



Aula 11: Revisão para Prova 01

Introdução a Programação

Túlio Toffolo & Puca Huachi
<http://www.toffolo.com.br>

Avaliação

- 3 Provas (60% da nota):
 - **Prova 01: 15% da nota (23/04/2019).**
 - Prova 02: 20% da nota (28/05/2019).
 - Prova 03: 25% da nota (02/07/2019).
- Exercícios em aula práticas (10% da nota):
 - Atividades em todas as aulas serão entregues via moodle.
- Trabalho(s) prático(s) (30% da nota):
 - Entrega 01: 10% da nota.
 - Entrega 02: 20% da nota.
 - Código e documentação serão entregues via moodle.
 - Apresentação para o(s) professor(es) da disciplina no final do semestre.
- Ponto extra: frequência e exercícios nas aulas teóricas

Revisão

- A seguir alguns slides foram selecionados das aulas anteriores para uma breve revisão do conteúdo.
- A aula de hoje está organizada em 7 partes:
 1. Conceitos e Linguagens
 2. Variáveis, operadores, comandos de entrada/saída
 3. Fluxogramas
 4. Condicionais (if, if-else, switch)
 5. Funções
 6. Ponteiros e passagem por referência
 7. Macros e bibliotecas

Antes da Revisão...

- **Operador ternário em C:**

- Permite fazer uma comparação em uma única linha
- Particularmente úteis em macros e atribuições simples
- Sintaxe:

```
<condição> ? <op. se verdadeiro> : <op. se falso>;
```

1

Conceitos e Linguagens

Linguagens de Programação – LP

- Os programadores escrevem seus programas em várias LPs, algumas entendidas diretamente pelos computadores, outras requerendo passos intermediários de tradução.
- As LPs são divididas em três tipos gerais:
 - 1 Linguagens de Máquina
 - 2 Linguagens Assembly
 - 3 Linguagens de Alto-nível

Um ambiente típico de desenvolvimento C/C++

- Passos comuns utilizados na criação e execução de uma aplicação C/C++.

Um ambiente típico de desenvolvimento C/C++

- **Fase 1: Criando um programa**

- Esta fase consiste da edição de um arquivo com um **programa editor** (normalmente conhecido como um editor).
- Você digita um programa C/C++ (tipicamente conhecido como **programa fonte**) usando o editor, faz as correções necessárias e salva o programa em um dispositivo de memória secundária, por exemplo, o HD.
- Frequentemente, os nomes de arquivos dos programas fonte C terminam com a extenção .c e de C++ com as extensões .cpp, .cxx ou .cc.

Um ambiente típico de desenvolvimento C/C++

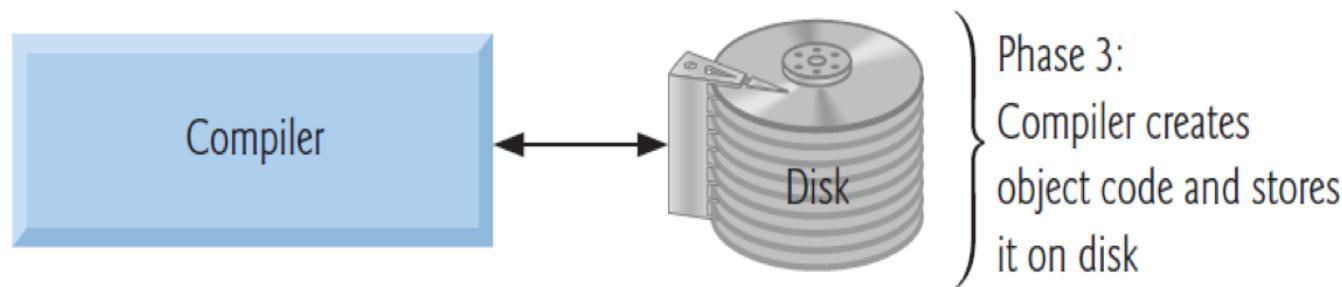
- **Fase 2: Pré-processando um Programa C/C++**

- Na fase 2, você fornece o comando para **compilar** o programa.
- Em um sistema C/C++, um programa **pré-processador** executado automaticamente antes que a fase de tradução do compilador inicie (então, chamaremos a fase 2 de pré-processamento e a fase 3 de compilação).
- O pré-processador obedece a comandos chamados **diretivas do pré-processador**, que indicam que certas manipulações são realizadas no programa antes da compilação.
- Estas manipulações usualmente incluem outros arquivos de texto para serem compilados, e realizam várias substituições de texto.

Um ambiente típico de desenvolvimento C/C++

● Fase 3: Compilando um Programa C/C++

- Na fase 3, o compilador **traduz** o programa C/C++ (código fonte, em **alto nível**) em um código de linguagem de máquina (código objeto, em **baixo nível**).



Um ambiente típico de desenvolvimento C/C++

- **Fase 4: Ligação (*linking*)**

- Tipicamente, um programa C/C++ contém referências para funções e dados definidos em outros lugares, tais como nas bibliotecas padrão ou nas bibliotecas privadas de um grupo de programadores trabalhando em um projeto particular.
- O código objeto produzido pelo compilador C ou C++ contém, tipicamente, “buracos” por causa dessas partes ausentes. Um **ligador (*linker*)** liga o código objeto com o código das funções ausentes para produzir um programa executável (sem partes ausentes).
- Se um programa é compilado e ligado corretamente, é produzida uma **imagem executável**.

