




LINGUAGEM C: VARIÁVEIS E EXPRESSÕES

Prof. André Backes

LINGUAGENS DE PROGRAMAÇÃO

- Linguagem de Máquina
 - Computador entende apenas pulsos elétricos
 - Presença ou não de pulso
 - 1 ou 0
 - Tudo no computador deve ser descrito em termos de 1's ou 0's (binário)
 - Difícil para humanos ler ou escrever
 - $00011110 = 30$
- 

LINGUAGENS DE PROGRAMAÇÃO

- Linguagem *Assembly*
 - Uso de mnemônicos
 - Conjunto de 0's e 1's é agora representado por um código
 - 10011011 -> ADD
- Linguagem *Assembly* - Problemas
 - Requer programação especial (*Assembly*)
 - Conjunto de instruções varia com o computador (processador)
 - Ainda é muito difícil programar



LINGUAGENS DE PROGRAMAÇÃO

- Linguagens de Alto Nível
 - Programas são escritos utilizando uma linguagem parecida com a linguagem humana
 - Independente da arquitetura do computador
 - Mais fácil programar
 - Uso de compiladores



LINGUAGENS DE PROGRAMAÇÃO

○ Primórdios

- Uso da computação para cálculos de fórmulas
- Fórmulas eram traduzidas para linguagem de máquinas
- Por que não escrever programas parecidos com as fórmulas que se deseja computar?



LINGUAGENS DE PROGRAMAÇÃO

○ FORTRAN (FORMula TRANSform)

- Em 1950, um grupo de programadores da IBM liderados por John Backus produz a versão inicial da linguagem;
- Primeira linguagem de alto nível;

○ Várias outras linguagens de alto nível foram criadas

- Algol-60, Cobol, Pascal, etc



LINGUAGEM C

- Uma das mais bem sucedidas foi uma linguagem chamada C
 - Criada em 1972 nos laboratórios por Dennis Ritchie
 - Revisada e padronizada pela ANSI em 1989
 - ANSI: American National Standards Institute
 - Mais utilizada
 - Novas revisões (dependem do compilador)
 - C99, padrão ISO/IEC 9899:1999
 - C11, padrão ISO/IEC 9899:2011 (mais atual)

PRIMEIRO PROGRAMA EM C

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

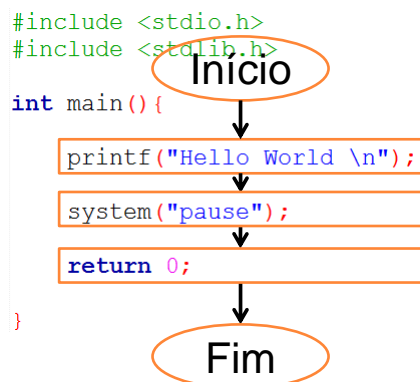
int main() {

    printf("Hello World \n");

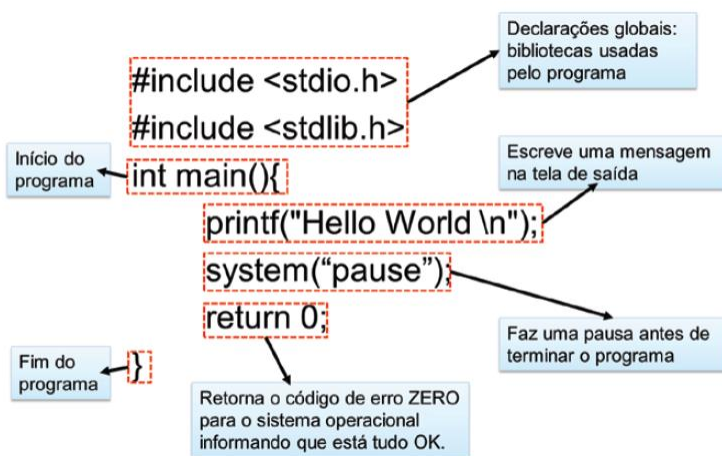
    system("pause");

    return 0;
}
```

PRIMEIRO PROGRAMA EM C



PRIMEIRO PROGRAMA EM C



PRIMEIRO PROGRAMA EM C

- Por que escrevemos programas?
 - Temos dados ou informações que precisam ser processados;
 - Esse processamento pode ser algum cálculo ou pesquisa sobre os dados de entrada;
 - Desse processamento, esperamos obter alguns resultados (Saídas);

COMENTÁRIOS

- Permitem adicionar uma descrição sobre o programa. São ignorados pelo compilador.

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

int main() {
    /*
    A função printf serve
    para escrever na tela
    */
    printf("Hello World \n");

    //faz uma pausa no programa
    system("pause");

    return 0;
}
```

VARIÁVEIS

○ Matemática

- é uma entidade capaz de representar um valor ou expressão;
- pode representar um número ou um conjunto de números
- $f(x) = x^2$



