

Variáveis e Tipos

Aula 03

Marcos Silvano Almeida

marcossilvano@professores.utfpr.edu.br

Departamento de Computação

UTFPR Campo Mourão


Roteiro

- Revendo primeiro programa
 - Avisos e erros do compilador
- Comentários
- Tipos
- Variáveis
- Scanf()

Revendo: primeiro programa

```
#include <stdio.h>           // inclui biblioteca necessária

int main() {                 // início do programa (abre bloco)
    printf("Primeiro programa\n"); // linhas são terminadas em ;
    return 0;                // retorna 0 (sucesso) ao sistema
}                             // final do programa (fecha bloco)
```

- O código **não compila** se **contiver erros**.
 - a. O próprio editor (VS Code) aponta alguns erros.
 - b. Devemos aprender a compreender as mensagens do compilador, pois nos auxiliam a identificar e corrigir os erros.
 isto é parte essencial do trabalho do desenvolvedor (DEBUG)

Qual é o erro?

>> CÓDIGO 01

```
#include <stdio.h>

int main() {
    printf("Primeiro programa\n")
    return 0;
}
```

>> CÓDIGO 02

```
#include <stdio.h>

int main() {
    printf("Primeiro programa\n");
    return 0;
}
```

>> CÓDIGO 03

```
include <stio.h>

int main() {
    printf("Primeiro programa\n");
    return 0;
}
```

>> CÓDIGO 04

```
#include <stdio.h>

int man() {
    printf("Primeiro programa\n");
}
```

Desenvolvendo software

- O processo de programação envolve alguns passos:



- ① Compreender o que deve ser feito / problema a ser resolvido
- ② Projetar a solução
- ③ Escrever o código
- ④ Testar
- ⑤ Depurar (debug): encontrar e resolver erros

- 🙌 Desenvolver programas é um **processo iterativo**
 - Repetido por completo para cada parte do programa: ① ⇒ ② ⇒ ③ ⇒ ④ ⇒ ⑤
 - Podemos retroceder a qualquer passo, se necessário
 - E realizar os passos seguintes
 - Muito comum os passos ③ ⇒ ④ ⇒ ⑤ serem “iterados” várias vezes

Comentários

```
#include <stdio.h>                // inclui biblioteca necessária
// Comentário de uma linha
/*
Comentário de múltiplas linhas.
Comentários não são compilados ou executados pelo computador.
Utilizamos para acrescentar explicações OU para desabilitar parte do
código, sem apagá-lo (caso queiramos utilizá-lo no futuro).
*/
int main() {                       // início do programa (abre bloco)
    printf("Primeiro programa\n"); // linhas são terminadas em ;
    return 0;                     // retorna 0 (sucesso) ao sistema
}                                 // final do programa (fecha bloco)
```

Tipos e Variáveis

Tipos de Dados

- Os dados armazenados e processados pelo computador são representados como sequências de Bytes
 - 1 Byte = 8 bits
- Utilizamos o sistema decimal: 10 símbolos \Rightarrow 0 1 2 3 4 5 6 7 8 9
 - Circuitos eletrônicos digitais utilizam sistema binário: 2 símbolos \Rightarrow 0 1
 - Nas trilhas, 1 e 0 representam a presença (ou não) de eletricidade
- Tipos mais comuns de dados
 - Números
 - Caracteres (letras, símbolos, ...)
 - Texto
 - Booleano (verdadeiro/falso)

Tipos de Dados (mais comuns)

Tipo	Descrição	Tamanho	Literal
int	número inteiro	4 Bytes = 32 bits	0, -5, 5678
float	número real (6 casas decimais)	4 Bytes = 32 bits	3.4f, -0.005
char	caractere da tabela ASCII	1 Byte = 8 bits	'A', '5', 'b', '#', ')
int	valor booleano (inteiro)	4 Bytes = 32 bits	0 (false) e 1 (≠ 0, true)
string	texto (não é um tipo simples)	1 Byte/caractere	"João Sauro", "Res: 15.6"

- Valor literal
 - Valor escrito no código, ao invés de ser calculado pela lógica do programa
- No início, usaremos o tipo **String** apenas para imprimir mensagens e formatação
 - Até chegarmos na aula de manipulação de **Strings...**

Tipos de Dados

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

As faixas de valores podem variar de acordo com a plataforma ou compilador.