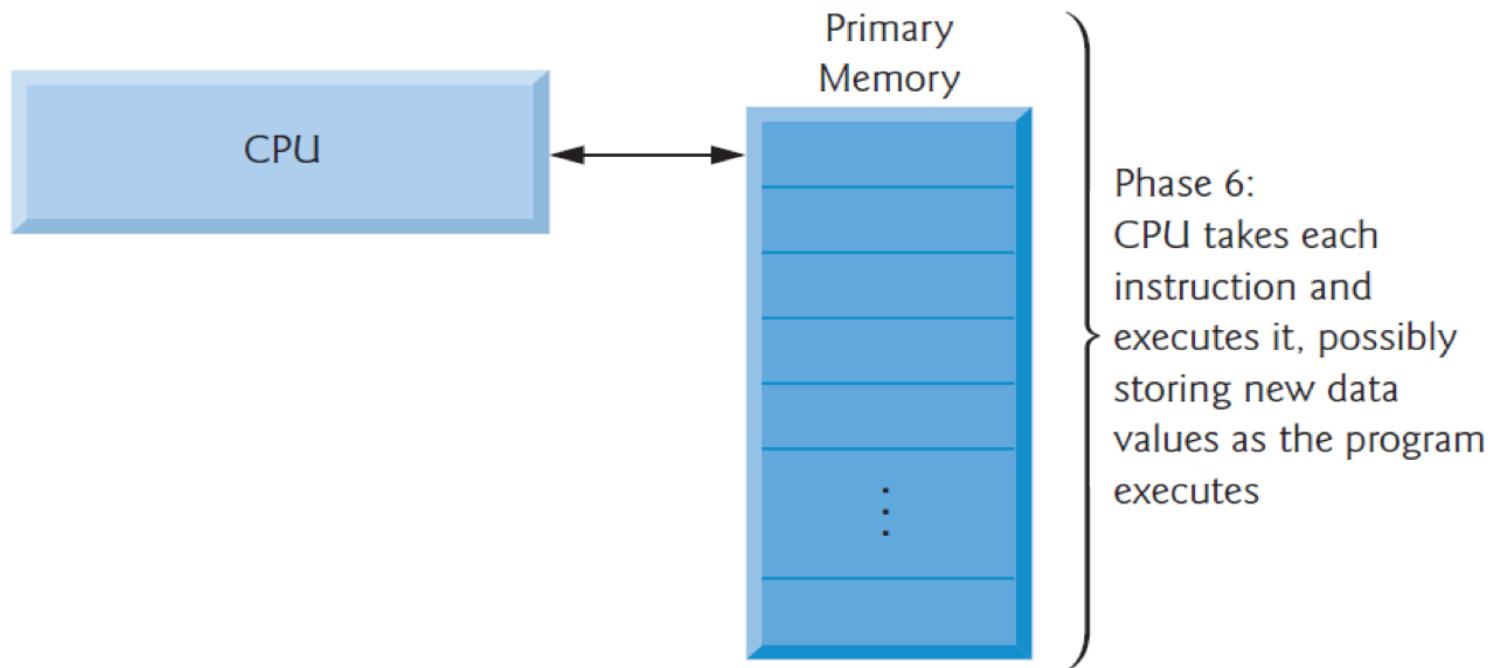
Um ambiente típico de desenvolvimento C/C++

- **Fase 5: Carga (*loading*)**
 - Antes de um programa ser executado, ele deve ser primeiramente colocado na memória (primária). Isto é feito pelo **carregador (*loader*)**, que toma a imagem executável do disco e a transfere para a memória.
 - Os componentes adicionais das bibliotecas compartilhadas, que proveem suporte ao programa, também são carregados.

Um ambiente típico de desenvolvimento C/C++

- **Fase 6: Execução**

- Finalmente, o computador, sob o controle de sua CPU, executa o programa, uma instrução por vez.
- A maioria das arquiteturas de computadores atuais podem executar várias instruções em paralelo.



2

Variáveis, operadores,
comandos de entrada/saída

Declaração de Variáveis

- Toda variável tem **tipo, nome, endereço de memória e valor**.
- Uma variável deve ser declarada com um identificador e um tipo de dado antes de ser usada no programa.
- Se já existir um valor armazenado na variável e um novo valor for atribuído a esta variável, esse valor **sobrescreve** o valor anterior.
- Exemplo: `int number;`
 - O tipo `int` especifica que o valor armazenado é do tipo inteiro (valor inteiro).
 - O identificador `number` é o nome da variável.
- Pode-se declarar várias variáveis em uma mesma linha:
 - `int number1, number2, number3, number4;`

Endereços

- Ao declararmos uma variável x, ela será associada a:
 - Um nome (exemplo: `x`)
 - Um endereço de memória ou referência (exemplo: `0xbfd267c4`)
 - Um valor (exemplo: `9`)

```
1 int x = 9;
```

- Para acessar o endereço de uma variável, utilizamos o operador &

Operador de atribuição

- `sum = number1 + number2;`
 - O símbolo ‘`=`’ é um operador de atribuição.
 - Avalia-se a expressão matemática do lado direito do ‘`=`’ e atribui-se o resultado à variável do lado esquerdo.
 - `=` e `+` são operadores binários; requerem dois operandos.
- **Dica:** coloque espaços em branco em ambos os lados de um operador binário para facilitar a leitura do programa.

Operadores aritméticos

Operação	Operador aritmético	Exemplo	Exemplo em C/C++
Adição	+	$f + 7$	<code>f + 7</code>
Subtração	-	$p - c$	<code>p - c</code>
Multiplicação	*	bm ou $b \times m$	<code>b * m</code>
Divisão	/	x/y ou $x \div y$ ou $\frac{x}{y}$	<code>x / y</code>
Módulo	%	$r \bmod s$	<code>r % s</code>

Observações:

- Operador módulo `%`: resulta no resto da divisão inteira (somente usado com operandos inteiros)
- Exemplo: $7 \% 4$ é igual a 3

Operadores de igualdade e relacionais

Operador algébrico de igualdade ou relacional padrão	Operador de igualdade ou relacional em C++	Exemplo de condição em C++	Significado da condição em C++
<i>Operadores relacionais</i>			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	≥=	x ≥ y	x é maior que ou igual a y
≤	≤=	x ≤ y	x é menor que ou igual a y
<i>Operadores de igualdade</i>			
=	==	x == y	x é igual a y
≠	!=	x != y	x não é igual a y

Operadores lógicos

Operador	Expressão	Nome	Descrição
!	$\neg p$	NÃO (negação)	$\neg p$ é falso, se p é verd.; $\neg p$ é verd., se p é falso.
<code>&&</code>	$p \ \&\& \ q$	E (conjunção)	$p \ \&\& \ q$ é verdadeiro, se ambos, p e q são verd.; é falso, caso contrário.
<code> </code>	$p \ \ q$	OU (disjunção)	$p \ \ q$ é verdadeiro, se p , q ou ambos é verd.; é falso, caso contrário.

A função printf

Uso de `printf`:

- `printf(formato, valor/variável);`

Exemplo:

```
1 printf("%d", 10);
```

(note que `%d` é usado para números inteiros)

A função printf

Alguns possíveis formatos para o comando `printf`:

- `"%d"`: `int` (número inteiro)
- `"%ld"`: `long long` (número inteiro)
- `"%f"`: `float` (ponto flutuante)
- `"%lf"`: `double` (ponto flutuante)
- `"%c"`: `char` (caractere)
- `"%s"`: `string` (cadeia de caracteres)

Formatando a saída

Caracteres especiais:

- `\n`: quebra de linha, ou seja, passa para a linha debaixo;
- `\t`: tabulação horizontal, equivalente a um **tab**;
- `\\"`: aspas duplas;
- `\'`: aspas simples ou apóstrofo;
- `\\\`: barra invertida
- `\a`: ???**beep** ;)

A função scanf

Uso de `scanf`:

- `scanf(formato, endereços de memória);`

Exemplo:

```
1 int x;  
2 scanf("%d", &x);
```

(note que `%d` é usado para números inteiros)

A função scanf

Porquê os códigos abaixo geram **erros**?

Erros comuns

```
1 int x;  
2 scanf(x);
```

```
1 double valor = 10.0;  
2 scanf(valor);
```

- ➊ `scanf` deve receber um texto/formato (entre aspas), não um `int` ou `double` (seja valor ou variável).
- ➋ `scanf` deve receber um endereço de memória, e não um valor.