VARIÁVEIS

○ Computação

- Posição de memória que armazena uma informação
- Pode ser modificada pelo programa
- Deve ser **definida** antes de ser usada



DECLARAÇÃO DE VARIÁVEIS

- Precisamos informar ao programa quais dados queremos armazenar
- Precisamos também informar o que são esses dados (qual o tipo de dado)
 - Um nome de uma pessoa
 - Uma cadeia de caracteres ("André" - 5 caracteres)
 - O valor da temperatura atual
 - Um valor numérico (com casas decimais)
 - A quantidade de alunos em uma sala de aula
 - Um valor numérico (número inteiro positivo ou zero)
 - Se um assento de uma aeronave está ocupado
 - Um valor lógico (ocupado: verdadeiro / desocupado: falso)

VARIÁVEIS

- Declaração de variáveis em C
 - <tipo de dado> nome-da-variável;
- Propriedades
 - Nome
 - Pode ter um ou mais caracteres
 - Nem tudo pode ser usado como nome
 - Tipo
 - Conjunto de valores aceitos
 - Escopo
 - global ou local

VARIÁVEIS

○ Nome

- Deve iniciar com letras ou underscore (_);
- Caracteres devem ser letras, números ou underscores;
- Palavras chaves não podem ser usadas como nomes;
- Letras maiúsculas e minúsculas são consideradas diferentes

VARIÁVEIS

○ Nome

- Não utilizar espaços nos nomes
 - Exemplo: nome do aluno, temperatura do sensor,
- Não utilizar acentos ou símbolos
 - Exemplos: garça, tripé, o,Θ
- Não inicializar o nome da variável com números
 - Exemplos: 1A, 52, 5ª
- Underscore pode ser usado
 - Exemplo: nome_do_aluno : caracter
- Não pode haver duas variáveis com o mesmo nome

VARIÁVEIS

○ Lista de palavras chaves (ANSI-C)

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

VARIÁVEIS

○ Lista de palavras chaves (C11)

auto	break	case	char	const	continue	do
else	for	int	union	static	default	void
enum	goto	long	unsigned	struct	extern	while
float	if	typedef	volatile	switch	register	typeof
double	return	sizeof	signed	short	_Atomic	_Imaginary
_Generic	inline	_Static_assert	_Alignas	_Bool	_Alignof	_Complex
_Noreturn	restrict	_Thread_local				

VARIÁVEIS

- Quais nomes de variáveis estão corretos:
 - Contador
 - contador1
 - comp!
 - .var
 - Teste_123
 - _teste
 - int
 - int1
 - 1contador
 - -x
 - Teste-123
 - x&

VARIÁVEIS

- Corretos:
 - Contador, contador1, Teste_123, _teste, int1
- Errados
 - comp!, .var, int, 1contador, -x, Teste-123, x&

VARIÁVEIS

- Tipo
 - Define os valores que ela pode assumir e as operações que podem ser realizadas com ela
- Exemplo
 - tipo **int** recebe apenas valores inteiros
 - tipo **float** armazena apenas valores reais

TIPOS BÁSICOS EM C

- **char**: um byte que armazena o código de um caractere do conjunto de caracteres local
 - **caracteres sempre ficam entre 'aspas simples'!**
- **int**: um inteiro cujo tamanho depende do processador, tipicamente 16 ou 32 bits

```
char sexo; // pode receber 'M' ou 'F'
char UnidadeTemperatura; //pode receber 'C' para Celsius
                        //ou 'F' para Fahrenheit
char opcoes; // pode ser '1', '2' , '3' ou '4'
```

```
int NumeroAlunos;
int Idade;
int NumeroContaCorrente;
int N = 10; // o variável N recebe o valor 10
```

TIPOS BÁSICOS EM C

○ Números reais

- Tipos: *float*, *double* e *long double*
- A parte decimal usa **ponto** e **não vírgula!**
- **float**: um número real com precisão simples

```
float Temperatura; // por exemplo, 23.30
float MediaNotas; // por exemplo, 7.98
float TempoTotal; // por exemplo, 0.0000000032 (s)
```

- **double**: um número real com precisão dupla
 - Números muito grandes ou muito pequenos

```
double DistanciaGalaxias; // número muito grande
double MassaMolecular; // em Kg, número muito pequeno
double BalancoEmpresa; // valores financeiros
```

TIPOS BÁSICOS EM C

○ Números reais

- Pode-se escrever números reais usando notação científica

```
double TempoTotal = 0.00000003295;
```

```
//notação científica
```

```
double TempoTotal = 3.295e-9; equivale à 3,295x10-9
```

MODIFICADORES DE TIPOS

- São aplicados antes dos tipos básicos (com a exceção do tipo **void**), e permitem alterar o significado do tipo:

- **signed**: determina que uma variável declarada dos tipos **char** ou **int** poderá ter valores positivos ou negativos

```
signed char x;
signed int y;
```

