

Variáveis e Operadores

Introdução à Programação - 2025/1

Professor Marciano

Bacharelado em Sistemas de Informação
Instituto de Informática - UFG

Sumário

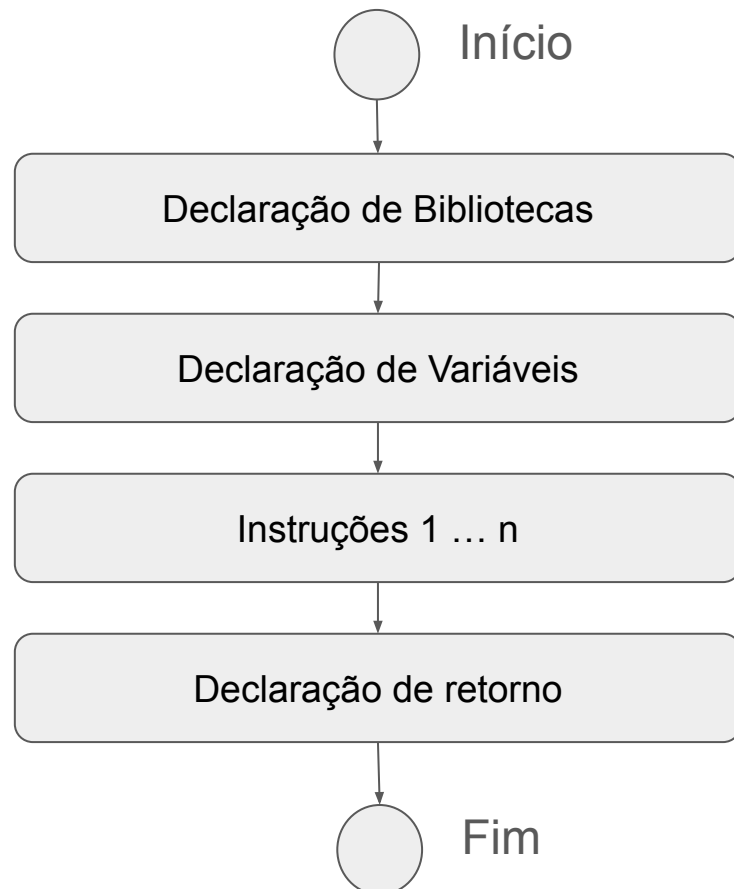
- Estrutura Básica de um Programa em C
- Variáveis
- Operadores de Atribuição e Aritméticos
- Valores Constantes
- Comandos de Entrada e Saída
- Variáveis Constantes e Macros
- Bibliotecas `<math.h>` e `<stdlib.h>`
- Exemplo e Exercícios

Estrutura Básica de um Programa em C

Estrutura Básica de um Programa em C

- A estrutura básica é a seguinte:

```
1  //Declaração de Bibliotecas
2
3  //Declaração de variáveis globais
4
5  int main (/*Declaração de parâmetros*/ ) {
6      //Declaração de variáveis locais
7
8      /* Instrução 01
9       * Instrução 02
10     * Instrução 03
11     * .
12     * .
13     * .
14     * Instrução 'n'
15     */
16
17     //Declaração do retorno e fim do programa
18 }
```



Estrutura Básica de um Programa em C

- Exemplo

```
1  #include <stdio.h>
```

Declaração de bibliotecas

```
2
```

```
3  int main ( ) {
```

```
4      int a;
```

```
5      int b, c;
```

Declaração de variáveis locais

```
6
```

```
7      a = 7 + 9;
```

```
8      b = a + 10;
```

```
9      c = b - a;
```

Instruções

```
10 }
```

Estrutura Básica de um Programa em C

- Vamos programar?

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- Como compilar
 1. Abrir o cmd (Windows) ou bash (Linux)
 2. Verificar se o compilador gcc está instalado
 - `gcc --version`
 3. Se não estiver instalado, baixar o gcc e instalar
 - <https://bitlybr.com/pYtW>
 4. Se estiver instalado digitar
 - `gcc primeiro.c -o primeiro`
 5. Executar o programa
 - `primeiro.exe` (Windows) ou `./primeiro` (Linux)
- Saída do programa: `Hello World!`

Estrutura Básica de um Programa em C

- O programa primeiro.c, escrito em C, é composto pelos seguintes componentes:

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```


Estrutura Básica de um Programa em C

- **Comentários:** São ignorados pelo compilador e servem para auxiliar o programador a descrever o programa.
 - Uma linha: **// Comentário**
 - Uma ou várias linhas: **/* Bloco de Comentários */**

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- Todas as instruções, sem exceção, terminam com um “;” (ponto-e-vírgula).

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- As chaves “{” e “}” indicam, respectivamente, o início e o fim de um bloco de instruções.