A função scanf

E os códigos a seguir? Também geram **erros**?

Erros comuns

```
1 int x;  
2 scanf("%d", x);
```

```
1 double valor = 10.0;  
2 scanf("%lf", valor);
```

Sim: **scanf** deve receber **endereços de memória**, não valores.

3

Fluxuogramas

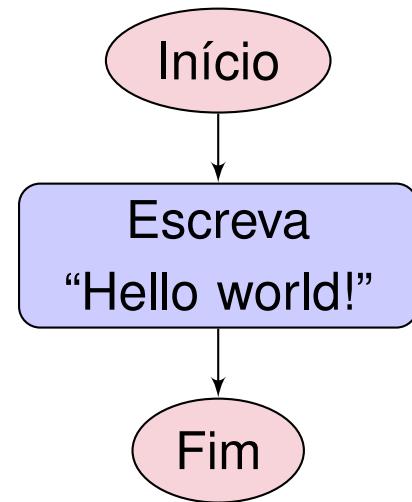
Fluxogramas

- Os fluxogramas são representações gráficas dos programas.
- São utilizados para nos ajudar a compreender um programa.
- Não estão associados a um linguagem específica.
- Apresentam a lógica do algoritmo e não as instruções da linguagem.
- Utilizam diferentes tipos de blocos para indicar os comandos (entradas, saídas, processamentos, decisões, etc) e setas para indicar a sequência de execução.

Estrutura básica de um programa em C/C++

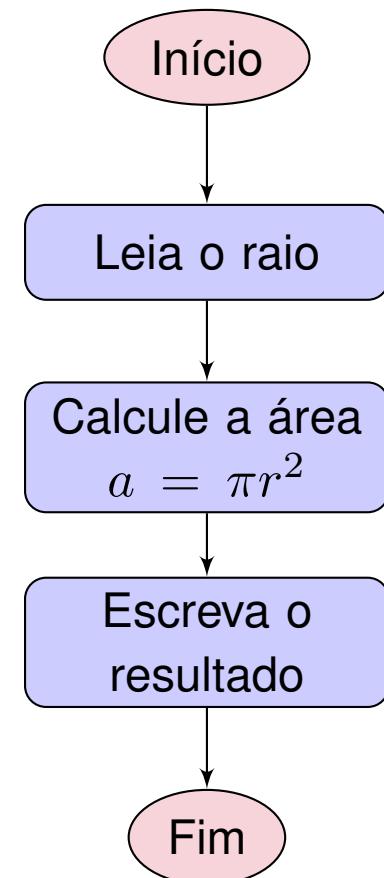
- Exemplo: fluxograma de um “Hello World”

```
1 // Meu Primeiro Programa
2
3 #include <stdio.h>
4
5 int main()
6 {
7     // comentário explicativo
8     printf("Hello world!\n");
9     return 0;
10 }
```



Solução do Exemplo 1:

```
1  /* Programa que calcula a área de um círculo
2   */
3
4 #include <stdio.h>
5
6 int main()
7 {
8     // declaração da constante Pi
9     const double PI = 3.141592;
10    double raio;
11
12    printf("Digite o raio do círculo: ");
13    scanf("%lf", &raio);
14
15    // calculando e imprimindo a área
16    double area = PI * raio * raio;
17    printf("\nÁrea do círculo: %lf\n", area);
18
19    return 0;
20 }
```



4

Condicionais (if, if-else, switch)

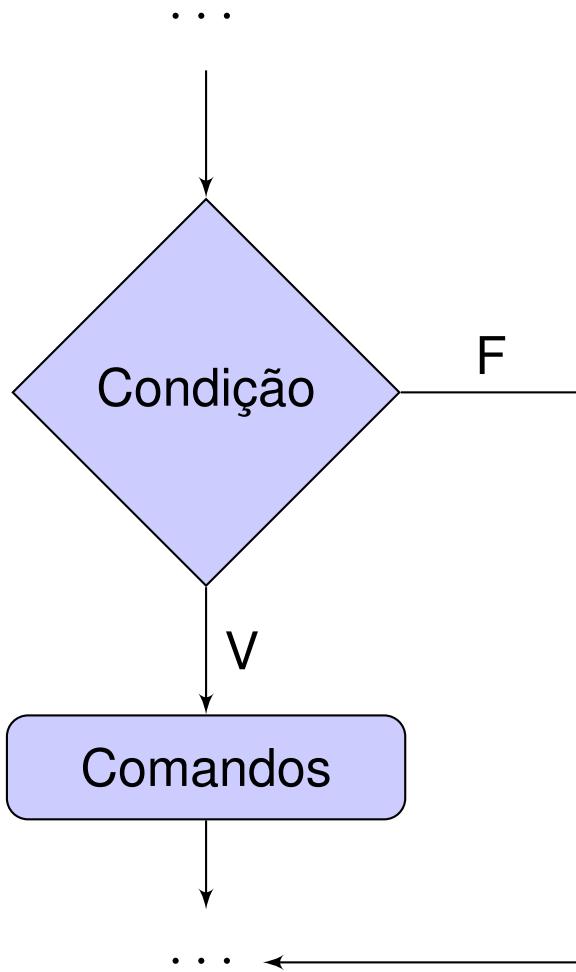
Sintaxe do comando if

```
1 if ( <expressão_de_teste> )
2     instrução_única;
```

OU

```
1 if ( <expressão_de_teste> )
2 {
3     instrução1;
4     instrução2;
5     instrução3;
6     ...
7 }
```

Tomada de decisão



```
1 // Programa que verifica se um no. é par ou ímpar
2 #include <stdio.h>
3
4 int main()
{
5     int numero; //variável para armazenar o número
6     printf("Digite um número inteiro: ");
7     scanf("%d", &numero);
8
9     // testa se o número é par
10    if (numero % 2 == 0) {
11        printf("\nO número %d é par.\n", numero);
12    }
13
14    // testa se o número é ímpar
15    if (numero % 2 != 0) {
16        printf("\nO número %d é ímpar.\n", numero);
17    }
18
19    return 0;
20}
21
```