- **unsigned**: determina que uma variável declarada dos tipos **char** ou **int** somente poderá ter valores positivos e o valor zero

```
unsigned char x;
unsigned int y;
```

MODIFICADORES DE TIPOS

- São aplicados antes dos tipos básicos (com a exceção do tipo **void**), e permitem alterar o significado do tipo:

- **short**: determina que uma variável do tipo **int** terá apenas 16 bits (*inteiro pequeno*)

```
short int i;
```

- **long**: determina que uma variável do tipo **int** terá 32 bits (*inteiro grande*) e que o tipo **double** possua maior precisão

```
long int n;
long double d;
```

TAMANHO DOS TIPOS

Tipo	Bits	Intervalo de valores
char	8	-128 A 127
unsigned char	8	0 A 255
signed char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
unsigned int	32	0 A 4.294.967.295
signed int	32	-32.768 A 32.767
short int	16	-32.768 A 32.767
unsigned short int	16	0 A 65.535
signed short int	16	-32.768 A 32.767
long int	32	-2.147.483.648 A 2.147.483.647
unsigned long int	32	0 A 4.294.967.295
signed long 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
long double	96	3,4E-4932 A 3,4E+4932

ATRIBUIÇÃO

Operador de Atribuição: =

- nome_da_variável = expressão, valor ou constante;



O operador de atribuição "=" armazena o valor ou resultado de uma expressão contida à sua **direita** na variável especificada à sua **esquerda**.

Ex.:

```
int main( ){
    int x = 5; // x recebe 5
    int y;
    y = x + 3; // y recebe x mais 3


    return 0;
}
```

- A linguagem C suporta múltiplas atribuições
 - x = y = z = 0;

COMANDO DE SAÍDA

o printf()

- *print formatted*
- Comando que realiza a impressão dos dados do programa na tela

 ← `printf("Texto");`

- O texto a ser escrito deve ser sempre definido entre “aspas duplas”

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


int main() {
    printf("Esse texto sera escrito na tela");


    return 0;
}
```

COMANDO DE SAÍDA

o printf()

- Quando queremos escrever dados formatados na tela usamos a forma geral da função, a qual possui os tipos de saída.
- Eles especificam o formato de saída dos dados que serão escritos pela função **printf()**.

 ← `printf("%tipo_de_saida", expressão);`

 ← `printf("%tipo1 %tipo2", valor1, valor2);`

COMANDO DE SAÍDA

o printf()

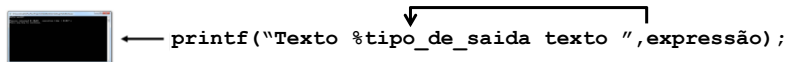
- Especificadores de formato de tipos de saída

Alguns tipos de saída	
%c	Escrita de um caractere (char)
%d ou %i	Escrita de números inteiros (int)
%u	Escrita de números inteiros sem sinal (unsigned)
%f	Escrita de números reais (float ou double)
%s	Escrita de vários caracteres (string)
%p	Escrita de um endereço de memória

COMANDO DE SAÍDA

o printf()

- Podemos também misturar o texto a ser mostrado com os especificadores de formato.



```
← printf("Texto %tipo_de_saida texto ", expressão);
```

COMANDO DE SAÍDA

o printf()

- Exemplos

```
int main() {
    printf("Esse texto sera escrito na tela");

    int x = 10;
    float y = 20;
    printf("%d", x);

    printf("%d %f", x, y);

    printf("Valor de x eh %d e o de y eh %f", x, y);

    return 0;
}
```

COMANDO DE ENTRADA

o scanf()

- Comando que realiza a leitura dos dados da entrada padrão (no caso o teclado)
- Forma geral:
 - o `scanf("tipo de entrada", lista de variáveis);`



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

COMANDO DE ENTRADA

o **scanf()**

- O tipo de entrada deve ser sempre definido entre “aspas duplas”
- Na linguagem C, é necessário colocar o símbolo & antes do nome de cada variável a ser lida pelo comando **scanf()**.
 - o O símbolo & indica qual é o endereço da variável que vai receber os dados lidos

```
int main() {
    int x;
    scanf("%d", &x);

    printf("Valor de x: %d", x);

    return 0;
}
```

COMANDO DE ENTRADA

o **scanf()**

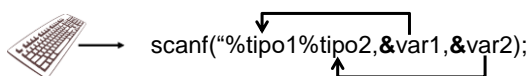
- Especificadores de formato do tipo de entrada

Alguns tipos de entrada	
%c	Leitura de um caractere (char)
%d ou %i	Leitura de números inteiros (int ou char)
%f	Leitura de números reais (float)
%lf	Leitura de números reais (double)
%s	Leitura de vários caracteres (string)

COMANDO DE ENTRADA

o scanf()

- Podemos ler mais de um valor em um único comando
 - Quando digitar vários valores, separar com espaço, TAB, ou Enter



COMANDO DE ENTRADA

o scanf()