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11      //Indica ao Sistema Operacional que o programa foi executado com sucesso
12      return 0;
13  }
```

Estrutura Básica de um Programa em C

- As palavras de comando da linguagem são palavras reservadas (também chamados de palavras-chave) e são escritos em **letras minúsculas**. **TODAS!**

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- A diretiva de compilação **#include <stdio.h>**, informa ao compilador que ele deve incluir a biblioteca **stdio** (Standard Input/Output) durante o processo de compilação.

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- `int main()`: declara a função principal `main()` que retorna um valor do tipo `int` (número inteiro).
 - **TODO** programa em C necessita incluir a função `main()`.

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```


Estrutura Básica de um Programa em C

- A função **printf** imprime o texto na saída padrão (console/terminal).

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```

Estrutura Básica de um Programa em C

- O comando “**return**” finaliza o programa retornando o valor **0**.
 - **0** → sucesso
 - **1** → erro

```
1  /* Meu primeiro programa em C: primeiro.c
2  * Programa que imprime uma mensagem de boas vindas no console/terminal.
3  */
4  #include <stdio.h> //Inclusão da biblioteca de I/O
5
6  //Função principal. Inicia a execução do programa
7  int main() {
8      //Função para envio de um texto para o console/terminal.
9      printf("Hello World!");
10
11     //Indica ao Sistema Operacional que o programa foi executado com sucesso
12     return 0;
13 }
```


Variáveis

Variáveis

- Definição
 - Variáveis são locais onde armazenamos valores, isto é, são localidades na memória do computador onde pode-se armazenar um valor;
 - Elas são utilizadas para armazenar e manipular dados.

Variáveis

- Toda variável é caracterizada:
 - por um **nome**, que a identifica em um programa;
 - e por um **tipo**, que determina o que pode ser armazenado naquela variável.
- Alguns tipos fundamentais
 - **int** – armazena um número inteiro;
 - **double** – especifica os números reais;
 - **char** – armazena um único caractere minúsculo ou maiúsculo, um dígito, ou um caractere especial.
- Em C, os tipos fundamentais são **palavras reservadas** escritas em **letras minúsculas**.

Variáveis

- As variáveis podem ocupar tamanhos diferentes na memória, **dependo do tipo**, exemplo:

Tipo	Bytes	Intervalo de valores aceito
char	1	0 a 255
short	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.647
long	4	-2.147.483.648 a 2.147.483.647
float	4	$1,2 \times 10^{-38}$ a $3,4 \times 10^{+38}$
double	8	$2,2 \times 10^{-308}$ a $1,8 \times 10^{+308}$

Variáveis

- Os tipos das variáveis podem ser precedidos por prefixos que modificam o tamanho original em bytes.

Tipo	Bytes	Intervalo de valores aceito
short <code>int</code>	2	-32.768 a 32.767
<code>int</code>	4	-2.147.483.648 a 2.147.483.647
long <code>int</code>	4 (arquitetura 32 bits) 8 (arquitetura 64 bits)	-2.147.483.648 a 2.147.483.647 (para 4 bytes) -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 (para 8 bytes)
long long <code>int</code>	8	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
unsigned short <code>int</code>	2	0 a 65.535 (sem sinal)
unsigned <code>int</code>	4	0 a 4.294.967.295 (sem sinal)
unsigned long <code>int</code>	4 (arquitetura 32 bits) 8 (arquitetura 64 bits)	0 a 4.294.967.295 (sem sinal) (para 4 bytes), 0 a 18.446.744.073.709.551.615 (sem sinal) (para 8 bytes)
unsigned long long <code>int</code>	8	0 a 18.446.744.073.709.551.615 (sem sinal)

Variáveis

- Identificador (nome)
 - é o **nome da variável**, e não pode ser uma palavra-chave.
 - É formado por uma **combinação de letras, dígitos e “_” sublinhado** (*underline*), começando **sempre** com uma letra ou “_”.
 - **Case sensitive**: letras maiúsculas e minúsculas são diferentes.
 - Para assegurar a portabilidade use **no máximo 31 caracteres**.
 - Escolha **nomes significativos** para facilitar a documentação e o entendimento do código.

Variáveis

- Identificadores válidos
 - `lampada_ligada;`
 - `SomaTotal;`
 - `_temperatura;`
 - `VALOR_DE_PI;`
 - `variavel1;*`
 - `a;*`
 - `*(identificadores válidos mas pouco significativos)`
- Identificadores não válidos
 - `double;`
 - `4mem;`
 - `12345678;`
 - `main;`

Variáveis

- As seguintes palavras já tem um significado na linguagem C e por esse motivo não podem ser utilizadas como nome de variáveis:

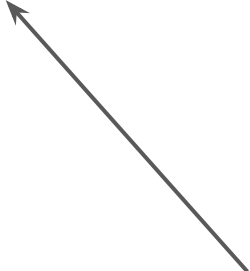
auto	double	int	struct
break	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Variáveis

- Declaração de variáveis
- Declara-se da seguinte forma:

Tipo da Variável Nome da Variável;

- Exemplos corretos:
 - int soma;
 - double preco_abacaxi;
 - char resposta;
- Exemplos incorretos:
 - soma int;
 - double preco_abacaxi
 - resposta char;



Atenção: em C, o sinal gráfico de ponto e vírgula faz parte da declaração da variável.

Variáveis

- 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;`
 - Todas as variáveis serão do mesmo tipo.

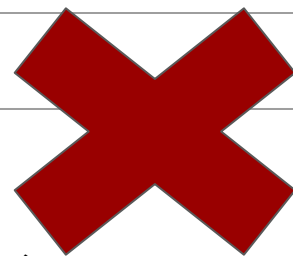
Variáveis

- Onde declarar?
 - Variáveis podem ser declaradas em qualquer lugar de um programa C, mas devem aparecer antes de serem usadas no programa.

Exemplo 1	Exemplo 2
<pre>int x; x = 80; printf("%d", x); int y; y = 60; printf("%d", y);</pre>	<pre>int x; int y; x = 80; y = 60; printf("%d", x); printf("%d", y);</pre>

Exemplo 3