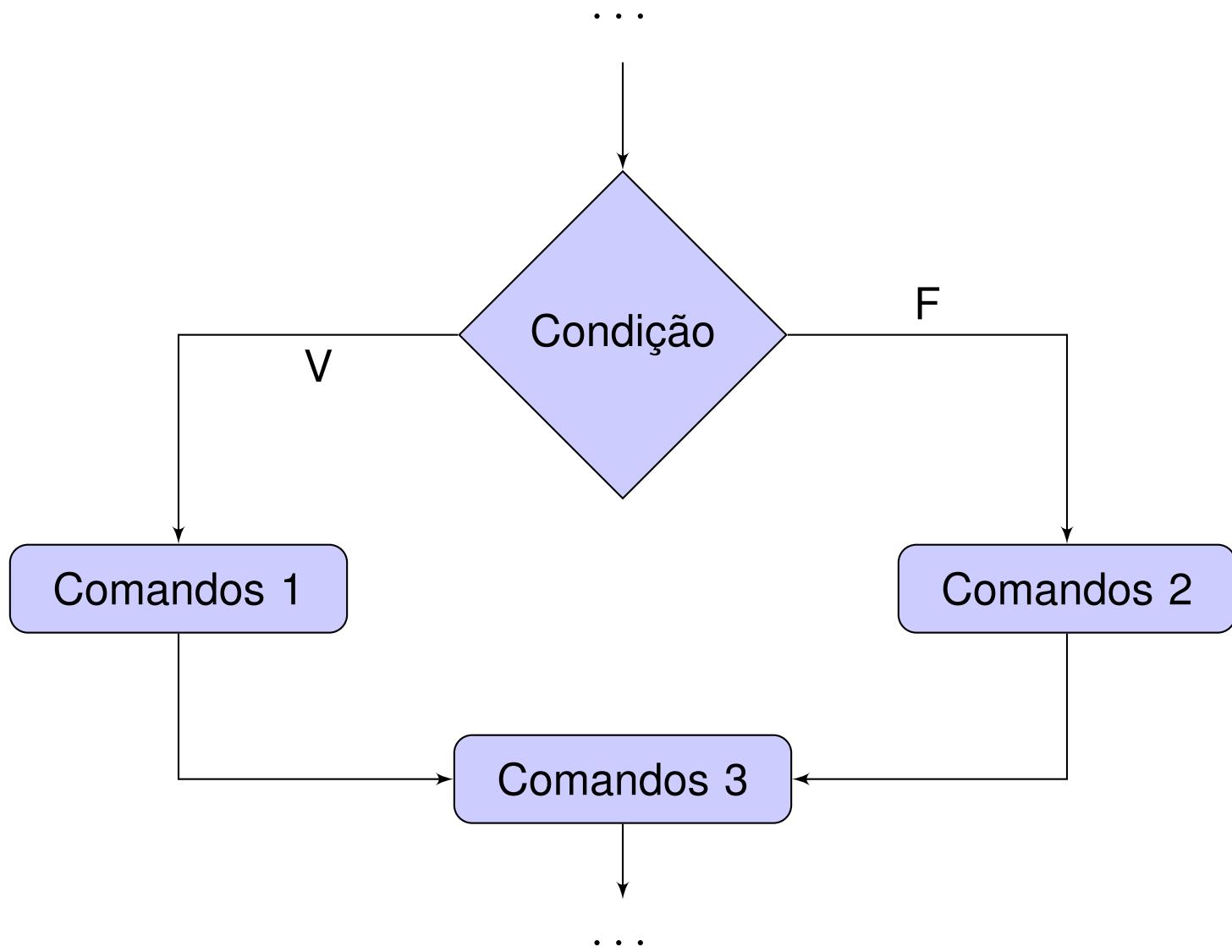
Sintaxe do comando if-else

```
1  if ( <expressão_de_teste> )
2      instrução_única_V;
3  else
4      instrução_única_F;
```

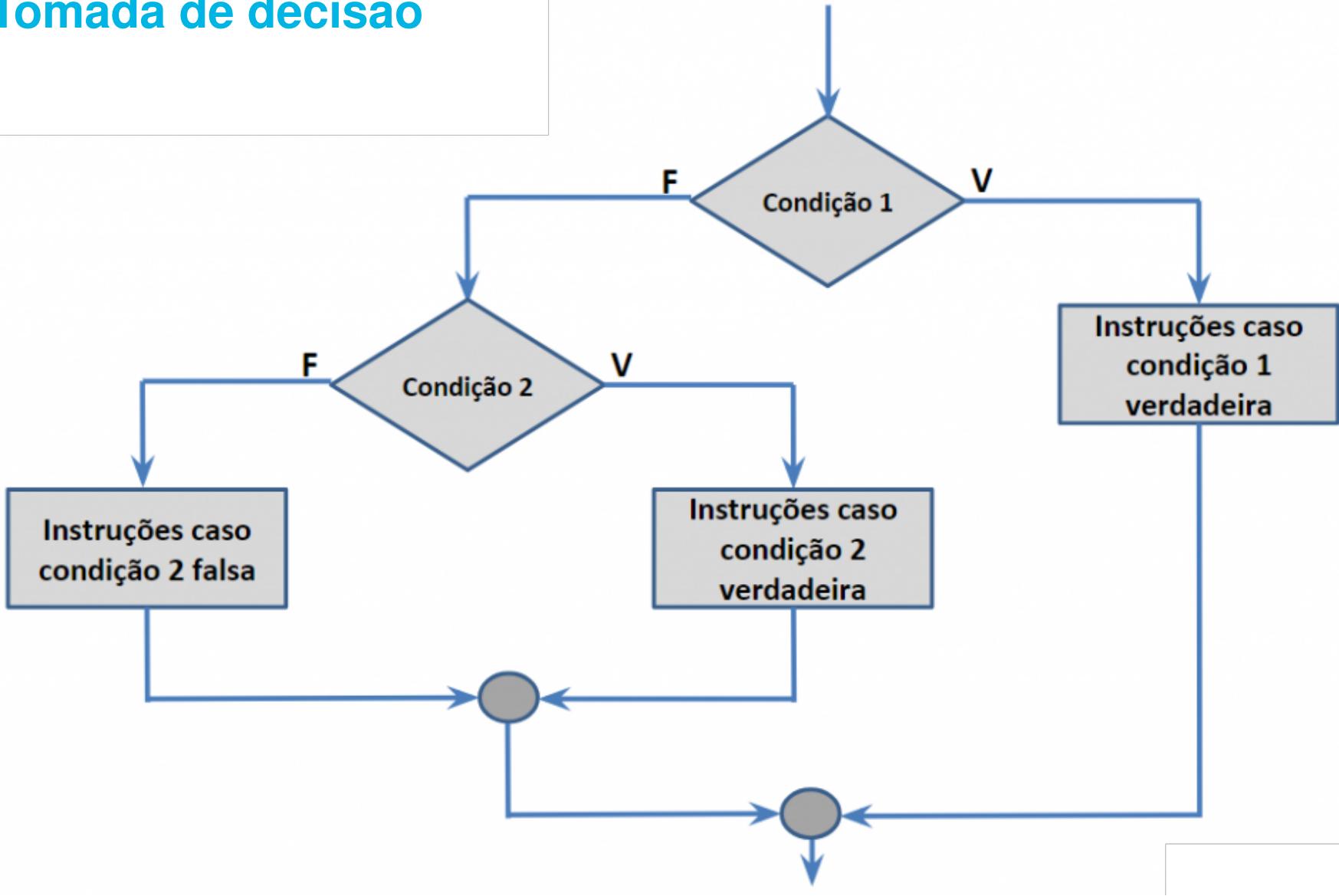
OU

```
1  if ( <expressão_de_teste> )
2  {
3      instrução_V1;
4      ...
5      instrução_Vn;
6  }
7  else
8  {
9      instrução_F1;
10     ...
11     instrução_Fn;
12 }
```

