

MCTA028-15: Programação Estruturada

Aula 2: Variáveis

Wagner Tanaka Botelho

wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com

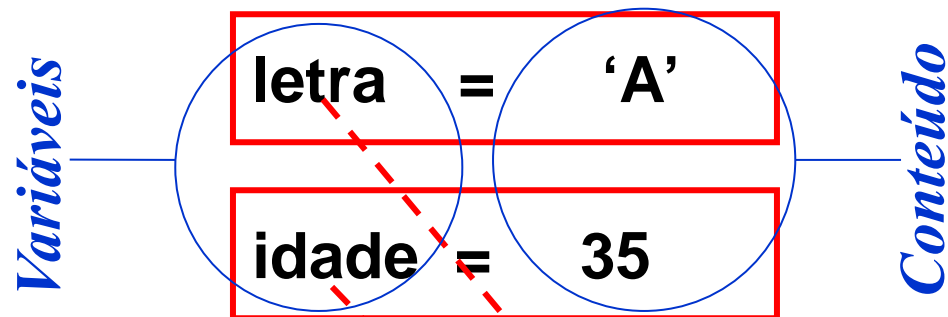
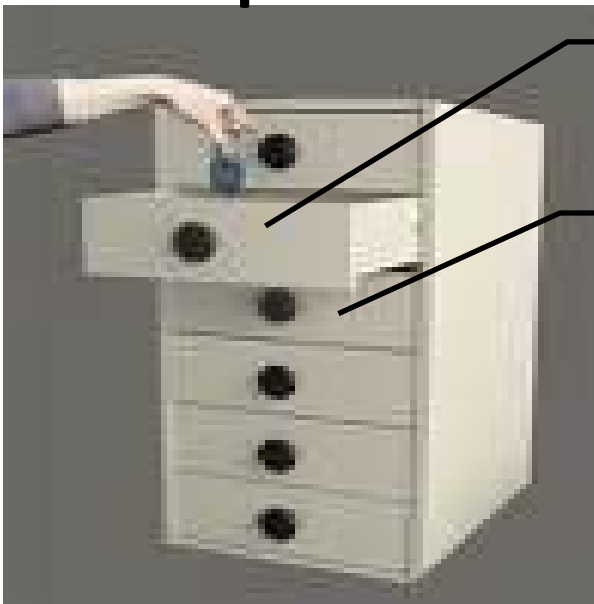
Universidade Federal do ABC (UFABC)

Centro de Matemática, Computação e Cognição (CMCC)

Introdução

- É tudo aquilo que está sujeito a **variações**, que é **inconstante**;
- Serve para **armazenar dados** do programa na **memória** principal;
- Cada **variável** corresponde a uma **posição** de memória, cujo **conteúdo** pode ser **alterado** ao longo do tempo durante a **execução** de um programa;
- Pode **assumir** apenas **um valor** a cada instante.

Memória do Computador



Uma **variável** é um espaço reservado na memória para armazenar um **tipo de dado**.

MEMÓRIA

Endereço	Variável	Conteúdo
.....	
120	char letra	A
121	int idade	35
122	
.....	

Declaração

Declaração

- Para que as variáveis possam **guardar** algum valor, elas precisam ser **declaradas**;
- Toda **variável** deve corresponder a um **tipo de dado**:
 - Sendo assim, uma variável do tipo **inteiro** só poderá **armazenar** valores **inteiros**.

IMPORTANTE!!!

A declaração de variáveis tem que ser **SEMPRE** antes da ser usada e **ANTES** de qualquer instrução.

```
main( ) {  
    Declaração de variáveis  
  
    Instrução_1;  
    Instrução_2;  
    Instrução_n;  
}
```

Declaração

Variável

```
tipo_de_dado var_1;
```

Variáveis

```
tipo_de_dado var_1, var_2, var_n;
```

Definindo o Tipo

Definindo o Tipo

Tipo	Bits	Intervalo de Valores
char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
float	32	1,175494E-038 A 3,402823E+038
double	64	2,225074E-308 A 1,797693E+308

Definindo o Tipo: char

Definindo o Tipo: char

- Permite armazenar **UM ÚNICO CARACTERE** numa variável desse tipo:
- Um dos **erros** mais comuns de programação em C é pensar que o tipo char permite armazenar strings ou **conjuntos de caracteres** numa variável do tipo char.

```
main() {  
    char a;  
    a = 'a';  
}
```

A **representação** de caracteres faz-se utilizando **aspas simples** ' '.

