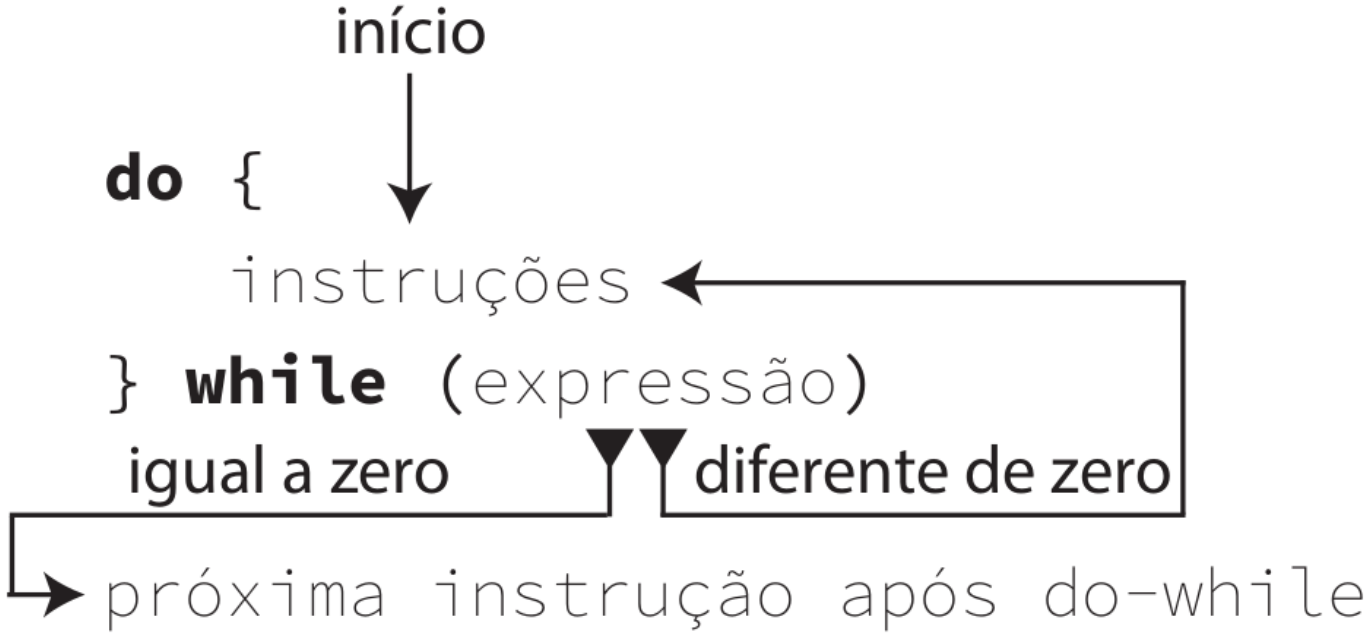
Comandos de repetição

- **while**



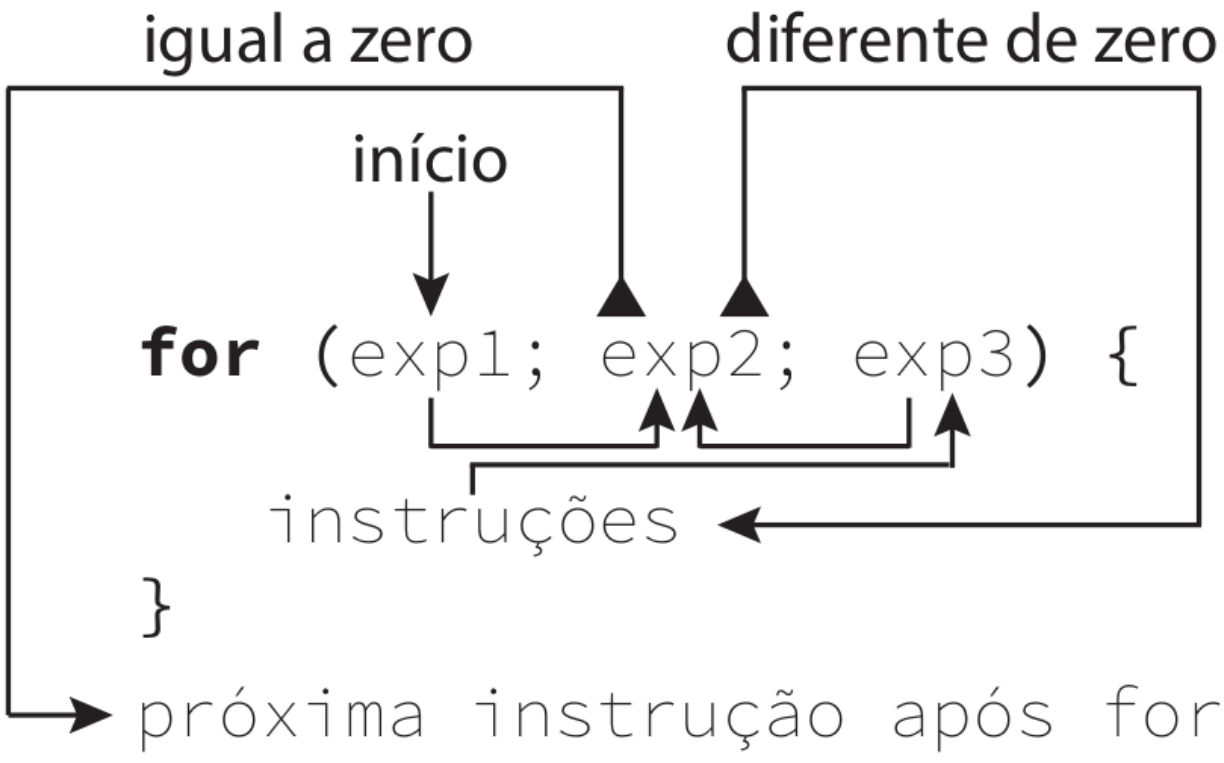