Tomada de decisão



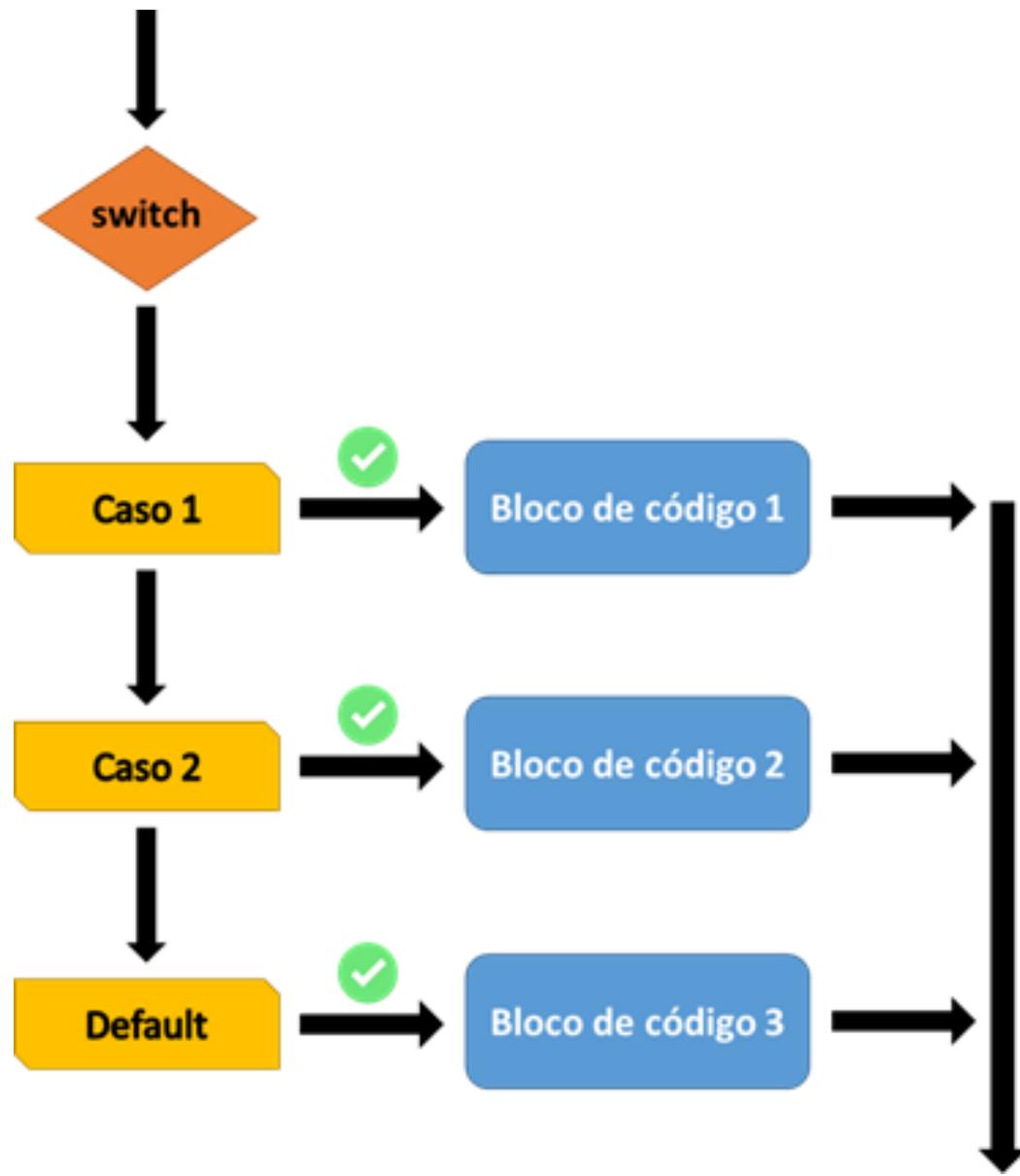
Tomada de decisão

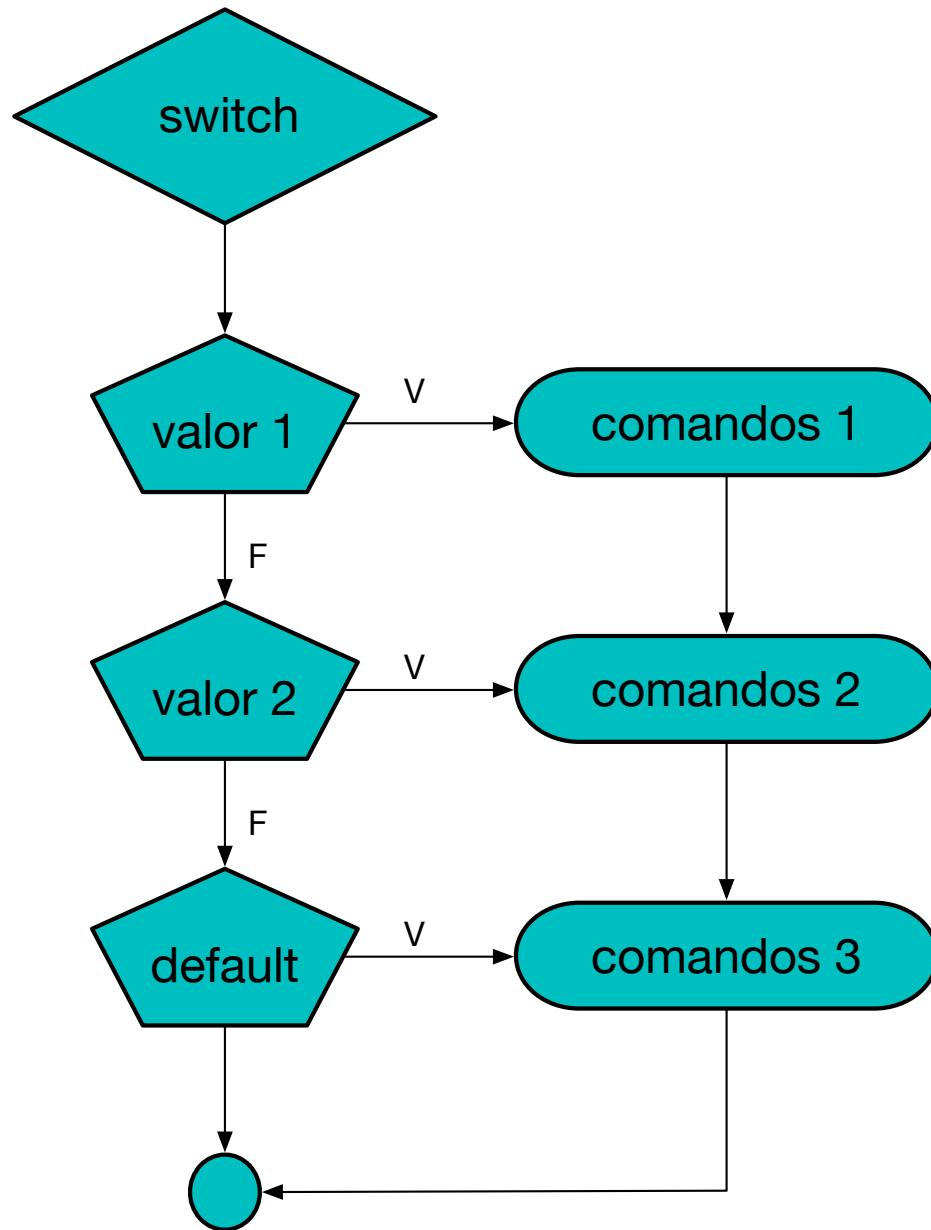


```
1 #include <stdio.h>
2
3 int main()
4 {
5     int numero; //variável para armazenar o número
6
7     printf("Digite um numero inteiro: ");
8     scanf("%d", &numero);
9
10    // se o número for par...
11    if (numero % 2 == 0)
12        printf("\nO número %d é par.\n", numero);
13
14    // caso contrário
15    else
16        printf("\nO número %d é ímpar.\n", numero);
17
18    return 0;
19 }
```