```
x = 80;  
int x;  
printf("%d", x);  
y = 60;  
printf("%d", y);  
int y;
```



Variáveis

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

- Toda variável declarada no programa ocupa um espaço na memória principal do computador;
- Memória
 - A memória é formada por várias células;
 - Cada célula contém um endereço e um valor;

Variáveis

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



i

- Memória
 - A variável `char i` ocupa 1 byte na memória.
 - **TODA** variável do tipo `char` ocupa 1 byte.

```
1  int main()  
2  {  
3      char i;  
4      return 0;  
5  }
```

Variáveis

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

i

- Memória

- O inteiro `int i` ocupa 4 bytes na memória.
- **TODA** variável do tipo `int` ocupa 4 bytes.
 - Exceção: quando precedida pelas palavras-chave `short`, `long` ou `long long`.
 - `short int` (2 bytes)
 - `long int` (4 ou 8 bytes)
 - `long long int` (8 bytes)

```
1  int main()  
2  {  
3      int i;  
4      return 0;  
5  }
```

Variáveis

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

i

- Memória
 - O ponto flutuante `float i` ocupa 4 bytes na memória.
 - **TODA** variável `float` ocupa 4 bytes.

```
1  int main()  
2  {  
3      float i;  
4      return 0;  
5  }
```

Variáveis

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

i

- Memória
 - `double i` ocupa 8 bytes na memória.
 - **TODA** variável `double` ocupa 8 bytes.
 - Exceção: quando precedida pela palavra-chave `long`.
 - `long double` (10, 12 ou 16 bytes)