Variáveis

- Variável: espaço de memória alocado para armazenar dados

```
int numero = 5;
```

```
float pi = 3.141593;
```

```
char letra = 'M';
```

```
<< tipo nome = valor_inicial;
```

- **Tipo:** indica um dos tipos aceitos na linguagem (int, float, char, ...)
 - **Tamanho:** o tipo indica o espaço ocupado na memória.
- **Nome/Identificador:**
 - Pode conter letras, números e underline
 - Não pode conter espaço ou iniciar com número (evite símbolos).
 - Não pode ser uma palavra reservada da linguagem.
- **Alocação da memória:** quando declaramos uma variável, um espaço de memória referente ao seu tamanho é reservado na RAM.

Variáveis: nomes/identificadores

```
#include <stdio.h>

int main() {

    // variáveis válidas

    int a = 5;

    float media = 0;        // não iniciada

    int x, y, pos;          // múltiplas variáveis em uma linha

    float min = 0, max = 0;

    // mais de uma palavra

    int valorFinal = 0;     // notacao "camelo" (CamelCase)

    float mediaFinal = 0;

    int valor_final = 0;    // usando underline como separador

    float media_final = 0;

    float total_de_produtos = 0;

}
```

Variáveis: declaração

```
#include <stdio.h>

int main() {

    // declarando variáveis
    int a = -5;
    char ch = '#';
    const float pi = 3.141593;
    int b; // AVISO: não se deve criar variável sem iniciá-la

    // imprimindo valores das variáveis
    printf("\nIMPRIMINDO VARIÁVEIS\n\n");
    printf("variavel int:    %d\n", a);
    printf("variavel int:    %d\n", b);
    printf("variavel float: %f\n", pi);
    printf("variavel char:   %c\n", ch);
    return 0;
}
```

const é útil para armazenarmos valores padrões.

Memória RAM

a	-5
ch	'#'
pi	3.141593
b	?

Variáveis: atribuição

```
#include <stdio.h>

int main() {

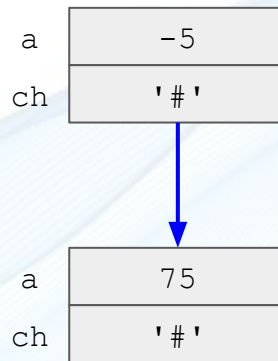
    // declaração + inicialização de variáveis
    int a = -5;
    char ch = '#';

    // atribuindo variáveis
    a = 75; // atribuição

    // podemos imprimir múltiplas
    // variáveis em um único printf
    // IMPORTANTE: observe a ordem!
    printf("Valores\n a = %d\n ch = %c\n\n", a, ch);

    return 0;
}
```


Memória RAM



Variáveis: atribuição

- Atribuição: **copia** o valor da **direita** para a posição de memória indicada pela variável à **esquerda**.

```
a = 75;    // atribuição
```



- Inicialização: define o valor inicial da variável
 - Pode apresentar diferenças em relação à atribuição
- Constantes
 - Não permitem a alteração posterior do valor
 - Úteis para guardarmos valores de referência

```
const float pi = 3.141593;    // constante
pi = 3.14;    // error: assignment of read-only variable 'pi'
                // pi = 3.14;
                //      ^
```


scanf(): entrada de valores pelo teclado

- Função scanf()
 - Permite ler dados do teclado
 - Interessante para ler números (int e float)
 - Problemas para leitura de caracteres e strings
 - Trataremos de soluções mais adiante na disciplina
 - Leitura ocorre até ser encontrada uma quebra de linha ('\n') ou espaço (' ')

- Uso

```
scanf(string_de_formato, &variávelA, &variávelB...);
```

- Exemplo

```
int a;  
scanf("%d", &a);  
printf("a = %d\n", a);
```


Exemplo scanf()

```
#include <stdio.h>

int main() {
    int a;
    float b;
    char c;

    printf("Digite char: ");    // se a leitura do char for após um número,
    scanf("%c", &c);           // termos problemas, pois o '\n' (tecla ENTER)
    printf("c = %c\n", c); // é um caractere válido

    printf("Digite int: ");    // lendo int
    scanf("%d", &a);
    printf("a = %d\n", a);

    printf("Digite float: ");  // lendo float
    scanf("%f", &b);
    printf("b = %f\n", b);
}
```

Exemplo scanf(): múltiplas leituras

```
#include <stdio.h>

int main() {
    int a;
    float b;
    char c;

    // podemos realizar a leitura de
    // vários valores em um único scanf
    printf("Digite int, float e char: ");

    // IMPORTANTE: observe a ordem!
    scanf("%d %f %c", &a, &b, &c);
    printf("a = %d, b = %.2f, c = '%c'\n", a, b, c);
}
```



Operadores

Aritméticos

Lógicos

Relacionais

Seletores

Indentação

Repetição