Sintaxe do switch

```
1  switch (op) {  
2      case valor1:  
3          comandos1;  
4          break;  
5      case valor2:  
6          comandos2;  
7          ...  
8      default:  
9          comandosN;  
10 }
```





```
1 //Verifica se uma letra é vogal ou consoante
2
3 int main()
4 {
5     char letra;
6     printf("Digite uma letra: ");
7     scanf("%d", &letra);
8
9     switch (letra) {
10         case 'a':
11         case 'e':
12         case 'i':
13         case 'o':
14         case 'u':
15             printf("Vogal\n");
16             break;
17         default:
18             printf("Consoante\n");
19     }
20     return 0;
21 }
```

Observações sobre o switch

- O `switch` só permite comparar expressões com **constants**.
- Se precisarmos comparar com variáveis ou verificar faixas de valores, devemos usar o comando `if`.
- Se não usarmos o comando `break` em cada case o programa continuará até o fim do bloco.

5

Funções

Exemplos de uso de funções

```
1 // função que calcula a raiz quadrada
2 double x = sqrt(y);
3
4 // função para gerar números aleatórios
5 int numero = rand();
6
7 // definição da função principal de um programa
8 int main() { ... }
```

Bibliotecas C/C++ são compostas de funções, de forma a permitir que o programador reproveite códigos existentes.

Protótipo

Definição Geral de uma Função

```
1 <tipo_retorno> <nome_função>(<lista_declaracão_parâmetro>)
2 {
3     <corpo_função>
4 }
```

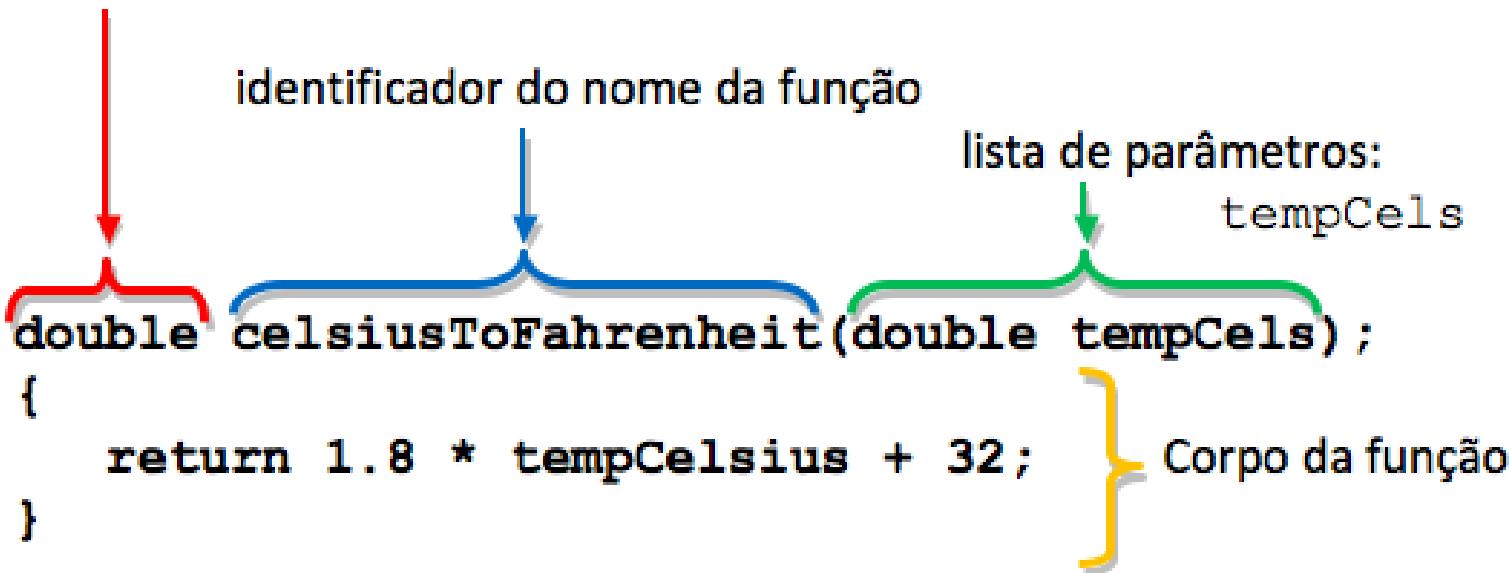
Onde:

- **<tipo_retorno>**: é o tipo do valor que a função retorna; quando a função não retorna nenhum valor utiliza-se a palavra chave **void**.
- **<nome_função>**: é o identificador que nomeia a função.
- **<lista_declaracão_parâmetro>**: é uma lista, possivelmente vazia, de declarações separadas por vírgulas, dos parâmetros da função.
- **<corpo_função>**: descreve o comportamento da função.

