

Linguagem de Programação II

Prof. Mario Bessa

Aula 2

<http://mariobessa.info>

Identificadores

- Os identificadores servem para nomear **palavras-chave, variáveis, constantes, tipos, funções** etc.
- Um bom estilo de programação exige uma seleção adequada de identificadores de tal forma que sejam significativos no contexto do programa.
- Sequência de letras, dígitos e caracter sublinhado (_), podendo começar por letra ou sublinhado.
- As letras maiúsculas e minúsculas são consideradas ***distintas*** em um identificador.

Identificadores

- a .. z (qualquer letra de a to z)
- A .. Z (qualquer letra de A to Z)
- 0 .. 9 (qualquer dígito de 0 to 9)
- _ (caracter underscore)

Identificadores

- Corretos

- n
- *Area_do_Triangulo*
- *XM23*
- *Fim, fim*
- *A, a*

- Incorretos


- ~~• *8Alfa* : começa por dígito~~
- ~~• *Alfa+T* : presença de um caractere não permitido(+)~~
- ~~• char: palavra reservada~~
- ~~• first name: espaço entre palavras~~

Variáveis

- Em C, o acesso a memória principal é feito através do uso de **variáveis**.
- Uma **variável** é um espaço da memória principal reservado para armazenar dados tendo um nome para referenciar o seu conteúdo.
- O valor armazenado em uma variável pode ser modificado ao longo do tempo.
- Cada programa estabelece o número de variáveis que serão utilizadas.

Variáveis

- Exemplo:
 - Nome: `dia`
 - Tipo: `int`
 - Endereço: `0022FF74` (hexadecimal) ou
`2293620` (decimal) ou
`&dia` (representação simbólica)
 - Conteúdo: `27`

`int dia`
`0022FF74` 

Variáveis

- Variáveis possuem:

- **Nome:**

- Identificador usado para acessar o conteúdo.
 - Formado por caracteres alfanuméricos ou pelo caractere de sublinhar, mas não pode iniciar com números.
 - Não pode ter o mesmo nome de uma palavra-chave de C, e nem igual ao nome de uma função declarada pelo programador ou pelas bibliotecas C.
 - Em C letras minúsculas e maiúsculas são diferentes.
 - Habitue-se em declarar variáveis com letras minúsculas.

- **Tipo:**

- Determina a capacidade de armazenamento.
 - Determina a forma como o conteúdo é interpretado.
 - **Ex:** Número real ou inteiro.

- **Endereço:**

- Posição na memória principal.

Tipos de dados

- A linguagem C possui 5 tipos de dados:
 - `char` (Caracter)
 - Representa caracteres ou valores numéricos.
 - Tamanho da representação é de 1 byte ou 8 bits de informação.
 - `int` (Inteiro)
 - Representa 2 ou 4 bytes.
 - `float` e `double` - Ponto flutuante (reais)
 - `double` possui o dobro da precisão do tipo `float`.
 - `void` é um tipo especial e serve para indicar que o tipo é vazio.
- Existem quatro tipos de modificadores para os tipos:
 - `signed`, `unsigned`, `long` e `short`.

Tipos Primitivos de C

Tipo Genérico	Tipo em C	Tamanho	Intervalo de Valores
Tipos inteiros	<i>[signed]_{opc} char</i>	8 bits (byte)	-128...+127
	unsigned char	8 bits (byte)	0...+255
	int	32 bits (word)	-2147483648...+2147483647
	signed [int]_{opc}		
	unsigned [int]_{opc}	32 bits (word)	0...+4294967295
	<i>[signed]_{opc} short [int]_{opc}</i>	16 bits	-32768...+32767
	unsigned short [int]_{opc}	16 bits	0...+65535
	<i>[signed]_{opc} long [int]_{opc}</i>	32 bits	-2147483648...+2147483647
	unsigned long [int]_{opc}	32 bits	0...+4294967295
	enum...{...} (enumeração)	32 bits (word)	-2147483648...+2147483647
Tipos reais (ponto flutuante)	float	32 bits	$\approx \pm 10.0^{-40} \dots \pm 10.0^{+40}$
	double	64 bits	$\approx \pm 10.0^{-310} \dots \pm 10.0^{+310}$
	long double	96 bits	$\approx \pm 10.0^{-4934} \dots \pm 10.0^{+4934}$
Tipo vazio (sem valor)	void	8 bits	N/A

Tamanho varia por Compilador

- Tamanho de `integer` e `float` varia dependendo do compilador.
- ANSI C define a seguinte regra:
 - `short int <= int <= long int`
 - `float <= double <= long double`

Declaração de variáveis

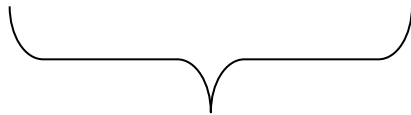
- Sintaxe: `<tipo> <nome> [=valor];`
- Ex:

```
int ano = 1980;
```

```
float salario = 970.0;
```

```
char letra = 'A'; // 'A' é o valor 65.
```

```
int numero, Numero; // C é Case Sensitive.
```



É possível declarar mais de uma variável do mesmo tipo de uma única vez, separando seus nomes por vírgulas.