Comandos de repetição

- **do-while**



Comandos de repetição

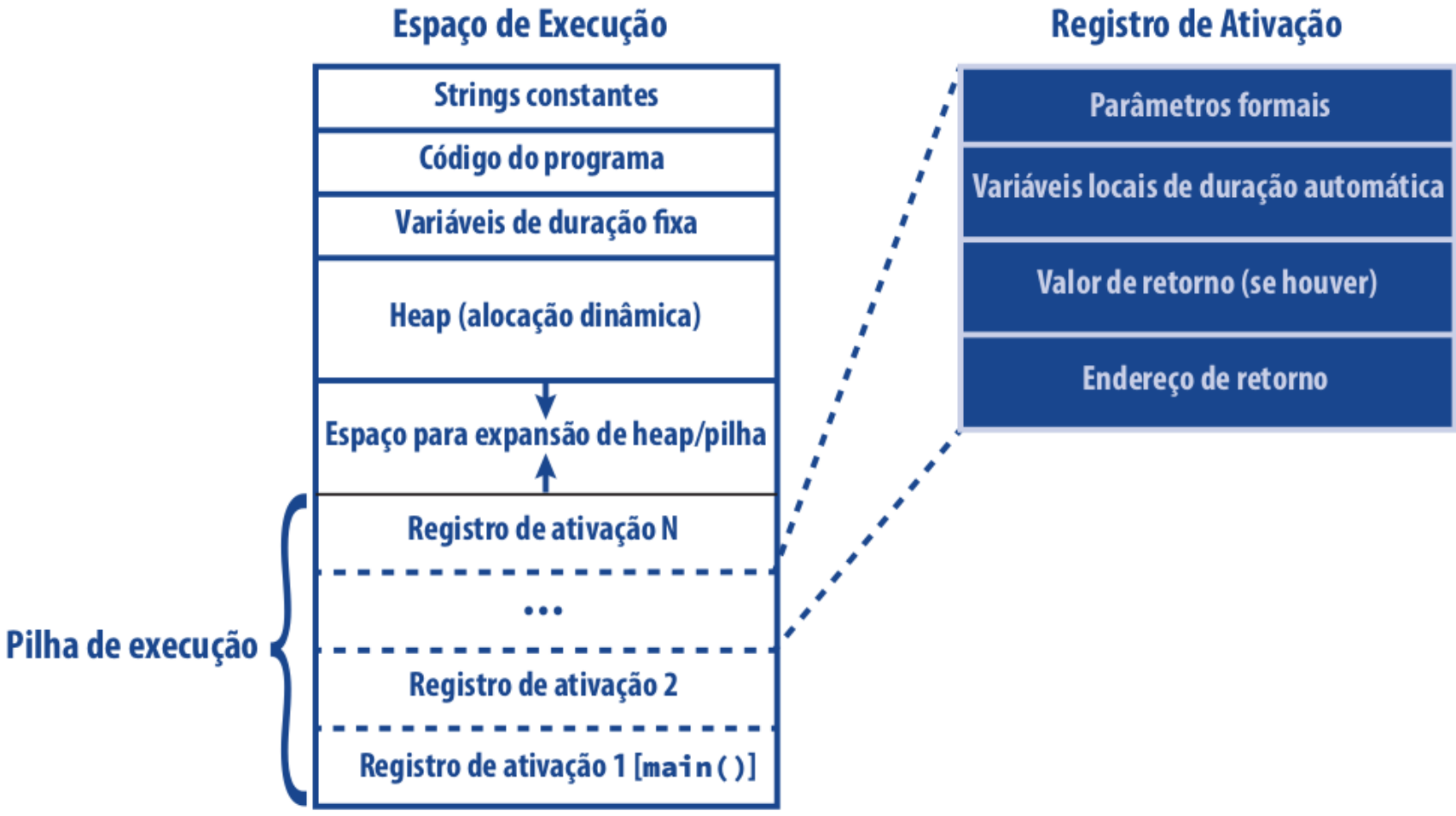
- **for**