Definição de funções

Exemplo: Definição da função `celsiusToFahrenheit()`

tipo do retorno da função



Exemplo: conversão de temperaturas usando função

```
1 // definição da função
2 double celsiusToFahrenheit(double tempCels)
3 {
4     double f;
5     f = 1.8 * tempCels + 32;
6     return f;
7 }
```

ou

```
1 // definição da função
2 double celsiusToFahrenheit(double tempCels)
3 {
4     return 1.8 * tempCels + 32;
5 }
```

Exemplo: conversão de temperaturas usando função

```
1 #include <stdio.h>
2
3 // protótipo da função
4 double celsiusToFahrenheit(double tempCels);
5
6 // método main (principal)
7 int main()
8 {
9     double tempC, tempF;
10    printf("Conversão Celsius para Fahrenheit\n");
11    printf("(valor menor que -273.15 encerra o programa)\n\n");
12    printf("Temperatura em Celsius: ");
13    scanf("%lf", &tempC);
14
15    if (tempC >= -273.15) {
16        tempF = celsiusToFahrenheit(tempC);
17        printf("%lf graus Celsius = %lf graus Fahrenheit.\n",
18               tempC, tempF);
19    }
20    return 0;
21 }
```

Como alterar o valor da variável dentro da função

Nós já utilizamos uma função que faz isso...

```
1 int main()
2 {
3     int x;
4     scanf("%d", &x); // passamos o endereço de memória de x: &x
5
6     if (x % 2 == 0)
7         printf("%d é um número par!\n", x);
8     else
9         printf("%d é um número ímpar!\n", x);
10 }
```

Passagem por valor

Qual o problema da função a seguir?

```
1 void naoTroca(int a, int b)
2 {
3     int aux = a;
4     a = b;
5     b = aux;
6 }
```

- Os parâmetros são passados por valor!
- Assim, cópias de **a** e **b** são passadas para a função.
- Logo: a função não efetua a troca de fato!

Passagem de ponteiros

Ao contrário de C++, C não implementa passagem por referência...

- A solução é utilizar **ponteiros** para simular a passagem por referência.

```
1 void troca(int *a, int *b)
2 {
3     int aux = *a;
4     *a = *b;
5     *b = aux;
6 }
```

- A função recebe ponteiros para duas variáveis.
- Em seguida, troca o conteúdo das memórias apontadas.

Como usar essas funções?

Eis um exemplo de uso das funções apresentadas:

```
1 int main()
2 {
3     int a = 1;
4     int b = 2;
5     naoTroca(a, b); // valores a e b são passados (e não há troca)
6     printf("a = %d, b = %d", a, b); // a = 1, b = 2
7 }
```

```
1 int main()
2 {
3     int a = 1;
4     int b = 2;
5     troca(&a, &b); // endereço de memória de a e b são passados
6     printf("a = %d, b = %d", a, b); // a = 2, b = 1
7 }
```

6

Ponteiros

Ponteiros

Breve revisão:

Um ponteiro (apontador ou *pointer*) é um tipo especial de variável que armazena um **endereço de memória**

- Ponteiros são declarados utilizando o caractere especial *****:

```
1 int *pi;      // pi é um ponteiro do tipo int
2 char *pc;     // pc é um ponteiro do tipo char
3 float *pf;    // pf é um ponteiro do tipo float
4 double *pd;   // pd é um ponteiro do tipo double
```

- Vários podem ser declarados em uma única linha:

```
1 int *p1, *p2, *p3;
```

Ponteiros

O **conteúdo** da memória apontada por um ponteiro se refere ao valor armazenado no endereço de memória para o qual o ponteiro aponta.

- Este conteúdo (valor) pode ser alterado usando o operador *:

```
1 int main()
2 {
3     int x = 10, y = 100;
4     int *px = &x;
5     → *px = *px + 1; // conteúdo de px recebe o conteúdo de px mais 1
6     printf("x = %d", x);
7     printf("y = %d", y);
8     return 0;
9 }
```

- O que será impresso?

```
1 x = 11
2 y = 100
```

Memória

Endereço	Valor
00010000	??
00010001	??
00010002	??
00010003	??
00010004	??
00010005	??
00010006	??
00010007	??
00010008	??
00010009	??
0001000A	??
0001000B	??
0001000C	??
0001000D	??
0001000E	??
0001000F	??

Exemplo:

- Note os quatro ponteiros...

```
1 int main()
2 {
3     char *c;
4     int *i;
5     float *f;
6     double *d;
7     return 0;
8 }
```

- Todos requerem o mesmo tamanho (32/64 bits).
- Lembre-se: um ponteiro armazena um **endereço de memória**, independente do tipo.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004		
0x1008		
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     → int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004		
0x1008		
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     → naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     → naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		x
0x1020		y
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     → int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016	100	x
0x1020	200	y
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     → aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016	100	x
0x1020	200	y
0x1024		aux
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

The memory dump shows the state of variables at different stages of execution. The first four rows (0x1000-0x1008) represent the initial values in the main function. The next four rows (0x1012-0x1016) show the state after the call to naoTroca(). The final four rows (0x1020-0x1024) show the state after the call to troca(). Red brackets on the right side group the variables x and y under the main function, and green brackets group the variables x, y, and aux under the naoTroca function. The aux variable is explicitly shown in the memory dump for the naoTroca scope.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     → x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016	100	x
0x1020	200	y
0x1024	100	aux
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

The memory dump shows the state of variables at different stages of execution. The first four rows (0x1000-0x1008) represent the initial values in the main function. The next four rows (0x1016-0x1024) represent the state after the call to naoTroca. The last four rows (0x1040-0x1048) are empty. Red curly braces on the right side group the first four rows under 'main' and the next four under 'naoTroca'. A blue arrow points to the assignment statement 'x = y;' in the naoTroca function code.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016	200	x
0x1020	200	y
0x1024	100	aux
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 } →
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016	200	x
0x1020	100	y
0x1024	100	aux
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

The memory dump shows the state of variables at different addresses. The first two rows (0x1004 and 0x1008) are highlighted in red and grouped by a red brace on the right labeled "main". The next three rows (0x1016, 0x1020, and 0x1024) are highlighted in green and grouped by a green brace on the right labeled "naoTrocada". The other rows are empty.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     → printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra a memória com endereços de 4 bytes (0x1000 a 0x1056). As células para os endereços 0x1004 e 0x1008 estão destacadas com um fundo vermelho, e uma grande chama vermelha à direita delas indica que elas pertencem ao escopo da função main().

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     → troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra o conteúdo da memória no programa. As linhas para endereços 0x1004 e 0x1008 estão destacadas com um fundo vermelho, e uma grande chave vermelha à direita delas indica que elas pertencem ao escopo da função main().

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     ←→ troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		px
0x1036		py
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032	0x1004	px
0x1036	0x1008	py
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

The table shows memory addresses from 0x1000 to 0x1056. It contains four rows highlighted with red outlines: 0x1004 (content 100, name x), 0x1008 (content 200, name y), 0x1032 (content 0x1004, name px), and 0x1036 (content 0x1008, name py). A red brace on the right side groups the first two rows under the label "main". A blue brace on the right side groups the last two rows under the label "troca".