Definindo o Tipo: char

- Também é permitido armazenar o **código (número)** de um caractere do conjunto de caracteres da **Tabela ASCII**. Por exemplo, o número **64** representa o **@** na tabela:

ASCII printable characters			
32	space	64	@
33	!	65	A
34	"	66	B
35	#	67	C

```
main() {  
    char b;  
    b = 64;  
}
```

Se for imprimir a variável b, a saída será o @.

Definindo o Tipo: `int`

Definindo o Tipo: `int`

- Permite armazenar um **número inteiro** (sem parte fracionária);
- Assim como o tipo `char`, o `int` pode ser especificado nas bases **decimal**, **octal** ou **hexadecimal**. Nesta aula, vamos considerar a **base-padrão**, conhecida como **decimal**.

```
main( ){  
    int a, b;  
    a = 1;  
    b = -20;  
}
```

Definindo os Tipos: `float` e `double`

Definindo os Tipos: float e double

- Permitem armazenar um valor real (com **parte fracionária**), também conhecido como **ponto flutuante**;
- A **notação científica** é uma forma de escrever **números** extremamente **grandes** ou **pequenos**;
- A diferença entre float e double é a sua **precisão**:
 - float: precisão **simples**;
 - double: **dupla** precisão.
- O valor é seguido por uma letra “e” ou “E” e um **número inteiro** (**positivo** ou **negativo**):

```
main(){  
    float a;  
    double b;  
    a = 5.25;  
    b = 15.673;  
}
```

Padrão numérico americano, ou seja, a **parte decimal** fica depois de **um ponto**.

```
main(){  
    double b;  
    b = 5.0e10;  
}
```

Equivale a **double b = 500000000000**

Definindo os Nomes

Nomes

➔ Tomar cuidado com a definição dos nomes das variáveis:

```
int idade;           /*CORRETO*/
int Num_Clientes;    /*CORRETO*/
float a1b2c4;        /*CORRETO*/
float 7a2b3c;        /*INCORRETO: primeiro caractere é um dígito*/
char float;          /*INCORRETO: utilizou-se uma palavra reservada*/
double vinte%;       /*INCORRETO: utilizou-se caractere inadmissível*/
char sim?não;        /*INCORRETO: utilizou-se caractere inadmissível*/
char Num, NUM;       /*CORRETO, pois o C é case sensitive. Entretanto, não
                     acho aconselhável, pois você pode confundir a
                     diferença de Num e NUM.*/
```

Atribuindo Valores

Atribuição

- Ao realizar uma **atribuição**, o **valor anterior** presente na variável é **eliminado**, ficando nela o **novo valor** que lhe foi atribuído;
- A atribuição de valores em C é realizado por meio do sinal de **=**, sendo a variável a alterar **SEMPRE** colocada no **lado esquerdo** da atribuição, e o **valor a atribuir** no **lado direito**.

Exemplo 1

```
int num;           /* Declaração da variável num */  
num = -17;         /* num passa a ter o valor -17 */
```

Exemplo 2

```
int n1=3, n2=5;    /* n1 e n2 são declaradas e ficam com os valores 3 e  
                  5, respectivamente */
```

Exemplo 3

```
int val=0, num=8;  /* val e num são declaradas e ficam com os valores 0  
                  e 8, respectivamente */  
val = num          /* val recebe o valor que esta em num */
```

Imprimindo o Conteúdo das Variáveis

Imprimindo o Conteúdo: `printf`

➤ A forma geral da função `printf` é:

```
printf("tipos de saída", lista de variáveis);
```

- **Tipos de saída:** conjunto de caracteres que especifica o **formato dos dados** a serem escritos e/ou o **texto** a ser escrito;
- **Lista de variáveis:** conjunto de nomes de variáveis, separadas por vírgula, que serão escritos.

Imprimindo o Conteúdo: Função `printf`

```
printf("tipos de saída", lista de variáveis);
```

- Cada **tipo de saída** é precedido pelo sinal **%**;
- Um **tipo de saída** deve ser **especificado** para **cada variável** a ser escrita;
- Se quiser escrever uma **única expressão** com o comando `printf()`:



```
printf("%tipo_de_saída", expressão);
```

- Se for escrever **duas expressões**:



```
printf("%tipo1 %tipo2", expressão1, expressão2);
```

Tipos de Saída: Função `printf`

```
printf("tipos de saída", lista de variáveis);
```

Tipos de Saída	
%c	Escrita de um caractere (char)
%d ou %i	Escrita de números inteiros (int ou char)
%f	Escrita de números reais (float ou double)
%s	Escrita de vários caracteres
%p	Escrita de um endereço de memória
%e ou %E	Escrita em notação científica