```
1  int main()  
2  {  
3      double i;  
4      return 0;  
5  }
```


Variáveis

- Ao declararmos uma variável **x**, ela será associada a:
 - Um **nome** (exemplo: **x**)
 - Um **tipo** (exemplo: **int**)
 - 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 &**

Operadores de Atribuição e Aritméticos

Operadores de Atribuição e Aritméticos

- O comando de atribuição (=) serve para atribuir valores para variáveis.
- A sintaxe do uso do comando é:

`variável = valor;`

- Exemplos:
 - `int idade = 5;`
 - `float PI = 3.1415;`
 - `char sexo = 'M';`

Operadores de Atribuição e Aritméticos

- O comando de atribuição pode conter expressões do lado direito:

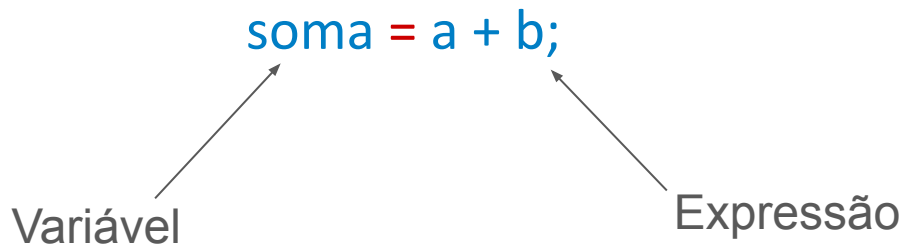
`variável = expressão;`

- Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.

`int soma = numero1 + numero2;`

Operadores de Atribuição e Aritméticos

- No exemplo abaixo, a variável `soma` recebe o valor calculado da expressão `a + b`



- Exemplos:
 - `int anos_para_maioridade = 18 - idade;`
 - `float comprimento_circulo = 2 * PI * raio;`
 - `char sexo_minusculo = 'M' + 32;` → verificar a tabela ASCII

Operadores de Atribuição e Aritméticos

- O sinal de igual '=' no comando de atribuição é chamado de operador de atribuição.

`sum = number1 + number2;`

- 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.
- Boa Prática de Programação: coloque espaços em branco em ambos os lados de um operador binário para facilitar a leitura do programa.

Operadores de Atribuição e Aritméticos

Operação	Operador Aritmético	Exemplo	Exemplo em 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 $x \div y$	<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 Atribuição e Aritméticos

- Quando computamos "a dividido por b", isto tem como resultado um valor p e um resto $r < b$ que são únicos tais que

$$a = p * b + r$$

- Ou seja, a pode ser dividido em p partes inteiras de tamanho b , e sobrar um resto $r < b$.
- Outros exemplos:
 - $5 \% 2$ tem como resultado o valor 1.
 - $15 \% 3$ tem como resultado o valor 0.
 - $1 \% 5$ tem como resultado o valor 1.
 - $19 \% 4$ tem como resultado o valor 3.

Operadores de Atribuição e Aritméticos

- Mais sobre o operador “/”
- Quando utilizado sobre **valores inteiros**, o resultado da operação de divisão **será inteiro**. Isto significa que a parte fracionária da divisão será desconsiderada.
 - $5 / 2$ tem como resultado o valor 2.
- Quando pelo menos um dos **operandos for ponto flutuante**, então a divisão **será fracionária**. Ou seja, o resultado será a divisão exata dos valores.
 - $5.0 / 2$ tem como resultado o valor 2.5.

Operadores de Atribuição e Aritméticos

- Regras da Precedência de Operadores
- São as mesma da álgebra:
 1. Operadores entre parênteses são avaliados primeiro;
 - O parênteses quebra a precedência de um operador.
 2. A seguir, aplicam-se as operações de **multiplicação**, **divisão** e **módulo**.
 - Se uma expressão contém vários desses operadores, as operações são aplicadas da esquerda para a direita.
 3. Por último aplicam-se a **adição** e a **subtração**.
 - Se há vários desses operadores, a aplicação ocorre da esquerda para a direita.

Operadores de Atribuição e Aritméticos

- Regras da Precedência de Operadores

Operação	Operador	Ordem de Avaliação
Parênteses	()	Avaliados primeiro (pares mais internos avaliados antes)
Multiplicação Divisão Módulo	* / %	Avaliados em segundo lugar. Se houver vários, avaliação da esquerda para direita.
Adição Subtração	+ -	Avaliados por último. Se houver vários, avaliação da esquerda para direita.

Operadores de Atribuição e Aritméticos

- $y = 2 * 5 * 5 + 3 * 5 + 7;$

$$2 * 5 = 10$$

(Qual o valor de y?)

(Multiplicação mais à esquerda)

$$y = 10 * 5 + 3 * 5 + 7;$$

$$10 * 5 = 50$$

(Multiplicação mais à esquerda)

$$y = 50 + 3 * 5 + 7;$$

$$3 * 5 = 15$$

(Multiplicação antes da adição)

$$y = 50 + 15 + 7$$

$$50 + 15 = 65$$

(Adição mais à esquerda)

$$y = 65 + 7 \Rightarrow y = 72$$

(Última adição e atribuição de 72 à variável y)

Operadores de Atribuição e Aritméticos

Exemplo 01: Qual o valor da variável **a** ao final da execução do programa?

a. 3

b. 6

c. 8

d. 4



```
1  #include <stdio.h>
2
3  int main () {
4      int a, b, c, d;
5
6      d = 3;
7      c = 2;
8      b = 4;
9      d = c + b;
10     a = d + 1;
11     a = a + 1;
12 }
```

Operadores de Atribuição e Aritméticos

- Incremento(++) e Decremento(--)
- É muito comum escrevermos expressões para incrementar/decrementar o valor de uma variável por 1.

```
a = a + 1;
```

- Em C, o operador unário ++ é usado para incrementar de 1 o valor de uma variável.

```
a = a + 1;
```

(é o mesmo que **a++;**)

- O operador unário -- é usado para decrementar de 1 o valor de uma variável.

```
a = a - 1;
```

(é o mesmo que **a--;**)

Operadores de Atribuição e Aritméticos

- Há uma diferença quando estes operadores são usados à esquerda ou à direita de uma variável e fizerem parte de uma expressão maior:
- **++a** : Neste caso o valor de **a** será incrementado antes e só depois o valor de **a** é usado na expressão.
- **a++**: Neste caso o valor de **a** é usado na expressão maior, e só depois é incrementado.
- A mesma coisa acontece com o operador **--**.

Operadores de Atribuição e Aritméticos

- O programa à direita imprime “b = 6”.