Exemplos

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

```
int main(){
```

```
    int a;
```

```
    unsigned int b;
```

```
    short c;
```

```
    char g;
```

```
    a = 10;           // Correto.
```

```
    b = -6;           // Errado.
```

```
    c = 100000;       // Errado.
```

```
    g = 'e';          // Correto.
```

```
    g = e;            // Errado.
```

```
    return 0;
```

```
}
```

Casting

- Casting é a maneira de converter uma variável de um tipo de dado para um outro tipo de dado.
- Sintaxe: (tipo) <expressão>;
- Ex:

```
float f;
```

```
int i=10;
```

```
f = (float) i; // assinalado 10.0 a f
```

```
f = 3.14;
```

```
i = (int) f; // assinalado 3 a i
```

```
}
```

Casting

```
#include <stdio.h>
main()
{
    int sum = 17, count = 5;
    double mean;
    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean );
}
```

Exemplos

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

```
int main(){
```

```
    int e,f;
```

```
    int a = 10, b = -30;
```

```
    float c;
```

```
    char d = '4'; // '4' é o valor 52.
```

```
    c = a; // converte para float e copia 10.0 para "c".
```

```
    c = a + 1.8; // atribui valor 11.8 para "c".
```

```
    b = c; // converte para int truncando e copia 11 para "b".
```

```
    b = a + b; // soma 10 e 11, e copia 21 para "b".
```

```
    a = a + d; // soma 10 e 52, e copia 62 para "a".
```

```
    e = 0.2 + c; // soma 0.2 e 11.8 e copia 12 para "e".
```

```
    f = 0.2 + (int)c; /* converte "c" para 11 antes, soma 0.2 e trunca novamente  
                      para 11 e copia 11 para "f" */
```

```
    printf("a=%i\n",a);
```

```
    printf("b=%i\n",b);
```

```
    printf("c=%f\n",c);
```

```
    printf("d=%c\n",d);
```

```
    printf("f=%i\n",f);
```

```
    return 0;
```

```
}
```

Constantes

- Valor armazenado em um endereço de memória que possui um tipo de dado e seu valor **é fixo e inalterável** durante a execução do programa.
- Implementado com **#define** ou através da palavra reservada **const**.
- Sintaxe: `#define <NOME> <valor> //não termina com ;`
`const <tipo> <NOME> = valor;`
- Exemplo:

```
const float PI = 3.1415926;  
#define PI 3.1415926 //não termina com ;  
#define TRUE 1  
#define FALSE 0
```


Exemplo

```
#include <stdio.h>
int main(){
    const float PI = 3.1415926;
    PI=PI+1; //erro
    printf("PI+1=%f\n",PI+1);
    return 0;
}
```

Operadores Aritméticos

- C oferece 6 operadores aritméticos binários (**operam sobre dois operandos**) e um operador aritmético unário (**opera sobre um operando**).

Binários	
=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)
Unário	
-	Menos unário

Precedência	Operador
1	- unário
2	* / %
3	+ -

O uso de parênteses altera a ordem de prioridade das operações.

Ex:

$$(a + b) * 80 \neq a + b * 80$$

Mais Operadores

- C tem vários operadores que permitem comprimir comandos.

variável **op=** expressão; \longrightarrow **variável** = (**variável**) **op** (expressão);

Operador	Exemplos		
++	i++;	equivale a	i = i + 1;
--	i--;	equivale a	i = i - 1;
+=	i += 2;	equivale a	i = i + 2;
-=	d -= 3;	equivale a	d = d - 3;
*=	x *= y+1;	equivale a	x = x*(y+1);
/=	t /= 2.5;	equivale a	t = t/2.5;
%=	p %= 5;	equivale a	p = p%5;

Operadores Aritméticos

- Os operadores ++ e -- podem ser pré-fixados e pós-fixados e existe diferença entre eles.