```
int main() {
    int x,z;
    float y;
    //Leitura de um valor inteiro
    scanf("%d", &x);
    //Leitura de um valor real
    scanf("%f", &y);
    //Leitura de um valor inteiro e outro real
    scanf("%d%f", &x, &y);
    //Leitura de dois valores inteiros
    scanf("%d%d", &x, &z);
    //Leitura de dois valores inteiros com espaço
    scanf("%d %d", &x, &z);

    return 0;
}
```

COMANDO DE ENTRADA

o **getchar()**

- Comando que realiza a leitura de um único caractere

```
int main() {  
    char c;  
    c = getchar();  
    printf("Caractere: %c\n", c);  
    printf("Codigo ASCII: %d\n", c);  
  
    return 0;  
}
```

ESCOPO DE VARIÁVEIS

o Escopo

- Define onde e quando a variável pode ser usada.

o Escopo global

- Fora de qualquer definição de função
- Tempo de vida é o tempo de execução do programa

o Escopo local

- Bloco ou função

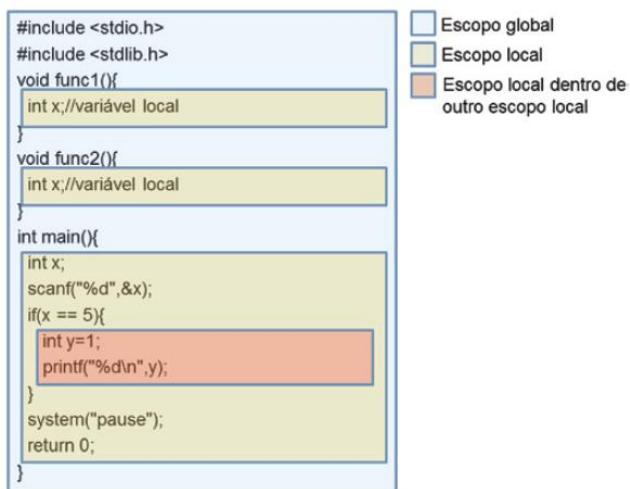
ESCOPO DE VARIÁVEIS

○ Escopo local

- Bloco: visível apenas no interior de um bloco de comandos
- Função: declarada na lista de parâmetros da função ou definida dentro da função

```
//bloco
if(x == 10) {
    int i;
    i = i + 1;
}
//função
int soma (int x, int y) {
    int z;
    z = x + y;
    return z;
}
```

ESCOPO DE VARIÁVEIS



CONSTANTES

- Como uma variável, uma constante também armazena um valor na memória do computador.
- Entretanto, esse valor não pode ser alterado: é constante.
- Para constantes é obrigatória a atribuição do valor.

CONSTANTES

- Usando **#define**
 - Você deverá incluir a diretiva de pré-processador **#define** antes de início do código:
 - Ela informa ao compilador que ele deve procurar todas as ocorrências da palavra definida e substituir pelo valor quando o programa for compilado
 - **Cuidado: não colocar “;”**

```
#define PI 3.1415
```

```
float x = 2 * PI;
```

CONSTANTES

○ Usando **const**

- Usando **const**, a declaração não precisa estar no início do código
- A declaração é igual a de uma variável inicializada
- Basicamente, criamos uma variável que não pode ter seu valor alterado.

```
const double pi = 3.1415;
```



SEQUÊNCIAS DE ESCAPE

- São constantes predefinidas
- Elas permitem o envio de caracteres de controle não gráficos para dispositivos de saída

Código	Comando
\a	som de alerta (bip)
\b	retrocesso (backspace)
\n	nova linha (new line)
\r	retorno de carro (carriage return)
\v	tabulação vertical
\t	tabulação horizontal
\'	apóstrofe
\"	aspa
\\	barra invertida (backslash)
\f	alimentação de folha (form feed)
\?	símbolo de interrogação
\0	caractere nulo (cancela a escrita do restante)

SEQUÊNCIAS DE ESCAPE

○ Exemplo

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

int main() {
    printf("Hello World\n");
    printf("Hello\nWorld\n");
    printf("Hello \\ World\n");
    printf("\\"Hello World\\"n");

    return 0;
}
```

○ Saída

```
Hello World
Hello
World
Hello \ World
"Hello World"
```

TIPOS BOOLEANOS EM C

- Um tipo booleano pode assumir dois valores:
 - verdadeiro ou falso (true ou false)
- Na linguagem ANSI-C não existe o tipo de dado booleano, apenas a partir do padrão C99. Para armazenar esse tipo de informação, usa-se uma variável do tipo **int** (número inteiro)
 - Valor 0 significa falso
 - Números + ou - : verdadeiro
- Exemplos:

```
int AssentoOcupado = 1; // verdadeiro
int PortaAberta = 0; // falso
```

OPERADORES

- Os operadores são usados para desenvolver diferentes tipos de operações. Com eles podemos:
 - Realizar operações matemáticas com suas variáveis.
 - Realizar operações de comparação entre suas variáveis.
 - Realizar operações lógicas entre suas variáveis.
 - Realizar operações em nível de bits com suas variáveis

OPERADORES ARITMÉTICOS

- São aqueles que operam sobre números (valores, variáveis, constantes ou chamadas de funções) e/ou expressões e têm como resultados valores numéricos
 - Note que os operadores aritméticos são sempre usados em conjunto com o operador de atribuição.