```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 5, b, c;
6      b = ++a;
7      printf("b = %d \n", b);
8  }
```

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 5, b, c;
6      b = a++;
7      printf("b = %d \n", b);
8  }
```

- O programa à esquerda imprime “b = 5”.



- Em ambos programas o valor final da variável **a** é igual a 6.

Operadores de Atribuição e Aritméticos

- Atribuições simplificadas
- Uma expressão da forma:

$a = a + b;$

- onde ocorre uma atribuição a uma das variáveis da expressão pode ser simplificada como:

$a += b;$

Operadores de Atribuição e Aritméticos

Comando	Exemplo	Corresponde a:
<code>+=</code>	<code>a += b;</code>	<code>a = a + b;</code>
<code>-=</code>	<code>a -= b;</code>	<code>a = a - b;</code>
<code>*=</code>	<code>a *= b;</code>	<code>a = a * b;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>a %= b;</code>	<code>a = a % b;</code>

Operadores de Atribuição e Aritméticos

- Conversão de tipos
- É possível **converter** alguns tipos entre si.
- Existem duas formas de fazê-lo: **implícita e explícita**:
 - **Implícita**: Capacidade (tamanho) do destino deve ser maior que a origem senão há perda de informação.
 - Ex1.:

```
int a;  
short b = 2;  
a = b;
```
 - Ex2:

```
float a;  
int b = 10;  
a = b;
```
 - Ex3:

```
float a = 3.4;  
int b;  
b = a;
```



Operadores de Atribuição e Aritméticos

- Conversão de tipos

- **Explícita**: Explicitamente informa o tipo que o valor da variável ou expressão é convertida.

- Ex.

- `int a, b = 5, c = 2;`
- `a = (int)((float)b / (float)c);`

- Não modifica o tipo “real” da variável, só o valor dela enquanto participando de uma expressão.

- Ex.

- `int a;`
- `(float)a = 1.0;` ← Errado

Valores Constantes

Constantes

- **Constantes** são valores previamente determinados e que por algum motivo, devem aparecer dentro de um programa.
- Assim como as variáveis, as constantes também **possuem um tipo**.
- Os tipos permitidos são exatamente os mesmos das variáveis.
 - Adiciona-se o tipo **string**, que corresponde a uma sequência de caracteres.
- Exemplos de constantes:
85, 0.10, 'c', "Hello, world!"

Constantes

- Uma **constante inteira** é um número na forma decimal, como escrito normalmente
 - Ex: 10, 145, 1000000
- Uma **constante ponto flutuante** é um número real, onde a parte fracionária vem depois de um ponto
 - Ex: 2.3456, 32132131.5, 5.0
- Uma **constante do tipo caracter** é sempre representada por um caracter (letra, dígito, pontuação, etc.) entre aspas simples.
 - Ex: 'A', '!', '4', '('
- Uma **constante do tipo string** é um texto entre aspas duplas
 - Ex: "Hello, world!"

Constantes

- Expressão Simples
- Uma constante é uma expressão e como tal, pode ser atribuída a uma variável (ou em qualquer outro lugar onde uma expressão seja necessária).
- Ex1:
 - `int a;`
 - `a = 10;`
- Ex2:
 - `char b;`
 - `b = 'F';`
- Ex3:
 - `double c;`
 - `c = 3.141592;`
- Ex4:
 - `string s;`
 - `s = "Hello";`

Constantes

- Expressão Simples
- Uma variável também é uma expressão e pode ser atribuída a outra variável.
- Ex:
 - `int a, b;`
 - `a = 5;`
 - `b = a;`


Constantes

- Exemplos de Atribuição
 - OBS: Sempre antes de usar uma variável, esta deve ter sido declarada.
-
- | | |
|--------------------------|-----------------------------|
| ● <code>int a, b;</code> | ● <code>a = 10;</code> |
| ● <code>float f;</code> | ● <code>b = -15;</code> |
| ● <code>char h;</code> | ● <code>f = 10.0;</code> |
| | ● <code>h = 'A';</code> |
| | ● <code>a = b;</code> |
| | ● <code>f = a;</code> |
| | ● <code>a = (b + a);</code> |


Constantes

- Exemplos errados de atribuição

- `int a, b;`
- `float f, g;`
- `char h;`


- `a + b = 10;` 

- `b = -15` 

- `d = 90;` 

- `f = '6';` 

- `h = g;` 

- `a = b + h;` 

o valor da variável h será manipulado como sendo um código ASCII

Comandos de Entrada e Saída

Comandos de Entrada e Saída - printf

- A função `printf`
- A função `printf` é parte da biblioteca `<stdio.h>`:
- Utilizada para imprimir no dispositivo de saída padrão:
 - tela;
 - console;
 - terminal.
- Exemplo de uso:

```
1 printf("Olá mundo!!!\n");
```

Comandos de Entrada e Saída - printf

- Mas.. Como imprimir um número inteiro?