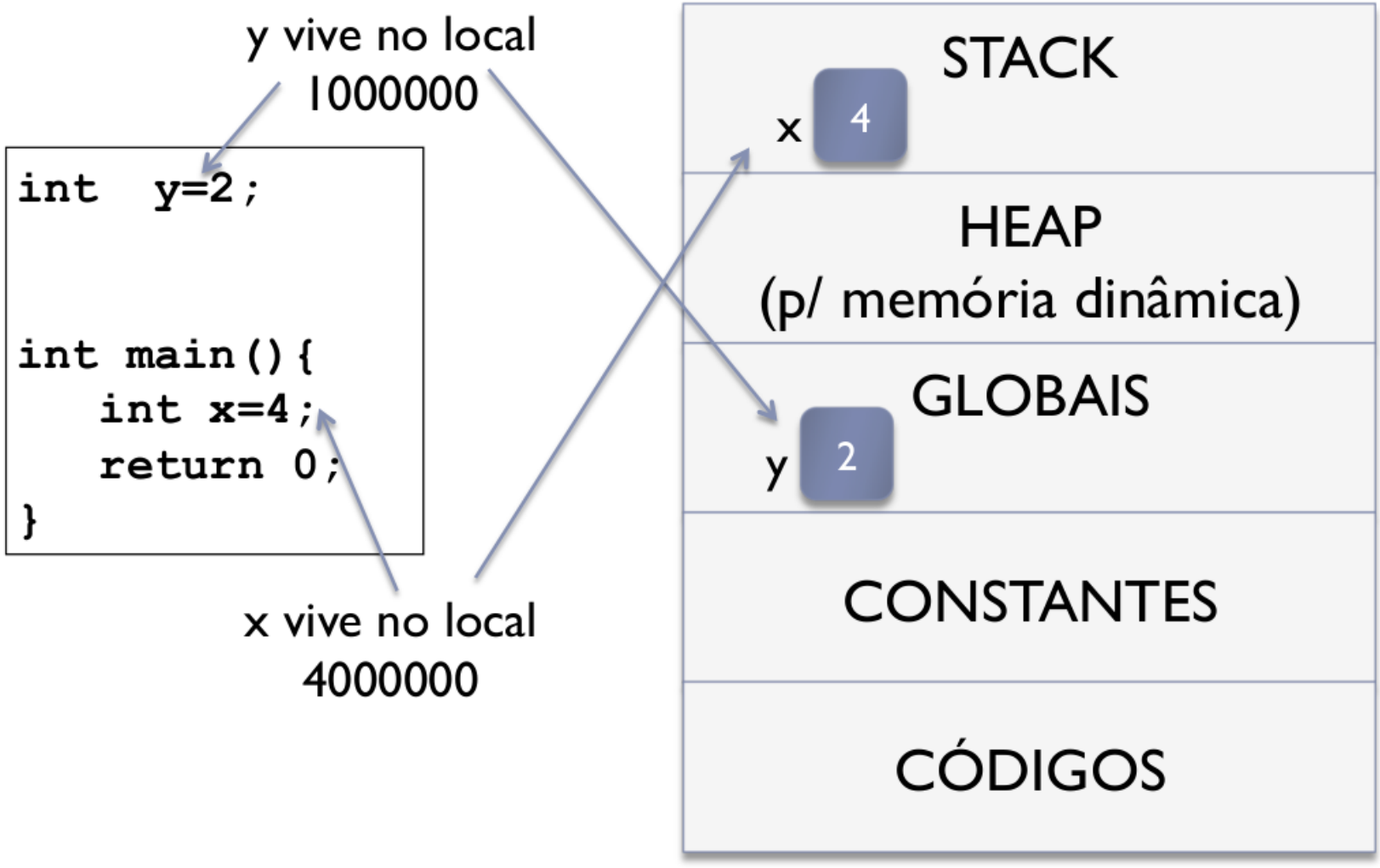
Ponteiro

Ponteiro

- Como a memória do computador está estruturada?