Operador	Significado	Exemplo
+	Adição de dois valores	$z = x + y$
-	Subtração de dois valores	$z = x - y$
*	Multiplicação de dois valores	$z = x * y$
/	Quociente de dois valores	$z = x / y$
%	Resto de uma divisão	$z = x \% y$

OPERADORES ARITMÉTICOS

- Podemos devolver o resultado para uma outra variável ou para um outro comando ou função que espere receber um valor do mesmo tipo do resultado da operação, no caso, a função printf()

```
int main() {
    int x = 10, y = 20, z;

    z = x * y;
    printf("z = %d\n", z);

    z = y / 10;
    printf("z = %d\n", z);

    printf("x+y = %d\n", x+y);

    return 0;
}
```

OPERADORES ARITMÉTICOS

◦ IMPORTANTE

- As operações de multiplicação, divisão e resto são executadas antes das operações de adição e subtração. Para forçar uma operação a ser executada antes das demais, ela é colocada entre parênteses
 - $z = x * y + 10;$
 - $z = x * (y + 10);$
- O operador de subtração também pode ser utilizado para inverter o sinal de um número
 - $x = -y;$
- Neste caso, a variável **x** receberá o valor de **y** multiplicado por **-1**, ou seja,
 - $x = (-1) * y;$

OPERADORES ARITMÉTICOS

○ IMPORTANTE

- Em uma operação utilizando o operador de quociente /, se o numerador e o denominador forem números inteiros, por padrão o compilador retornará apenas a parte inteira da divisão

```
int main() {
    float x;
    x = 5/4; // x = 1.000000
    printf("x = %f\n", x);

    x = 5/4.0; // x = 1.250000
    printf("x = %f\n", x);

    return 0;
}
```

OPERADORES RELACIONAIS

- São aqueles que verificam a magnitude (qual é maior ou menor) e/ou igualdade entre dois valores e/ou expressões.
 - Os operadores relacionais são operadores de comparação de valores
 - Retorna **verdadeiro** (1) ou **falso** (0)

Operador	Significado	Exemplo
>	Maior do que	X > 5
>=	Maior ou igual a	X >= Y
<	Menor do que	X < 5
<=	Menor ou igual a	X <= Z
==	Igual a	X == 0
!=	Diferente de	X != Y

IMPORTANTE

- Símbolo de atribuição = é diferente, muito diferente, do operador relacional de igualdade ==

```
int Nota;
Nota == 60; // Nota é igual a 60?
Nota = 50; // Nota recebe 50
// Erro comum em C:
// Teste se a nota é 60
// Sempre entra na condição
if (Nota = 60) {
    printf("Você passou raspando!!");
}
// Versão Correta
if (Nota == 60) {
    printf("Você passou raspando!!");
}
```

IMPORTANTE

- Símbolo de atribuição = é diferente, muito diferente, do operador relacional de igualdade ==
- Por que sempre entra na condição?

```
if (Nota = 60) {
    printf("Você passou raspando!!");
}
```

- Ao fazer **Nota = 60** ("Nota recebe 60") estamos atribuindo um valor inteiro à variável Nota.
- O valor atribuído **60 é diferente de Zero**. Como em C os booleanos são números inteiros, então vendo **Nota** como booleano, essa assume **true**, uma vez que é diferente de zero

OPERADORES LÓGICOS

- Certas situações não podem ser modeladas utilizando apenas os operadores aritméticos e/ou relacionais
 - Um exemplo bastante simples disso é saber se determinada variável x está dentro de uma faixa de valores.
 - Por exemplo, a expressão matemática
 - $0 < x < 10$
 - indica que o valor de x deve ser maior do que 0 (zero) e também menor do que 10

OPERADORES LÓGICOS

- Os operadores lógicos permitem representar situações lógicas unindo duas ou mais expressões relacionais simples em uma composta
 - Retorna **verdadeiro** (1) ou **falso** (0)
- Exemplo
 - A expressão $0 < x < 10$
 - Equivale a $(x > 0) \ \&\& \ (x < 10)$

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

OPERADORES LÓGICOS

○ Tabela verdade

- Os termos **a** e **b** representam o resultado de duas expressões relacionais

a	b	!a	!b	a && b	a b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

OPERADORES LÓGICOS

○ Exemplos

```
int main(){
    int r, x = 5, y = 3;
    r = (x > 2) && (y < x); //verdadeiro (1)
    printf("Resultado: %d\n", r);
    r = (x%2==0) && (y > 0); //falso (0)
    printf("Resultado: %d\n", r);
    r = (x > 2) || (y > x); //verdadeiro (1)
    printf("Resultado: %d\n", r);
    r = (x%2==0) || (y < 0); //falso (0)
    printf("Resultado: %d\n", r);
    r = !(x > 2); // falso (0)
    printf("Resultado: %d\n", r);
    r = !(x > 7) && (x > y); // verdadeiro (1)
    printf("Resultado: %d\n", r);

    return 0;
}
```