Ex_01.c X

```
1  #include <stdlib.h>
2  void main() {
3      int x = 10;
4      printf("%d \n", x);
5
6      float y = 5.0;
7      printf("%d %f \n", x, y);
8  }
9
```

Exemplo 01

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\bin

```
10
10 5.000000
```

```
Process returned 13 (0xD)   execution time : 0.859 s
Press any key to continue.
```


Exemplo 02

Ex_02.c X

```
1  #include <stdio.h>
2
3  void main() {
4      char x;
5      x='A';
6
7      printf("Caractere atribuido = %c \n", x);
8      printf("%c foi o caractere atribuido \n", x);
9  }
```

 D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\bi

```
Caractere atribuido = A
A foi o caractere atribuido
```

```
Process returned 29 (0x1D)    execution time : 0.014 s
Press any key to continue.
```

Lendo Variáveis pelo Teclado

Lendo Variáveis pelo Teclado: Função scanf

- Função `scanf` lê do **teclado** um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado:

```
scanf("tipos de entrada", lista de variáveis);
```

- **Tipos de entrada:** conjunto de caracteres que especifica o formato dos dados a serem lidos;
- **Lista de variáveis:** conjunto de nomes de variáveis que serão lidos e separados por vírgula, em que cada nome de variável é precedido pelo operador `&`.

Lendo Variáveis pelo Teclado: Função scanf

```
scanf("tipos de entrada", lista de variáveis);
```

➤ Cada tipo de entrada é precedido por % e especificado para cada variável a ser lida;

➤ Para ler uma **única variável**:



```
scanf("%tipo_de_entrada", &variável);
```

➤ Para ler **duas variáveis**:



```
scanf("%tipo1 %tipo2", &var1, &var2);
```

Lendo Variáveis pelo Teclado: Função scanf



```
scanf( "%tipo_de_entrada", &variável);
```


➤ Com relação ao **&**:

➤ É uma **exigência** da Linguagem C;

➤ Todas as variáveis que receberão **valores do teclado** por meio do `scanf ()` deverão ser **passadas** pelos seus **ENDEREÇOS**.

Tipos de Entrada	
%c	Escrita de um caractere (char)
%d ou %i	Escrita de números inteiros (int ou char)
%f	Escrita de números reais (float ou double)
%s	Escrita de vários caracteres

```
1  #include <stdio.h>
2
3  void main(){
4      int x,y;
5      float z;
6
7      printf("Digite dois numeros do tipo int: \n");
8      scanf("%d %d", &x, &y);
9
10     printf("Digite um numero do tipo float: \n");
11     scanf("%f", &z);
12
13     printf("Os numeros do tipo int digitados foram: %d e %d. \n", x, y);
14     printf("%f foi o numero digitado do tipo float. \n", z);
15 }
```

 D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\bin\l

Digite dois numeros do tipo int:

2

5

Digite um numero do tipo float:

8.4

Os numeros do tipo int digitados foram: 2 e 5.

8.400000 foi o numero digitado do tipo float.

Process returned 47 (0x2F) execution time : 10.972 s

Press any key to continue.

Operadores Aritméticos

Operadores Aritméticos

- A Linguagem C possui um total de cinco operadores aritméticos:
 - Soma: +
 - Subtração: -
 - Multiplicação: *
 - Divisão: /
 - Resto de uma divisão: %

Operadores Relacionais

Operadores Relacionais

- São operadores de **comparação de valores**;
- A Linguagem C possui um total de **seis operadores** relacionados:

Operador	Significado	Exemplo
>	Maior do que	x>5
>=	Maior ou igual a	x>=10
<	Menor do que	x<5
<=	Menor ou igual a	x<=10
==	Igual a	x==0
!=	Diferente de	x!=0

- Como **resultado**, esse tipo de operador retorna:
 - O valor **UM** (1), se a **expressão relacional** for considerada **VERDADEIRA**;
 - O valor **ZERO** (0), se a **expressão relacional** for considerada **FALSA**.