Ponteiro


y vive no local
1000000
int y=2;
int main(){
int x=4;
return 0;
}
x vive no local
4000000
STACK
x
4
HEAP
(p/ memória dinâmica)
GLOBAIS
y
2
CONSTANTES
CÓDIGOS

Ponteiro

Endereço de uma variável:

- Indica o local (posição) em memória onde ela é armazenada
- Determinado por meio do uso do operador de endereço &
- Exemplo:

```
int   x =4;
...
&x
```

Acessaria o endereço (localização) de x na memória.

Nesse exemplo, o valor 4000000.

- Não é permitido modificar o endereço de uma variável
- `&x = 5000; /* ILEGAL */`

Ponteiro

- **Tipo especial de variável** que contém **um endereço**
- Funciona como um **apontador para outras variáveis**
- Pode apontar para um valor de qualquer tipo armazenado em memória
- Definição:
- `<tipo apontado> *variavelPonteiro;`

Ponteiro

- Exemplo: Definição de Ponteiro

```
int meuInteiro;
int *ponteiroParaInteiro;

ponteiroParaInteiro = &meuInteiro;
```

Define `ponteiroParaInteiro` como um ponteiro capaz de apontar para uma variável do tipo int

`ponteiroParaInteiro` é um ponteiro para o tipo int iniciado com o endereço da variável `meuInteiro`

Ponteiro

- Pode-se atribuir um valor ao conteúdo da posição de memória apontada por um ponteiro usando o operador de indireção

```
float  meuFloat = 3.14;
float  *ponteiroParaFloat = &meuFloat;


*ponteiroParaFloat = 1.6;
```

Atribui 1.6 ao conteúdo da posição de memória apontada por `ponteiroParaFloat`

Atribuições equivalentes → `meuFloat = 1.6;`

Ponteiro

- O fato de C utilizar "*"para declaração e indireção de ponteiros pode causar alguma confusão em iniciantes.

```
float  meuFloat = 3.14;
float  *ponteiroParaFloat = &meuFloat;
...

*ponteiroParaFloat = 1.6;
```

O asterisco NÃO representa o operador de indireção

O operador de indireção é utilizado aqui

Ponteiro

Faculdade Católica Imaculada Conceição

Ponteiro nulo

- Não aponta para nenhum endereço válido
- Reconhecido pelo sistema como inválido
- Recebe o valor inteiro 0
  - Ou macro NULL (arquivo de cabeçalho stdlib.h)
- O programa é abortado quando se tenta acessar o conteúdo apontado por ele
- Exemplo:

```
char  *p;
...
p = 0;
```

Torna `p` um ponteiro nulo

Funções

Funções

- Função é um subprograma que consiste de um conjunto de instruções e declarações que executam **uma tarefa específica**
- Benefícios:
  - Facilidade de manutenção
  - Melhora de legibilidade
- Uma definição de função representa a implementação da função e é dividida em duas partes:
  - **Cabeçalho** que informa o nome da função, qual é o tipo do valor que ela produz e quais são os dados de entrada e saída (parâmetros) que ela manipula.
  - **Corpo da função** que processa os parâmetros de entrada para produzir os resultados desejados.

Funções

- Exemplo de função retornando um número inteiro:

```
int SomaAteN(int n)
{
   int i, soma = 0;

   for (i = 1; i <= n; ++i) {
      soma = soma + i;
   }

   return soma;
}
```

Funções

- Exemplo de função sem retorno, ou seja `void`:

```
#include <stdio.h> /* printf() */

void ApresentaMenu(void)
{
   printf( "\nAs opcoes deste programa sao:\n\n"
           "\tOpcao 1\n"
           "\tOpcao 2\n"
           "\tOpcao 3\n"
           "\tOpcao 4\n"
         );
}

int main(void)
{
   ApresentaMenu();

   return 0;
}
```

Funções

- **Passagem por valor:** quando um parâmetro é passado por valor, o parâmetro formal recebe uma cópia do parâmetro real correspondente. Qualquer alteração sofrida pelo parâmetro formal no corpo da função fica restrita a essa cópia e, portanto, não é transferida para o parâmetro real correspondente. Essa é a única modalidade de passagem de parâmetros existente em C.
- **Passagem por referência:** o parâmetro formal e o parâmetro real correspondente, que deve ser uma variável, compartilham o mesmo espaço em memória, de maneira que qualquer alteração feita pela função no parâmetro formal reflete-se na mesma alteração no parâmetro real correspondente. A linguagem C não possui passagem de parâmetros por referência.