`a = ++b;` equivale a `b = b + 1; a = b;`

`a = b++;` equivale a `a = b; b = b + 1;`

- Exemplo: `x=23; y=x++;` e `x=23; y=++x;`
 - No primeiro teremos y igual a 23 e x igual a 24.
 - No segundo teremos x igual a 24 e y igual a 24.

Operadores Relacionais

- São usados para fazer comparações. Retornam zero (0) ou um (1) dependendo da expressão ser **falsa** ou **verdadeira** respectivamente.
- Possuem menor **precedência** que a dos operadores aritméticos.

Operador	Função
>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
==	igualdade
!=	diferente

Operadores Relacionais

- **Exemplo:**

```
int main(){  
    int teste1, teste2;  
    teste1 = (10 < 30);  
    teste2 = (20 == 25);  
    printf("Teste1= %d, teste2= %d\n", teste1, teste2);  
    return 0;  
}
```

- **Saída:**

Teste1= 1, teste2= 0

Operadores Lógicos

- C possui 3 operadores chamados lógicos:

Operador	Função
&&	lógico E
 	lógico OU
!	lógico de negação

Exemplos	
exp1 && exp2	É verdadeira se as duas exp1 e exp2 forem verdadeiras.
exp1 exp2	É verdadeira se uma das duas exp1 ou exp2 for verdadeira ou se as duas forem verdadeiras.
!exp1	É verdadeira se exp1 for falsa.

Operadores Lógicos

- **Exemplo:**

```
int main(){  
    int teste;  
    float x,y;  
    scanf("%f %f",&x,&y);  
    teste = ((x>y && x<5.0)|| y<2.0) ;  
    printf("Teste: %d\n",teste);  
    return 0;  
}
```


A função printf()

- A função **printf()** escreve o texto passado no interior dos parênteses na saída padrão (terminal/monitor).

`printf("format-string", expression, ...);`

- *format-string* pode conter caracteres regulares, que são impressos diretamente, e especificações de formato.
- As especificações de formato usam os caracteres de controle \ e %.
- Para cada caracter de controle %, deve haver uma expressão correspondente.

`printf("%i + %i = %i\n", 2, 3, (2+3)); // 2 + 3 = 5`

Especificações de formato

Formato	Descrição
<code>\b</code>	retorno de um caracter
<code>\f</code>	form feed FF (also clear screen)
<code>\n</code>	nova linha
<code>\r</code>	retorno de carro (cursor no início da linha)
<code>\t</code>	tabulação horizontal
<code>\v</code>	tabulação vertical
<code>\”</code>	aspas duplas
<code>\’</code>	aspa simples ’
<code>\\</code>	contrabarra \
<code>\?</code>	interrogação

Especificações de formato para números inteiros

Formato	Descrição
%d, %i	Exibe valores como um inteiro (base decimal)
%ld ou %li	Exibe valores como um inteiro longo (base decimal)
%hd ou %hi	Exibe valores como um inteiro curto (base decimal)
%u	Exibe valores como um inteiro sem sinal (base decimal)
%o	Exibe valores como um inteiro sem sinal (base octal)
%x ou %X	Exibe valores como um inteiro sem sinal (base hexadecimal)

Formato	Tipo	Exemplo	Saída
%d, %i	int	printf("%i", 17);	17
%ld ou %li	long int	printf("%ld", 2000000000L);	2000000000
%hd ou %hi	short int	printf("%hd", 3200);	3200
%u	unsigned int	printf("%u", 17u);	17
%o	unsigned int	printf("%o", 17);	21
%x ou %X	unsigned int	printf("%x", 26); printf("%X", 26);	1a 1A

Especificações de formato para números decimais

Formato	Descrição
%f ou %F	Exibe valores de ponto flutuante em notação fixa. Sempre imprime pelo menos um dígito à esquerda do ponto decimal. Mostra 6 dígitos de precisão à direita do ponto decimal.
%e ou %E	Exibe um valor de ponto flutuante em notação exponencial.
%g ou %G	%e ou %f. O que for mais curto para imprimir. Sem zeros no final.
L	usado antes de qualquer especificação de ponto flutuante para indicar que um valor de ponto flutuante <i>long double</i> será exibido.