Erros comuns

```
1 printf(10);
```

```
1 int valor = 10;  
2 printf(valor);
```

- Os códigos acima produzirão um **erro**, pois **printf** deve receber um texto/formato (entre aspas), não um inteiro (seja valor ou variável).

Comandos de Entrada e Saída - printf

- Uso de `printf`:

`printf(formato, valor/variável);`

- Exemplo:

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

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

Comandos de Entrada e Saída - printf

- Alguns possíveis formatos para o comando `printf`:
 - `"%d"`: `int` (número inteiro)
 - `"%ld"`: `long int` (número inteiro)
 - `"%f"`: `float` (ponto flutuante)
 - `"%lf"`: `double` (ponto flutuante)
 - `"%Lf"`: `long double` (ponto flutuante)
 - `"%c"`: `char` (caractere)
 - `"%s"`: `string` (cadeia de caracteres)
 - `"%e"`: `float` (em notação científica)

Comandos de Entrada e Saída - printf

- Assim, para imprimir um número inteiro usamos o formato "%d" como texto e indicamos o inteiro como próximo argumento.
- Exemplos:

```
1 printf("%d", 100); // imprime o número inteiro 100
```

```
1 int number = 10;  
2 printf("%d", number); // imprime o valor da variável number
```

```
1 int n1 = 10;  
2 int n2 = 20;  
3 int soma = n1 + n2;  
4 printf("%d\n", soma); // imprime o valor de soma e a quebra de linha
```

Comandos de Entrada e Saída - printf

- Note que é possível mesclar formato com texto, como por exemplo em "O resultado é %d"

```
1 int n1 = 10;  
2 int n2 = 20;  
3 int soma = n1 + n2;  
4 printf("A soma de %d e %d é igual a %d.\n", n1, n2, soma);
```

- Naturalmente, o código acima produzirá a saída:

```
1 A soma de 10 e 20 é igual a 30.
```

Comandos de Entrada e Saída - printf

- Outro exemplo:

```
1 double n1 = 10.8;  
2 double n2 = 19.3;  
3 double soma = n1 + n2;  
4 printf("%lf + %lf = %lf\n", n1, n2, soma);
```

- O código acima produzirá a saída:

```
1 10.8 + 19.3 = 30.1
```

- Note que o caractere ponto (.) é usado para separar os decimais - padrão norte-americano.

Comandos de Entrada e Saída - printf

- A função `printf` permite formatar a saída de dados.
- O usuário pode especificar, entre outros:
 - número de casas decimais;
 - número de caracteres ocupados pela impressão.
- Exemplos:
 - `"%3d"`: um `int` usando no mínimo 3 espaços;
 - `"%-3d"`: um `int` usando no mínimo 3 espaços (alinhado à esquerda);
 - `"%5s"`: uma `string` usando no mínimo 5 espaços;
 - `"%.3f"`: um `float` usando 3 casas decimais;
 - `"%3f"`: um `float` usando no mínimo 3 espaços;
 - `"%5.3f"`: um `float` usando 3 casas decimais e no mínimo 5 espaços.

Comandos de Entrada e Saída - printf

- Exemplo:

```
1 printf("%-3s %8s\n", "Var", "Val");  
2 printf("%-3s %8.1f\n", "x", 10.222);  
3 printf("%-3s %8.1f\n", "y", 20.33);  
4 printf("%-3s %8.1f\n", "z", 30);
```

- Imprimirá na saída:

```
1 Var      Val  
2 x        10.2  
3 y        20.3  
4 z        30.0
```

Comandos de Entrada e Saída - printf

- Caracteres especiais:
 - `\n`: quebra de linha, ou seja, passa para a linha debaixo (*enter*);
 - `\t`: tabulação horizontal, equivalente a um *tab*;
 - `\"`: aspas duplas;
 - `\'`: aspas simples ou apóstrofo;
 - `\\`: barra invertida
 - `\a`: ???beep ;)
 - `\b`: retrocesso (*backspace*)
 - `\r`: voltar ao início da linha
 - `%%`: porcentagem

Comandos de Entrada e Saída - printf

- Exemplo de impressão de tabela:

```
1 printf("Var \t Val\n");  
2 printf("x \t 10\n");  
3 printf("y \t 20\n");  
4 printf("z \t 30\n");
```

- Resultado:

1	Var	Val
2	x	10
3	y	20
4	z	30

Comandos de Entrada e Saída - scanf

- A função `scanf`
- A função `scanf` também é parte da biblioteca `<stdio.h>`:
 - Utilizada para ler da entrada padrão (terminal).
 - O `scanf` tem algumas (grandes) diferenças em relação ao `printf`:
 - A função `printf` imprime texto e o **valor de variáveis**.
 - A função `scanf` **altera o conteúdo** das variáveis.
 - Alterar conteúdo equivale a **modificar o que está na memória**.
 - Por esta razão, **SEMPRE** passamos um endereço de memória para a função `scanf`.