Funções

- Função `Trocar1` para trocar os valores de duas variáveis:

```
void Troca1(int inteiro1, int inteiro2)
{
   int aux = inteiro1; /* Guarda o valor do primeiro inteiro */

      /* A variável cujo valor foi guardado recebe o valor da  */
      /* outra variável e esta recebe o valor que foi guardado */
   inteiro1 = inteiro2;
   inteiro2 = aux;
}
```

```
int main(void)
{
   int i = 2, j = 5;

   printf( "\n\t>>> ANTES da troca <<<\n"
           "\n\t    i = %d, j = %d\n", i, j );

   Troca1(i, j);

   printf( "\n\t>>> DEPOIS da troca <<<\n"
           "\n\t    i = %d, j = %d\n", i, j );

   return 0;
}
```

Funções

- Função `Trocar2` para trocar os valores de duas variáveis:

```
void Troca2(int *ptrInt1, int *ptrInt2)
{
   int aux = *ptrInt1; /* Guarda o valor do primeiro */
                       /* inteiro que será alterado  */

   /*                                                  */
   /* A variável cujo valor foi guardado recebe o valor da  */
   /* outra variável e esta recebe o valor que foi guardado */
   /*                                                  */

   *ptrInt1 = *ptrInt2;
   *ptrInt2 = aux;
}
```

```
int main(void)
{
   int i = 2, j = 5;

   printf( "\n\t>>> ANTES da troca <<<\n"
           "\n\t    i = %d, j = %d\n", i, j );

   Troca2(&i, &j);

   printf( "\n\t>>> DEPOIS da troca <<<\n"
           "\n\t    i = %d, j = %d\n", i, j );

   return 0;
}
```

- Saída da Função `Trocar1`:

```
>>> ANTES da troca <<<
    i = 2, j = 5
>>> DEPOIS da troca <<<
    i = 2, j = 5
```

- Saída da Função `Trocar2`:

```
>>> ANTES da troca <<<
    i = 2, j = 5
>>> DEPOIS da troca <<<
    i = 5, j = 2
```

Arrays (Vetores)

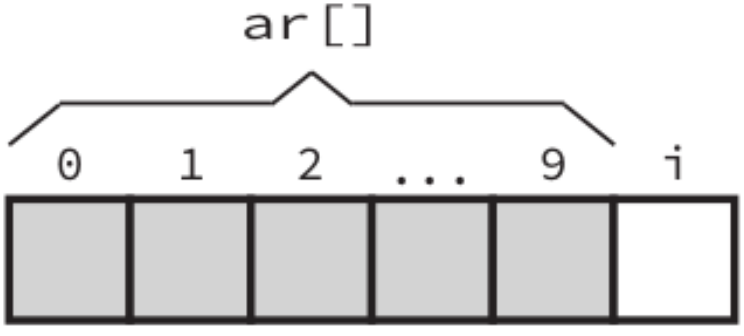
Arrays

- **Array** é uma variável estruturada e homogênea que consiste numa coleção de variáveis do mesmo tipo armazenadas contiguamente em memória.
- Cada variável que compõe um array é denominada **elemento** e pode ser acessada usando o nome do array e um **índice**
- Definição de um array:

```
tipo nome[tamanho];
```

- Acesso a Elementos de um Array:

```
nome[indice];
```



Arrays

- Array multidimensional é um array cujos elementos também são arrays
- Definição de um array multidimensional:

  `tipo nome[tamanho1][tamanho2]...[tamanhoN];`

- Frequentemente, a primeira e a segunda dimensões de um array bidimensional são denominadas, respectivamente, **linha** e **coluna**.
- `arBi[]` é definido como um array com 5 linhas e 3 colunas, mas apenas suas três primeiras linhas são iniciadas explicitamente e apenas o primeiro elemento da sua segunda linha é iniciado explicitamente.

```
int  arBi[5][3] = { {1, 2, 3},
                    {4},
                    {5, 6, 7}
                  };
```

String

String

- **String** é um array de elementos do tipo char terminado pelo caractere nulo, representado pela sequência de escape '\0'
- Definição de uma string:

```
char str1[] = "bola";
char str1[] = {'b', 'o', 'l', 'a', '\0'};
```

- Especificador de formato %*s*
- Limitar a quantidade de caracteres "%10*s*"
- Para receber string com espaço %[^\n]*s*
- Limitar a quantidade de caracteres e com espaços %10[^\n]*s*