OPERADORES DE PRÉ E PÓS-INCREMENTO/DECREMENTO

- Esses operadores podem ser utilizados sempre que for necessário somar uma unidade (incremento) ou subtrair uma unidade (decremento) a determinado valor

Operador	Significado	Exemplo	Resultado
++	incremento	++x ou x++	$x = x + 1$
--	decremento	--x ou x--	$x = x - 1$

OPERADORES DE PRÉ E PÓS-INCREMENTO/DECREMENTO

- Qual a diferença em usar antes ou depois da variável?

Operador	Significado	Resultado
++x	pré-incremento	soma +1 à variável x antes de utilizar seu valor
x++	pós-incremento	soma +1 à variável x depois de utilizar seu valor
--x	pré-decremento	subtrai -1 da variável x antes de utilizar seu valor
x--	pós-decremento	subtrai -1 da variável x depois de utilizar seu valor

- Essa diferença de sintaxe no uso do operador não tem importância se o operador for usado sozinho
 - Porém, se esse operador for utilizado dentro de uma expressão aritmética, a diferença entre os dois operadores será evidente

OPERADORES DE PRÉ E PÓS-INCREMENTO/DECREMENTO

- Essa diferença de sintaxe no uso do operador não tem importância se o operador for usado sozinho
 - Porém, se utilizado dentro de uma expressão aritmética, a diferença entre os dois operadores será evidente

```
int main() {
    int x, y;
    x = 10;
    y = x++;
    printf("%d \n", x); // 11
    printf("%d \n", y); // 10

    y = ++x;
    printf("%d \n", x); // 12
    printf("%d \n", y); // 12

    return 0;
}
```

OPERADORES DE ATRIBUIÇÃO SIMPLIFICADA

- Muitos operadores são sempre usados em conjunto com o operador de atribuição.
 - Para tornar essa tarefa mais simples, a linguagem C permite simplificar algumas expressões

Operador	Significado	Exemplo		
+=	Soma e atribui	x += y	igual a	x = x + y
-=	Subtrai e atribui	x -= y	igual a	x = x - y
*=	Multiplica e atribui	x *= y	igual a	x = x * y
/=	Divide e atribui o quociente	x /= y	igual a	x = x / y
%=	Divide e atribui o resto	x %= y	igual a	x = x % y

OPERADORES DE ATRIBUIÇÃO SIMPLIFICADA

Sem operador

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x = 10, y = 20;
    x = x + y - 10;
    printf("x = %d\n", x);
    x = x - 5;
    printf("x = %d\n", x);
    x = x * 10;
    printf("x = %d\n", x);
    x = x / 15;
    printf("x = %d\n", x);

    return 0;
}
```

Com operador

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x = 10, y = 20;
    x += y - 10;
    printf("x = %d\n", x);
    x -= 5;
    printf("x = %d\n", x);
    x *= 10;
    printf("x = %d\n", x);
    x /= 15;
    printf("x = %d\n", x);

    return 0;
}
```

OPERADORES

Exercício

- Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x, y, z;
x = y = 10;
z = ++x;
x -= x;
y++;
x = x + y - (z--);
```

OPERADORES

○ Exercício

- Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x, y;  
int a = 14, b = 3;  
float z;  
x = a / b;  
y = a % b;  
z = y / x;
```

OPERADORES

○ Exercício

- Diga se as seguintes expressões serão verdadeiras ou falsas:

```
int x = 7;  
(x > 5) || (x > 10)  
(!(x == 6) && (x >= 6))
```

CONVERSÕES DE TIPOS NA ATRIBUIÇÃO

○ Atribuição entre tipos diferentes

- O compilador converte automaticamente o valor do lado direito para o tipo do lado esquerdo do operador de atribuição “=”

○ Pode haver perda de informação

```
int x = 65;
char ch;
float f = 25.1;
//ch recebe 8 bits menos significativos de x
//converte para a tabela ASCII
ch = x;
printf("ch = %c\n", ch); // 'A'
//x recebe parte apenas a parte inteira de f
x = f;
printf("x = %d\n", x); // 25
//f recebe valor 8 bits convertido para real
f = ch;
printf("f = %f\n", f); // 65.000000
//f recebe o valor de x
f = x;
printf("f = %f\n", f); // 25.000000
```

MODELADORES (CASTS)

- Um modelador é aplicado a uma expressão
- Força o resultado da expressão a ser de um tipo especificado.
 - (tipo) expressão