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     →aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032	0x1004	px
0x1036	0x1008	py
0x1040		aux
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra o estado da memória ao longo do programa. Os endereços estão na coluna 'Endereço', os valores na coluna 'Conteúdo' e os nomes das variáveis na coluna 'Nome'. A coluna 'Nome' é dividida em duas seções: 'main' (de 0x1004 a 0x1008) e 'troca' (de 0x1032 a 0x1040). Um cursor aponta para a linha 12 do código, que é a linha de chamada para a função 'troca'. Na tabela, a linha 0x1032 (que corresponde ao endereço da variável 'px' no escopo 'troca') tem um valor de 0x1004, que é o endereço da variável 'x' no escopo 'main'. Isto ilustra que os ponteiros contêm endereços de variáveis, não os valores das variáveis.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     →*px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	100	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032	0x1004	px
0x1036	0x1008	py
0x1040	100	aux
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra o conteúdo da memória em hexa. Os endereços 0x1004 e 0x1008 são destacados com um fundo vermelho e agrupados por uma chave vermelha rotulada "main". Os endereços 0x1032, 0x1036, 0x1040 e 0x1044 são destacados com um fundo azul e agrupados por uma chave azul rotulada "troca".

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	200	x
0x1008	200	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032	0x1004	px
0x1036	0x1008	py
0x1040	100	aux
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra o estado da memória ao longo da execução do programa. Os endereços estão na coluna 'Endereço', os valores na coluna 'Conteúdo' e os nomes das variáveis na coluna 'Nome'. A coluna 'Nome' é dividida em duas seções: 'main' (de 0x1004 a 0x1008) e 'troca' (de 0x1032 a 0x1040). Um cursor aponta para a linha 14 do código, que é a chamada à função 'troca'. Na tabela, a linha 0x1032 (que corresponde ao endereço da variável 'px' no escopo 'troca') tem seu conteúdo (0x1004) colorido azul, e a linha 0x1036 (que corresponde ao endereço da variável 'py' no escopo 'troca') também tem seu conteúdo (0x1008) colorido azul.

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 } →
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	200	x
0x1008	100	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032	0x1004	px
0x1036	0x1008	py
0x1040	100	aux
0x1044		
0x1048		
0x1052		
0x1056		

A tabela mostra o conteúdo da memória em hexa. Os endereços 0x1004 e 0x1008 são destacados com um fundo vermelho e agrupados por uma chave vermelha rotulada "main". Os endereços 0x1032, 0x1036 e 0x1040 são destacados com um fundo azul e agrupados por uma chave azul rotulada "troca".

Exemplo de execução

```
1 void naoTroca(int x, int y)
2 {
3     int aux;
4     aux = x;
5     x = y;
6     y = aux;
7 }
8
9 void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome
0x1000		
0x1004	200	x
0x1008	100	y
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		

A table showing memory addresses (Endereço) from 0x1000 to 0x1056. The row for address 0x1004 contains the value 200 and is labeled 'x'. The row for address 0x1008 contains the value 100 and is labeled 'y'. A red brace on the right side of the table groups these two rows together and is labeled 'main'.

7

Diretivas de compilação e a biblioteca padrão de C

Diretivas

#include

- Inclui outro arquivo (geralmente bibliotecas) em nosso código-fonte.
- Na prática, o pré-processador vai substituir a diretiva `#include` pelo conteúdo do arquivo indicado.

Diretivas

#define

- Em sua forma mais simples, define constantes simbólicas com nomes mais apropriados.
- Quando um identificador é associado a um `#define`, todas as suas ocorrências no código-fonte são substituídas pelo valor da constante.
- Note que `#define` também pode ser utilizado para criar diretivas mais elaboradas, inclusive aceitando argumentos, chamadas **Macros**.

Exemplo

```
1 // incluindo a biblioteca stdio
2 #include <stdio.h>
3
4 // definindo o valor de PI
5 #define PI 3.141592
6
7 // definindo o que é um 'beep'
8 // (obs: há formas mais elaboradas de fazer um 'beep')
9 #define BEEP "\x07"
10
11 int main()
12 {
13     printf("pi = %d\n", PI);
14     printf(BEEP);
15     return 0;
16 }
```

Biblioteca Matemática – Parte I

Algumas funções matemáticas disponíveis na biblioteca `math.h`.

Para usá-las é necessário: `#include <math.h>`

Função	Descrição	Exemplo
<code>double ceil(x)</code>	arredonda x para cima	<code>ceil(9.1) → 10.0</code>
<code>double floor(x)</code>	arredonda x para baixo	<code>floor(9.8) → 9.0</code>
<code>double round(x)</code>	arredonda x	<code>round(9.5) → 10.0</code> <code>round(9.4) → 9.0</code>
<code>double trunc(x)</code>	retorna a parte inteira de x	<code>trunc(9.8) → 9.0</code>

Biblioteca Matemática – Parte II

Funções para potências:

Função	Descrição	Exemplo
double pow(x, y)	x elevado a y: x^y	pow(3, 2) → 9.0
double sqrt(x)	raiz quadrada de x: \sqrt{x}	sqrt(25) → 5.0
double cbrt(x)	raiz cúbica de x: $\sqrt[3]{x}$	cbrt(27) → 3.0

Biblioteca Matemática – Parte III

Funções trigonométricas:

Função	Descrição	Exemplo
double cos(x)*	retorna o cosseno x	$\cos(1.047) \rightarrow 0.5$
double sin(x)*	retorna o seno x	$\sin(1.571) \rightarrow 1.0$
double tan(x)*	retorna a tangente x	$\tan(0.785) \rightarrow 1.0$
double acos(x)**	retorna o arco cosseno	$\text{acos}(0.5) \rightarrow 1.047$
double asin(x)**	retorna o arco seno	$\text{asin}(1.0) \rightarrow 1.571$
double atan(x)**	retorna o arco tangente	$\text{atan}(1.0) \rightarrow 0.785$

*: valores em radianos

**: valores de x entre $[-1, 1]$

Biblioteca Matemática – Parte IV

Funções Exponenciais e Logarítmicas:

Função	Descrição	Exemplo
double exp(x)	retorna exponencial de x: e^x	$\exp(5) \rightarrow 148.4$
double log(x)	logaritmo natural de x: $\ln(x)$	$\log(5.5) \rightarrow 1.7$
double log10(x)	logaritmo de x: $\log(x)$	$\log10(1000) \rightarrow 3.0$

Biblioteca <stdlib.h>

Provê funções para alocação de memória (usaremos muito no futuro), controle de processos, conversão, etc.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int i = atoi("-10"); // atoi converte string -> int
7     float f = atof("10.5"); // atof converte string -> float
8     double d = strtod("10.5", NULL); // strtod converte string -> double
9
10    system("clear"); // executa o comando clear no terminal
11
12    srand(0); // seleciona a semente para geração de nros aleatórios
13    int r = rand(); // r recebe um nro aleatório
14    printf("Número aleatório: %d\n\n", r);
15
16    printf("i = %d, f = %f, d = %lf\n", i, f, d);
17    printf("Valor absoluto de i: %d\n\n", abs(i));
18
19    exit(0); // função que finaliza o programa imediatamente
20    return 0;
21 }
```



Perguntas?