String


```c
#include <stdio.h>   /* printf() */
#include <stdlib.h>  /* atoi()   */

int main(void)
{
   char *str = "1234";

   printf( "\n>>> String \"%s\" convertido em int: %d\n",
           str, atoi(str) );

   return 0;
}
```

Função recursiva

Função recursiva

- Função recursiva é uma função que chama a si mesma, ou diretamente ou por meio de outra função.

```
1 func1(...){
2     ...
3     func1(...);
4     ...
5 }
```

Exemplo:

O fatorial de um inteiro não negativo n:

$$n! = n * (n-1) * (n-2) * ... * 1$$

Função recursiva

$0! = 1$

$1! = 1$

$2! = 2 * 1$

$3! = 3 * 2 * 1$

$4! = 4 * 3 * 2 * 1$

...

Assim:

$$5! = 5 * \mathbf{4 * 3 * 2 * 1} = 5 * 4! = 120$$

$$(5-1)!$$

Função recursiva

Uma definição recursiva da função fatorial n é:

$$n! = n * (n-1)! = n * (n-1) * (n-2) * ... * 1$$

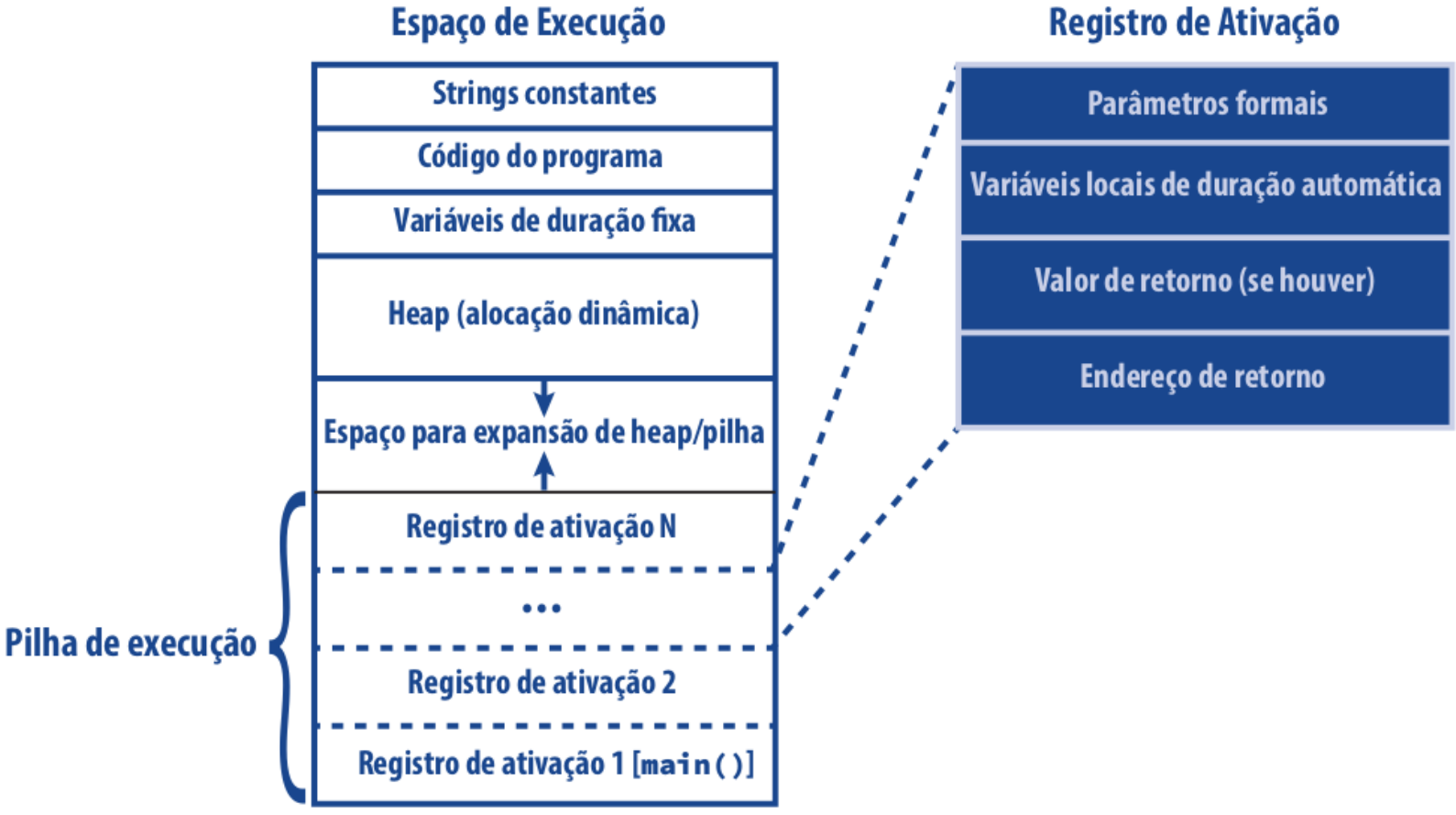
$n! = 1$ se $n = 0$

$n! = n * (n-1)!$ se $n > 0$

Solução iterativa de um inteiro n

```
6 fatorial = 1;
7 for (int contador = n; contador >= 1; contador--)
8   fatorial *= contador;
```

Função recursiva



Função recursiva

Solução recursiva de um inteiro *n*

```
10 if (n == 0)
11   fac = 1;
12 else
13   fac = n * fatorial (n-1);
```

Função recursiva

Fundamentos:

- **Recursão direta:** o código da função F contém uma sentença que invoca F
- **Recursão indireta:** a função F invoca uma função G que invoca, por sua vez, a função H, e assim sucessivamente, até que se invoca novamente a função F