Comandos de Entrada e Saída - scanf

- Uso de `scanf`:

`scanf(formato, endereços de memória);`

- Obter o `endereço de memória` de uma variável é fácil:
 - utilizamos o operador `&`.
- Exemplo:

```
1 int x;  
2 scanf("%d", &x); // &x retorna o endereço de memória de x
```

- ("`%d`" é usado para números inteiros)

Comandos de Entrada e Saída - scanf

- A função `scanf` usa os mesmos “formatos” que `printf`.
- Exemplos:
 - `"%d"`: `int` (número inteiro)
 - `"%ld"`: `long int` (número inteiro)
 - `"%f"`: `float` (ponto flutuante)
 - `"%lf"`: `double` (ponto flutuante)
 - `"%c"`: `char` (caractere)
 - `"%s"`: `string` (cadeia de caracteres)

Comandos de Entrada e Saída - scanf

- Porquê os códigos abaixo geram erros?

Erros comuns

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

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

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

Comandos de Entrada e Saída - 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**.

Comandos de Entrada e Saída - scanf

- Assim, para ler da entrada padrão usamos um “formato” e indicamos o endereço de memória como próximo argumento.
- Exemplos:

```
1 int x;  
2 scanf("%d", &x); // lê um inteiro da entrada padrão
```

```
1 char c;  
2 scanf("%c", &c); // lê um caractere da entrada padrão
```

```
1 int n1, n2, soma;  
2 scanf("%d %d", &n1, &n2); // lê dois inteiros da entrada padrão  
3 soma = n1 + n2;  
4 printf("A soma de %d e %d eh igual a %d", n1, n2, soma);
```

Variáveis Constantes e Macros

Variáveis Constantes e Macros

- Exemplo:
- Faça um programa em C, para calcular a **área de um círculo**. A área de um círculo é dada pela seguinte fórmula $a = \pi r^2$. O valor do raio **r** será digitado pelo usuário.

[CÓDIGO]

Variáveis Constantes e Macros

```
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      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 }
```


Variáveis Constantes e Macros

- A palavra-chave `const` assegura que a variável associada não será alterada em todo o programa.
- Esse qualificador é indicado para declarar valores constantes.
- **Obrigatoriamente**, as variáveis associadas ao qualificador `const` devem ser inicializadas.

Variáveis Constantes e Macros

```
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 }
```

Variáveis Constantes e Macros

- Pré-processador e diretivas
- O pré-processador é um programa que examina o código-fonte antes de o mesmo ser compilado;
- As diretivas do pré-processador são recursos que usamos para tornar nossos programas mais claros e fáceis de manter.
- São também sinais para o pré-processador de que algo deve ser alterado no código-fonte antes da compilação.

Variáveis Constantes e Macros

- `#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.

Variáveis Constantes e Macros

- `#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**...

Variáveis Constantes e Macros

```
1 // incluindo a biblioteca stdio
2 #include <stdio.h>
3
4 // definindo o valor de PI
5 #define PI 3.14159265359
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 = %f\n", PI);
14     printf(BEEP);
15     return 0;
16 }
```

Bibliotecas `<math.h>` e `<stdlib.h>`

Bibliotecas <math.h> e <stdlib.h>

- Como calcular πr^2 ?

```
double area = PI * raio * raio;
```

- E πr^5 ?

```
double area = PI * raio * raio * raio * raio * raio;
```

- E πr^{50} ?

```
double area = PI * raio * raio * raio * raio * raio * ...;
```



Bibliotecas <math.h> e <stdlib.h>

- As linguagens C não possuem um operador para potência, mas possuem uma **biblioteca** com diversas funções matemáticas, para usá-la devemos incluir a biblioteca **<math.h>**.
- A função para potência é a **pow()**, sintaxe:

```
double pow(double base, double expoente);
```