Formato	Exemplo	Saída
%f ou %F	printf("%f", 3.14); printf(" %5.2f \n", 3.147);	3.140000 3.15
%e ou %E	printf("%e", 31.4); printf("%E", 31.4);	3.140000e+01 3.140000E+01
%g ou %G	printf("%g, %g", 3.14, 0.0000314);	3.14, 3.14e-05
.precisão	Número de dígitos à direita do ponto decimal (A,e,E,f,F,g,G)	printf(" %5.2f ", 3.147); printf(" %5.2G ", 3.147);
	Precisão. Ex: %10.4f, exibe um número com no mínimo 10 caracteres e com 4 casas decimais.	

Exemplo de Especificações de formato

```
#include <stdio.h>
int main(){
    printf("%e\n",1234567.89);
    printf("%e\n",+1234567.89);
    printf("%e\n",-1234567.89);
    printf("%E\n",1234567.89);
    printf("%f\n",1234567.89);
    printf("%g\n",1234567.89);
    printf("%G\n",1234567.89);
    printf("%g\n",0.0000875);
    printf("%e\n",8750000.0);
    printf("%g\n",8750000.0);
    printf("%g\n",8.75);
    printf("%e\n",8.75);
    printf("%g\n",87.50);
    printf("%e\n",87.50);
    printf("%g\n",875.);
    printf("%e\n",875.);
    return 0;
}
```

Saída:

```
1.234568e+06
1.234568e+06
-1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457E+06
8.75e-05
8.750000e+06
8.75e+06
8.75
8.750000e+00
87.5
8.750000e+01
875
8.750000e+02
```

Especificações de formato para caracter e string

Formato	Descrição
%c	Exibe apenas um caracter
%s	Exibe um texto

Formato	Tipo	Exemplo	Saída
%c	Int ou char	<pre>printf("%c", 65); printf("%c", 'A'); printf("%i", 'A');</pre>	<pre>A A 65</pre>
%s	string	<pre>printf("%s", "Hello"); printf(" %7.3s ", "ABCDEF"); printf(" %3.7s ", "ABCDEF"); printf(" %7.8s ", "ABCDEFGH");</pre>	<pre>Hello ABC ABCDEF ABCDEFGH </pre>
	Largura mínima. Ex: %5.7s, exibe uma string que tem pelo menos 5 caracteres de comprimento e não passa de 7.		

Códigos de formatação: printf()

Formato	Descrição	Exemplo	Saída
flags	- justificado à esquerda	<code>printf(" %3i %-3i ", 12, 12);</code>	<code> ·12 12· </code>
	+ inclui + à esquerda de números positivos	<code>printf("%+i", 17);</code>	<code>+17</code>
	<i>espaço</i> inclui espaços à esquerda de números positivos	<code>printf(" % i ", 12);</code>	<code> ·12 </code>
	# força a saída em forma alternativa: o (octal), x(hex)	<code>printf("%#X", 26);</code>	<code>0X1A</code>
	0 inclui zeros à esquerda	<code>printf(" %04i ", 12);</code>	<code> 0012 </code>

Códigos de formatação: printf()

<i>Entrada</i>	<i>Caracter Contrôlê</i>	<i>Saída</i>
42	%6d	42
42	%-6d	42
324	%10d	324
-1	%-10d	-1
-1	%1d	-1
'z'	%3c	z
'z'	%-3c	z
2.71828	%10f	2.71828
2.71828	%10.2f	2.72
2.71828	%-10.2f	2.72
2.71828	%2.4f	2.7183
2.718	%.4	2.7180
2.718	%10.5f	2.71800

A função scanf()

- A função **scanf()** lê a informação da entrada padrão (terminal/monitor).

`scanf("format-string",&var1...&varN);`

– **Ex:** `scanf("%f",&variavel);`

- Para cada caracter de controle %, deve haver uma variável correspondente.
- Necessidade do operador & para as variáveis que não sejam ponteiros.

Exemplo de formatação

- **Problema: Sucessor e antecessor**
 - Leia um número e imprima o seu sucessor e seu antecessor

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

```
int main(){  
    int  numero;
```

```
    printf("Entre um número: ");
```

```
    scanf("%i",&numero);
```

```
    printf("Sucessor: %i\nAntecessor: %i",numero+1,numero-1);
```

```
    return 0;
```

```
}
```

Exemplo de formatação

- **Problema: Aumento de salário**

- Leia o valor de salário e calcule: um aumento de 30% e um desconto de 25%

```
#include <stdio.h>
int main(){
    float  salario, aumento=0.30, desconto=0.20;
    printf("Entre com o valor para salario: ");
    scanf("%f",&salario);
    printf("Salario com aumento: R$%.2f\nSalario com desconto: R$%.2f",
    salario*(1+aumento),salario*(1-desconto));

    return 0;
}
```