○ Exemplo

```
float x, y, f = 65.5;

x = f / 10.0;
y = (int) (f / 10.0);
printf("x = %f\n", x); //6.550000
printf("y = %f\n", y); //6.000000
```

PRECEDÊNCIA DOS OPERADORES

MAIOR PRECEDÊNCIA	
++ --	Pré-incremento/decremento
()	Parênteses (chamada de função)
[]	Elemento de array
.	Elemento de struct
->	Conteúdo de elemento de ponteiro para struct
++ --	Pós-incremento/decremento
+-	Adição e subtração unária
! ~	Não lógico e complemento bit a bit
(tipo)	Conversão de tipos (<i>type cast</i>)
*	Acesso ao conteúdo de ponteiro
&	Endereço de memória do elemento
sizeof	Tamanho do elemento
*/%	Multiplicação, divisão e módulo (resto)
+-	Adição e subtração
<< >>	Deslocamento de bits à esquerda e à direita
< <=	"Menor do que" e "menor ou igual a"
> >=	"Maior do que" e "maior ou igual a"
== !=	"Igual a" e "diferente de"
&	E bit a bit
^	OU exclusivo
	OU bit a bit
&	E lógico
	OU lógico
?:	Operador ternário
=	Atribuição
+= -=	Atribuição por adição ou subtração
*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)
<<= >>=	Atribuição por deslocamento de bits
&= ^= =	Atribuição por operações lógicas
,	Operador vírgula
MENOR PRECEDÊNCIA	

NOVOS TIPOS EM C

o O tipo **long long int**

- No padrão C99 existe agora um novo tipo inteiro, o **long long int**, o qual define um inteiro de 64 bits.
- Como os demais tipos, este pode ser com ou sem sinal (**unsigned long long int**).
- Foram adicionados também novos especificadores de tipos ("%lld" e "%llu") para uso com as funções **printf()** e **scanf()**

```
long long int x = 123456789123456789LL;

printf("%lld\n", x);
```

NOVOS TIPOS EM C

- O tipo complexo
 - No padrão C99 podemos usar a palavra-chave **_Complex** (ou a macro **complex**) para modificar qualquer tipo de ponto-flutuante (**float**, **double** e **long double**) para que ele se comporte como um par de números de tal modo que defina um número complexo
 - As macros **_Complex_I** e **I** representam a parte imaginária do número complexo

NOVOS TIPOS EM C

- O tipo complexo
 - Além disso, a biblioteca **complex.h** fornece funções para diversas operações sobre números complexos
- Trabalhando com números complexos

```
_Complex float z1 = 2.0 + 2.0 * _Complex_I;  
complex float z2 = 2.0 + 2.0 * I;  
  
complex float z3 = z2 + z1;  
  
printf("%f + %f*I\n", creal(z3), cimag(z3));
```

NOVOS TIPOS EM C

○ O tipo booleano

- O tipo booleano **_Bool** foi adicionado à linguagem C padrão 99, assim como à biblioteca **stdbool.h** por motivos de compatibilidade (nela é definida a macro **bool**, que pode ser usada no lugar de **_Bool**).
- Uma variável definida sob esse novo tipo de dado suporta apenas dois valores: **true** (verdadeiro) com valor igual a 1 e **false** (falso) com valor igual a 0

```
_Bool b = true; //verdadeiro  
b = false; //falso
```

NOVOS TIPOS EM C

○ Novos tipos inteiros

- Tipos inteiros básicos podem variar em diferentes sistemas
- Com o propósito de melhorar a portabilidade de programas, o padrão C99 suporta vários novos tipos inteiros
- Eles estão disponíveis nas bibliotecas **inttypes.h** e **stdint.h**.

NOVOS TIPOS EM C

○ Novos tipos inteiros

- Tipos inteiros de largura exata
 - Garantem possuir N bits em todos os sistemas: `int8_t`, `int16_t`, `int32_t` e `int64_t`.

```
int64_t n;  
int16_t x;
```

- Tipos inteiros de largura mínima
 - Garantem possuir pelo menos N bits em todos os sistemas: `int_least8_t`, `int_least16_t`, `int_least32_t` e `int_least64_t`.

```
int_least8_t n1;  
int_least16_t v;
```

NOVOS TIPOS EM C

○ Novos tipos inteiros

- Tipos inteiros rápidos
 - São os tipos inteiros mais rápidos disponíveis no sistema e que possuem pelo menos N bits: `int_fast8_t`, `int_fast16_t`, `int_fast32_t` e `int_fast64_t`.

```
int_fast32_t valor;  
int_fast64_t total;
```

- A versão **unsigned** desses tipos é obtida acrescentando o prefixo **u** ao nome do tipo

```
uint64_t n;  
uint_least16_t v;  
uint_fast32_t valor;
```