Bibliotecas <math.h> e <stdlib.h>

- Exemplo

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159265359

int main() {
    ...
    double raio = 10;
    double area = PI * pow(raio, 2);
    ...
}
```

Bibliotecas <math.h> e <stdlib.h>

- Algumas funções matemáticas disponíveis na biblioteca <math.h>. Para usá-las é necessário: `#include`

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

Bibliotecas <math.h> e <stdlib.h>

- Exemplo: Dada a tabela abaixo com os os valores de x, escreva os valores retornados pelas funções.

x	round(x)	floor(x)	ceil(x)	trunc(x)
2.3	2.0	2.0	3.0	2.0
3.8	4.0	3.0	4.0	3.0
5.5	6.0	5.0	6.0	5.0
-2.3	-2.0	-3.0	-2.0	-3.0
-3.8	-4.0	-4.0	-3.0	-3.0
-5.5	-6.0	-6.0	-5.0	-5.0

Bibliotecas <math.h> e <stdlib.h>

- Funções para potências:

Função	Descrição	Exemplo
<code>double exp(double x)</code>	exponencial de x: e^x	<code>exp(5)</code> → 148.4
<code>double pow(double x, double y)</code>	x elevado a y: x^y	<code>pow(3, 2)</code> → 9.0
<code>double sqrt(double x)</code>	raíz quadrada de x: \sqrt{x}	<code>sqrt(25)</code> → 5.0
<code>double cbrt(double x)</code>	raíz cúbica de x: $\sqrt[3]{x}$	<code>cbrt(27)</code> → 3.0

Bibliotecas `<math.h>` e `<stdlib.h>`

- Funções trigonométricas:

Função	Descrição	Exemplo
<code>double cos(double x)</code>	cosseno de x	<code>cos(1.047) → 0.5</code>
<code>double sin(double x)</code>	seno de x	<code>sin(1.571) → 1.0</code>
<code>double tan(double x)</code>	tangente de x	<code>tan(0.785) → 1.0</code>
<code>double acos(double x)</code>	arco cosseno de x	<code>acos(0.5) → 1.047</code>
<code>double asin(double x)</code>	arco seno de x	<code>asin(1.0) → 1.571</code>
<code>double atan(double x)</code>	arco tangente de x	<code>atan(1.0) → 0.785</code>

- OBS: valor de `x` em radianos
- OBS2: o valor de `x` para as funções `acos`, `asin` e `atan` deve estar entre -1 e 1.

Bibliotecas `<math.h>` e `<stdlib.h>`

- Funções logarítmicas:

Função	Descrição	Exemplo
<code>double log(double x)</code>	logaritmo natural de x : $\log_e(x)$	$\log(5.5) \rightarrow 1.7$
<code>double log2(double x)</code>	logaritmo de x : $\log_2(x)$	$\log_2(8) \rightarrow 3.0$
<code>double log10(double x)</code>	logaritmo de x : $\log(x)$	$\log_{10}(1000) \rightarrow 3.0$

Bibliotecas <math.h> e <stdlib.h>

- Biblioteca <stdlib.h>
- A biblioteca stdlib.h nos fornece várias **funções úteis** para manipulação de memória, geração de números aleatórios, execução de comandos no sistema, etc.
 - Hoje vamos conversar sobre geração de **números aleatórios!** (na verdade, vamos gerar números **pseudo-aleatórios**)
 - O primeiro passo é incluir/importar **<stdlib.h>**.

Bibliotecas <math.h> e <stdlib.h>

- A geração de números **pseudo-aleatórios** utiliza um valor como semente e um algoritmo para gerar números que **parecem** aleatórios.
 - Se conhecermos a semente, podemos prever quais números serão gerados pelo algoritmo...
 - Ainda assim, estes geradores são muito úteis!
- Como gerar números realmente aleatórios então?
 - Podemos utilizar **dados externos** (imprevisíveis).
 - Ou até mesmo uma semente baseada em dados externos, **imprevisíveis**!

Bibliotecas <math.h> e <stdlib.h>

- A função `srand()` altera a `semente` de números aleatórios.
- A função `rand()` gera um número aleatório entre 0 e `RAND_MAX`.

Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      srand(1); // estamos usando o número 1 como semente
6
7      int sorteio = rand() % 100;
8      printf("Nro aleatorio entre 0 e 99: %d\n", sorteio);
9      return 0;
10 }
```

- Resultado (sempre será o mesmo...):

```
1  Nro aleatorio entre 0 e 99: 7
```

Bibliotecas <math.h> e <stdlib.h>

- Que tal usar a **data/horário atual** como semente?
- A biblioteca <time.h> nos fornece a função **time()**.
- Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main() {
6      srand(time(NULL)); // estamos usando a data/hora atual como semente
7
8      int sorteio = rand() % 100;
9      printf("Nro aleatorio entre 0 e 99: %d\n", sorteio);
10     return 0;
11 }
```

Bibliotecas `<math.h>` e `<stdlib.h>`

- Algumas funções úteis disponibilizadas pela biblioteca `<stdlib.h>`:

Função	Descrição	Exemplo
<code>void abort()</code>	fecha o programa retornando erro	<code>abort()</code>
<code>int abs(int x)</code>	valor absoluto de um inteiro	<code>abs(-10) → 10</code>
<code>void exit(int x)</code>	fecha o programa retornando x	<code>exit(0)</code>
<code>int system(char[] cmd)</code>	executa o comando cmd	<code>system("clear")</code>



Até a próxima aula

Professor Marciano
marciano@ufg.br

INF

INSTITUTO DE
INFORMÁTICA



UFG

UNIVERSIDADE
FEDERAL DE GOIÁS