- **Componente-base**: $f(n)$ se defina diretamente (ou seja, não de maneira recursiva) para um ou mais valores de n.

$$f(n) = 1 \qquad \text{se } n = 0$$

$$f(n) = n * f(n-1) \qquad \text{se } n > 0$$

Problema 1

Escrever uma função recursiva que calcule o fatorial de um número n

$$n! = 1 \qquad \text{se } n = 0$$
$$n! = n * (n-1)! \qquad \text{se } n \geq 1$$

```
15 int Fatorial (int n) {
16   // calculo de n!
17   if (n <= 1)
18     return 1;
19   else
20     return n * Fatorial(n - 1);
21 }
```

Problema 2

Escrever uma função de Fibonacci de modo recursivo e um programa que manipule essa função, de modo que calcule o valor do elemento de acordo com a posição ocupada na série.

```
23 unsigned long fibonacci(unsigned long n) {
24   if (n == 0 || n == 1)
25     return n;
26   else
27     return fibonacci(n-1) + fibonacci(n-2);
28 }
```

Função recursiva


```
29 void A(int c);
30 void B(int c);
31
32 int main() {
33   A('Z');
34   printf("\n");
35   return 0;
36 }
37 void A(int c) {
38   if (c > 'A')
39     B(c);
40   putchar(c);
41 }
42 void B(int c) {
43   A(--c);
44 }
```

Função recursiva

Implementação recursiva vs implementação iterativa

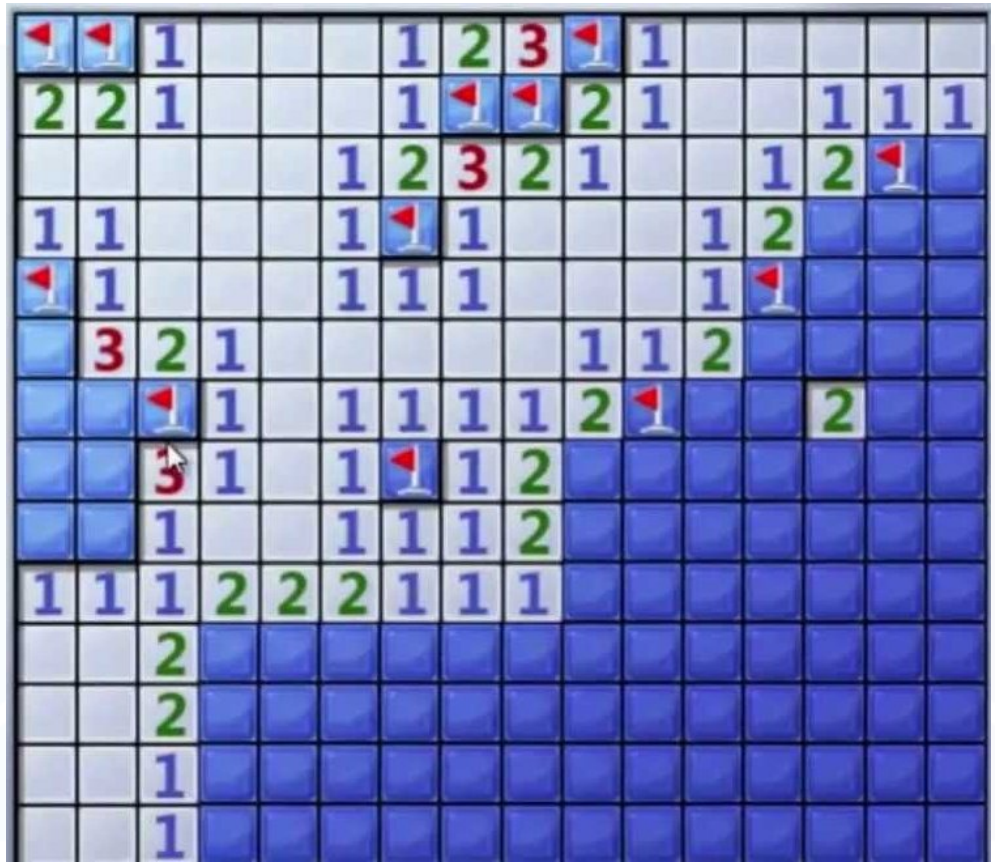
- A iteração utiliza uma estrutura repetitiva e a recursão utiliza uma estrutura de seleção
- A iteração e a recursão implicam repetição
  - A iteração utiliza explicitamente uma estrutura repetitiva
  - A recursão consegue a repetição por meio de chamadas repetidas de funções

