

Tipos em C

Aula 03

Marcos Silvano Almeida
marcoossilvano@utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

Roteiro

- Comentários
- Tipos
- Variáveis
- Scanf()

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 em C

Tipos de Dados (mais comuns)

Tipo	Descrição	Tamanho	Literal
int	número inteiro	4 Bytes/32 bits	0, -5, 5678
long	número inteiro longo	8 Bytes/64 bits	0, -5, 5678
float	real aproximado (6 casas), mantissa x 2^{exp}	4 Bytes/32 bits	3.4f, -0.005f
double	real aproximado (15 casas), mantissa x 2^{exp}	8 Bytes/64 bits	3.4, -0.005
char	caractere da tabela ASCII	1 Byte/8 bits	'A', '5', 'b', '#', ')'
string	sequência de caracteres	1 Byte por char	"João Sauro", "R: 15.6"
int	valor booleano (inteiro)	4 Bytes/32 bits	0 (false) e 1 (\neq 0, true)

- **Valor literal**
 - Valor escrito no código, ao invés de ser calculado pela lógica do programa
- Modificadores para inteiros:
 - short (2 bytes), long (8 bytes), unsigned (somente positivos)

Imprimindo valores de diferentes tipos com printf()

```
#include <stdio.h>

int main() {

    printf("\nIMPRIMINDO VALORES LITERAIS \n\n"); // string = texto

    printf("int.....: %d\n", 5);
    printf("string.....: 5 \n\n");

    printf("float.....: %f\n", 17.3f);
    printf("float.....: %f\n\n", 17.3); // (double)

    printf("char.....: %c\n", 'a');
    printf("(int)char.: %d\n\n", 'a');

    printf("boolean....: %d\n\n", 5 == 5); // 5 é igual a 5?

    return 0;
}
```

Variáveis

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: declaração

```
#include <stdio.h>

int main() {

    // declarando variáveis
    int a = -5;
    char ch = '#';
    const float pi = 3.141593;
    int b; // variável não iniciada: contém lixo de memória

    // imprimindo valores das variáveis
    printf("\nVARIÁVEIS\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	?

Signed vs Unsigned

- Quando definimos o valor para uma variável, os bits são armazenados de acordo com tal valor.
 - O significado do valor como número negativo ou positivo depende do uso que fizermos do mesmo

```
// short:    2 bytes
// signed:   [-32768, 32767]
// unsigned: [0, 65535]
short a = -10;
```

```
printf("unsigned short: %hu\n", a); // 65526
printf("signed short: %hd\n", a); // -10
```

```
printf("unsigned short: %hu\n", a + 9); // 65535
printf("signed short: %hd\n", a + 9); // -1
```

1111 1111 0110 = 65526 = -10

	unsigned	signed
1111 1111 0110 =	65526	-10
+ 0000 0000 1001 =	9	9

1111 1111 1111 =	65535	-1

Modificadores de tipos inteiros

```
char a = 'c'; // 99 // 1 byte [-128, 127]
```

```
printf("char.....: %c\n", a);
```

```
unsigned char b = 256; // 1 byte [0, 255]
```

```
printf("unsigned char.....: %d\n", b);
```

```
short int c = -1; // 2 bytes [-32768, 32767] ~ -32k..32k
```

```
printf("short int.....: %hd\n", c);
```

```
unsigned short int d = -1; // 2 bytes [0, 65535] ~ 0..65k
```

```
printf("unsigned short int.: %hu\n", d);
```

```
int e = 4123123123; // 4 bytes [-2147483648, 2147483647] ~ -2bi..2bi
```

```
printf("int.....: %d\n", e);
```

```
unsigned int f = 4123123123; // 4 bytes [0, 4294967295] ~ 0..4 bi
```

```
printf("unsigned int.....: %u\n", f);
```

```
long long g = 4123123123123123; // 8 bytes [-9223372036854775808, -9223372036854775808] ~ -9qua..9qua
```

```
printf("long long.....: %lld\n", g);
```

```
unsigned long long h = 4123123123123123; // 8 bytes [0, 18446744073709551615] ~ 0..18 qui
```

```
printf("unsigned long long.....: %llu\n", h);
```


Tamanho mínimo garantido

char	8 bits
short	16 bits
int	16 bits
long	32 bits
long long	64 bits

Variáveis: expressão de atribuição

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

```
a = 75;    // expressão de 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;  
//      ^
```

Conversões de tipos (*type casting*)

- Conversões de estreitamento/ampliação

```
float num1 = 5.432f;
```

```
int num2;
```

```
float num3;
```

```
// Conversão de estreitamento:
```

```
// >> corta parte fracionária (perda de dados)
```

```
num2 = num1;
```

```
printf("num2: %d\n", num2);
```

```
printf("cast: %d\n", (int)num1);
```

```
// Conversão de ampliação:
```

```
// >> muda representação de int para float
```

```
num3 = num2;
```

```
printf("num3: %f\n", num3);
```

```
printf("cast: %f\n", (float)num2);
```

Entrada de valores pelo teclado

scanf(): entrada de valores pelo teclado

- Função scanf()
 - Permite ler dados do teclado
 - Permite a leitura de tipos da linguagem (int, float, char, ...)
 - 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);
```

Espaço no início, antes do símbolo de formatação, impede que o '\\n seja lido como entrada.

Exemplo scanf(): leituras separadas

```
#include <stdio.h>

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

    printf("Digite char: ");
    scanf(" %c", &c);           // lendo char
    printf("c = %c\n", c);

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

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