Operadores Lógicos

Operadores Lógicos

- São usados para saber se determinada **variável** está dentro de uma **faixa de valores**.
Por exemplo:

$$0 < x < 10$$

- Para modelar esse tipo de situação, a Linguagem C possui um conjunto de **três operadores lógicos**:

Operador	Significado	Exemplo
&&	Operador E	$(x > 0 \ \&\& \ x < 10)$
	Operador OU	$(a == 'F' \ \ b != 32)$
!	Operador NEGAÇÃO	$!(x == 10)$

- Os **operadores lógicos** atuam sobre valores lógicos e retornam um valor lógico:
 - **1**: se a expressão for **VERDADEIRA**;
 - **0**: se a expressão for **FALSA**.

```
1  #include <stdio.h>
2
3  void main() {
4      int r, x = 5, y = 3;
5
6      r = (x > 2) && (y < x);
7      printf ("Resultado 1: %d\n", r);
8
9      r = (x % 2==0) && (y > 0);
10     printf ("Resultado 2: %d\n", r);
11
12     r = (x > 2) || (y > x);
13     printf ("Resultado 3: %d\n", r);
14
15     r = (x % 2==0) || (y < 0);
16     printf ("Resultado 4: %d\n", r);
17
18     r = !(x > 2);
19     printf ("Resultado 5: %d\n", r);
20
21     r = !(x > 7) && (x > y);
22     printf ("Resultado 6: %d\n", r);
23 }
24
```

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\l

```
Resultado 1: 1
Resultado 2: 0
Resultado 3: 1
Resultado 4: 0
Resultado 5: 0
Resultado 6: 1
```

```
Process returned 15 (0xF)   execution time : 0.418 s
Press any key to continue.
```

Operadores de Atribuição Simplificada

Operadores de Atribuição Simplificada

➤ A Linguagem C permite **simplificar** algumas **expressões**:

Operador	Significado	Exemplo		
<code>+=</code>	Soma e atribui	<code>x += y</code>	Igual	<code>x = x + y</code>
<code>-=</code>	Subtrai e atribui	<code>x -= y</code>	Igual	<code>x = x - y</code>
<code>*=</code>	Multiplica e atribui	<code>x *= y</code>	Igual	<code>x = x * y</code>
<code>/=</code>	Divide e atribui quociente	<code>x /= y</code>	Igual	<code>x = x / y</code>
<code>%=</code>	Divide e atribui resto	<code>x %= y</code>	Igual	<code>x = x % y</code>

Operadores de Pré e Pós-Incremento/Decremento

Operadores de Pré e Pós-Incremento

➤ Operador de **Incremento**:

➤ Função: Incrementar de **1** o operando;

➤ Trabalha de dois modos:

➤ **Pré-fixado** ➔ ++num

➤ A variável **num** é **incrementada ANTES** de seu valor ser usado.

➤ **Pós-fixado** ➔ num++

➤ A variável **num** é **incrementada DEPOIS** de seu valor ser usado.

Pré-fixado

```
num = 5;
x = ++num; { num=num+1;
              x=num; }
```

x=6 num=6

Pós-fixado

```
num = 5;
x = num++; { x=num;
              num=num+1; }
```

x=5 num=6

Operadores de Pré e Pós-Decremento

- Operador de **Decremento**:
 - Função: Decrementar de 1 o operando;
 - Trabalha de dois modos:
 - **Pré-fixado** ➔ `--num`
 - A variável **num** é **decrementada ANTES** de seu valor ser usado.
 - **Pós-fixado** ➔ `num--`
 - A variável **num** é **decrementada DEPOIS** de seu valor ser usado.

Pré-fixado

```
num = 5;
x = --num; { num=num-1;
              x=num; }
```

x=4 num=4


Pós-fixado

```
num = 5;
x = num--; { x=num;
              num=num-1; }
```

x=5 num=4

Ex_05.c X

```
1  #include <stdio.h>
2
3  void main() {
4      int x = 10, y = 0;
5
6      y = ++x;
7
8      printf("y = %d\n", y);
9      printf("x = %d\n", x);
10 }
```

Equivalente

```
x = x + 1;
y = x;
```

Exemplo: Operador de Pré-Incremento

 D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\t

```
y = 11
x = 11
```

```
Process returned 7 (0x7)   execution time : 0.407 s
Press any key to continue.
```

Exemplo: Operador de Pós-Incremento

Ex_06.c X

```
1  #include <stdio.h>
2
3  void main() {
4      int x = 10, y = 0;
5
6      y = x++;
7
8      printf("y = %d\n", y);
9      printf("x = %d\n", x);
10 }
```

Equivalente

y = x;
x = x + 1;

D:\UFABC\Disciplinas\2021-2025\Q1\PE\Aulas\02\Codigos\Aula02\b

y = 10
x = 11

Process returned 7 (0x7) execution time : 0.953 s
Press any key to continue.

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.