Função recursiva

Implementação recursiva vs implementação iterativa

- A iteração e a recursão implicam cada uma um teste de término
  - A iteração termina quando a condição do laço não se cumpre
  - A recursão termina quando se reconhece um caso-base ou a condição de saída é alcançada
- A recursão precisa de tempo suplementar para realizar as chamadas de funções
- Mais fácil de implementar em problemas complexos que possuem natureza recursiva.

Função recursiva



Estruturas

Estruturas

- **Estruturas** são variáveis estruturadas semelhantes a arrays que diferem destes por permitirem que seus elementos, denominados **campos** ou **membros**, sejam de tipos diferentes.
- Por causa dessa característica, estruturas constituem variáveis heterogêneas.
- Definição de uma estrutura:

```
typedef struct rótulo-da-estrutura {
        tipo1  campo1;
        tipo2  campo2;
        ...
        tipoN campoN;
} nome-do-tipo;
```

Estruturas

- Exemplo:

```
typedef  struct  {
            char nome[30];
            int  dia, mes, ano;
         } tRegistro;


tRegistro  registroDaPessoa, *ptrParaRegistro;
```

- Inicialização:

```
tRegistro registroDaPessoa = {"Jose da Silva", 12, 10, 1960};
```

- Exemplo de acesso com a estrutura:

Campo	Acesso com `registroDaPessoa`	Acesso com `ptrParaRegistro`
`nome`	`registroDaPessoa.nome`	`ptrParaRegistro->nome`
`dia`	`registroDaPessoa.dia`	`ptrParaRegistro->dia`
`mes`	`registroDaPessoa.mes`	`ptrParaRegistro->mes`
`ano`	`registroDaPessoa.ano`	`ptrParaRegistro->ano`

Estruturas

- Estrutura aninhadas:

```
typedef struct {
            int  dia, mes, ano;
        } tData;

typedef struct {
            char   nome[30];
            tData  nascimento;
        } tRegistro2;
```

Estruturas

```
#include <stdio.h>

typedef struct  {
           int  dia, mes, ano;
        } tData;

typedef struct  {
           char   nome[30];
           tData  nascimento;
        } tRegistro2;

int main(void)
{
   tRegistro2 pessoa = { "Jose da Silva", {12, 10, 1960} };
   tRegistro2 *ptrPessoa = &pessoa;

   printf("\n\n***** Usando pessoa *****\n\n");

   printf("Nome: %s\n", pessoa.nome);
   printf( "Nascimento: %d/%d/%d\n", pessoa.nascimento.dia,
           pessoa.nascimento.mes, pessoa.nascimento.ano );

   printf("\n***** Usando ptrPessoa *****\n\n");

   printf("Nome: %s\n", ptrPessoa->nome);
   printf( "Nascimento: %d/%d/%d\n", ptrPessoa->nascimento.dia,
           ptrPessoa->nascimento.mes, ptrPessoa->nascimento.ano );

   return 0;
}
```

Nome	Símbolo	Utilizado em...
Operador de indexação	[]	Acesso a elemento de array
Operador de chamada de função	()	Chamada de função
Operador ponto	.	Acesso a campo de estrutura
Operador seta	->	Acesso a campo de estrutura
Operador de indireção	*	Acesso indireto a variável

FICR
Faculdade Católica Imaculada Conceição

Estruturas de Dados

Mayrton Dias
mayrton.queiroz@p.ficr.edu.br