LEITURA E ESCRITA DOS NOVOS TIPOS

- Devemos tomar um cuidado extra na hora de ler e escrever os novos tipos inteiros.
 - Os tipos de saída/entrada padrão do **int** (**%d**, **%i**, **%u**) não funcionam para esses novos tipos inteiros.
 - Devemos usar as macro pré-definidas para cada tipo.
 - Além disso, as novas macros de leitura e escrita não possuem o símbolo de **%** dentro delas. É tarefa do programador acrescentar o símbolo de **%** a respectiva macro usada

LEITURA E ESCRITA DOS NOVOS TIPOS

- Macros para escrita

	Tipos de inteiro e respectivos tipos de saída		
Imprime um	intN_t N = 8, 16, 32 ou 64	int_leastN_t N = 8, 16, 32 ou 64	int_fastN_t N = 8, 16, 32 ou 64
inteiro decimal com sinal	PRIdN PRIiN	PRIdLEASTN PRIiLEASTN	PRIdFASTN PRIiFASTN
inteiro decimal sem sinal	PRIduN	PRIdLEASTN PRIuLEASTN	PRIdFASTN PRIuFASTN

LEITURA E ESCRITA DOS NOVOS TIPOS

Macros para leitura

	Tipos de inteiro e respectivos tipos de entrada		
Lê um	intN_t N = 8, 16, 32 ou 64	int_leastN_t N = 8, 16, 32 ou 64	int_fastN_t N = 8, 16, 32 ou 64
inteiro decimal com sinal	SCNdN SCNiN	SCNdLEASTN SCNiLEASTN	SCNdFASTN SCNiFASTN
inteiro decimal sem sinal	SCNuN	SCNuLEASTN	SCNuFASTN

LEITURA E ESCRITA DOS NOVOS TIPOS

Exemplos

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

int main(){
    //imprime o tamanho do tipo int64_t
    printf("Tamanho = %u\n", sizeof(int64_t));
    //imprime a string que representa a macro PRId64
    printf("String = %s\n", PRId64);
    //imprime o maior valor do tipo int64_t
    printf("Minimo = %"PRId64"\n", INT64_MIN);
    //imprime o menor valor do tipo int64_t
    printf("Maximo = %"PRId64"\n", INT64_MAX);

    int64_t n;
    //Lê uma variável do tipo int64_t
    scanf("%"PRId64, &n);
    //Imprime uma variável do tipo int64_t
    printf("Valor = %"PRId64"\n", n);
    return 0;
}
```

VALORES INFINITY E NAN

- O padrão C99 segue o padrão IEEE 754 para ponto flutuante.
- Isso significa que podemos representar os valores infinito positivo ($+\infty$) ou negativo ($-\infty$) e **NaN** (não um é número), definidos na biblioteca **math.h**
 - **INFINITY**: macro que representa o infinito positivo. É produzido por operações matemáticas como **(1.0/0.0)**.
 - **NAN**: macro que representa o valor de que não um é número. É produzido por operações matemáticas como $\sqrt{-1}$.

VALORES INFINITY E NAN

- Os valores **INFINITY** e **NAN** também suportam operações de comparação entre si
- No entanto, é recomendado utilizar a sua respectiva função macro para testes
 - **isfinite()** para testar se um valor é infinito ou não
 - **isnan()** para testar se um valor não um é número

VALORES INFINITY E NAN

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

int main(){
    double f;
    f = sqrt(-1);
    printf("Resultado: %f\n", f);

    f = 1.0/0.0;
    printf("Resultado: %f\n", f);

    f = -1.0/0.0;
    printf("Resultado: %f\n", f);

    if(isfinite(f))
        printf("Valor finito");
    else
        printf("Valor infinito");

    return 0;
}
```

SAÍDA

Resultado: nan
 Resultado: inf
 Resultado: -inf
 Valor infinito

MATERIAL COMPLEMENTAR

○ Vídeo aulas

- Aula 01: Introdução:
 - www.youtube.com/watch?v=GiCt0Cwcp-U
- Aula 02: Declaração de Variáveis:
 - www.youtube.com/watch?v=q51cHsgRHU4
- Aula 03: printf:
 - www.youtube.com/watch?v=07YPObbEpU8
- Aula 04: scanf:
 - www.youtube.com/watch?v=yQx8sD6vK6M
- Aula 05: Operadores de Atribuição:
 - www.youtube.com/watch?v=tQhnuVR2gc4
- Aula 06: Constantes:
 - youtu.be/GdjGrVjRgTI

MATERIAL COMPLEMENTAR

○ Vídeo aulas

- Aula 07: Operadores Aritméticos:
youtu.be/NsRwpFNZhJs
- Aula 08: Comentários:
youtu.be/8PAWmHdreoc
- Aula 09: Pré e Pós Incremento:
youtu.be/YbVmQKTuajY
- Aula 10: Atribuição Simplificada:
youtu.be/x0uEgxYtW-E
- Aula 11: Operadores Relacionais:
youtu.be/kaivxmdkyTg
- Aula 12: Operadores Lógicos:
youtu.be/TlIEIMmutQo