

Linguagem C

Variáveis e expressões



Prof. Bruno Travençolo
Baseado em slides do Prof. André Backes

Algoritmos

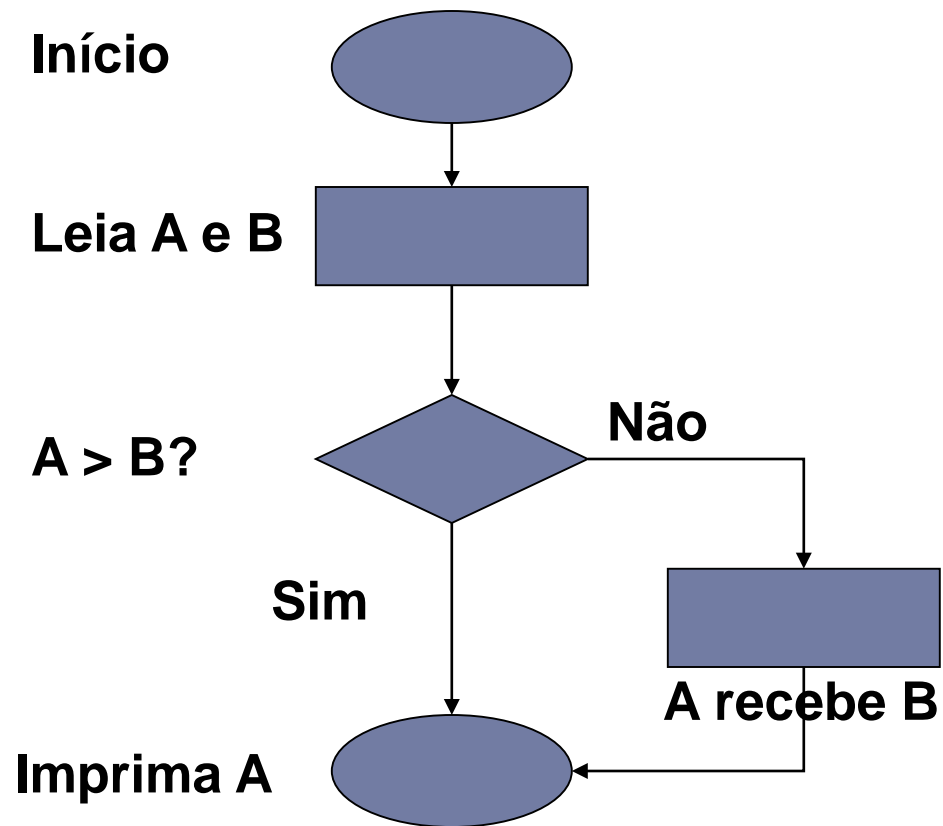
- ▶ O algoritmo é a lógica do nosso problema.
 - ▶ É a sequência de passos que eu faço desenvolvo (na cabeça ou no papel) antes de escrever o programa
 - ▶ Podem existir vários algoritmos diferentes para resolver o mesmo problema



Pseudo-código e Fluxograma

- ▶ Ex.: imprimir maior valor

Leia A;
Leia B;
Se $A > B$ então
Imprima A;
Senão
Imprima B;
Fim Se



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

▶ 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



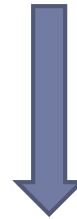
Linguagem C

- ▶ **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
 - ▶ Padrão mais utilizado

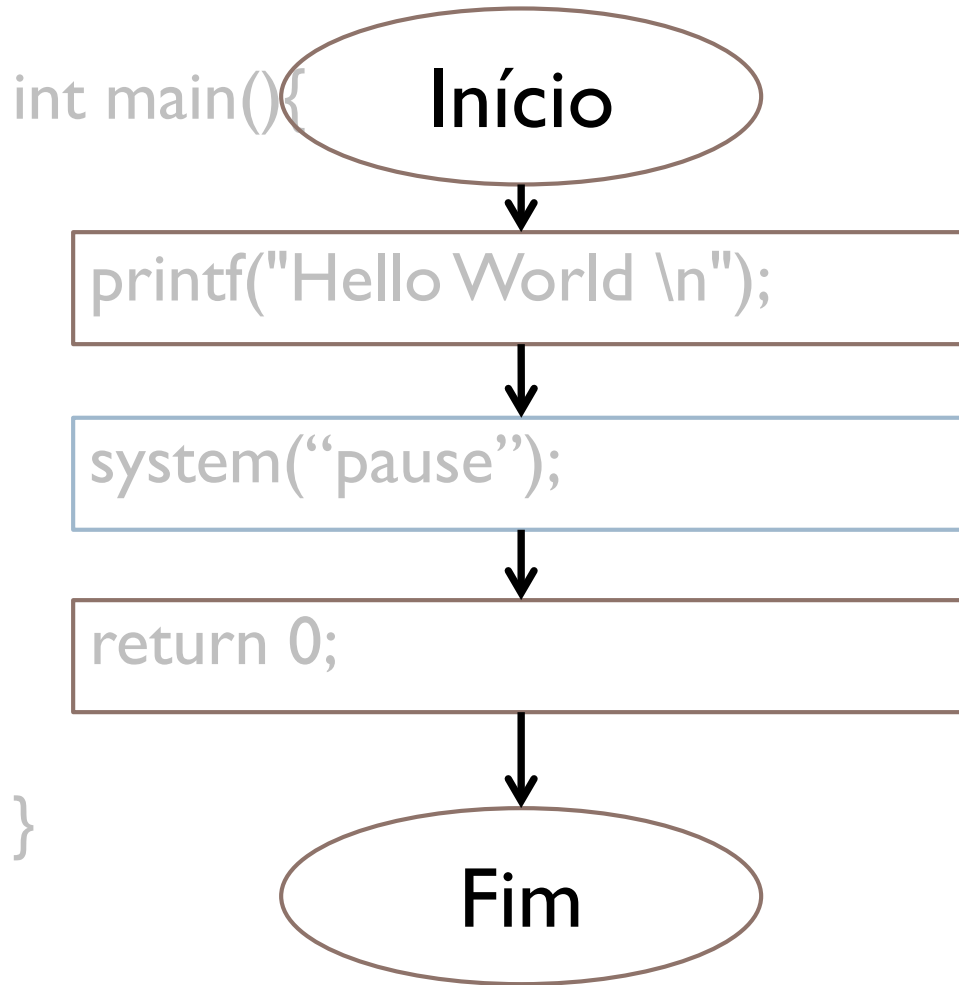


Primeiro programa em C

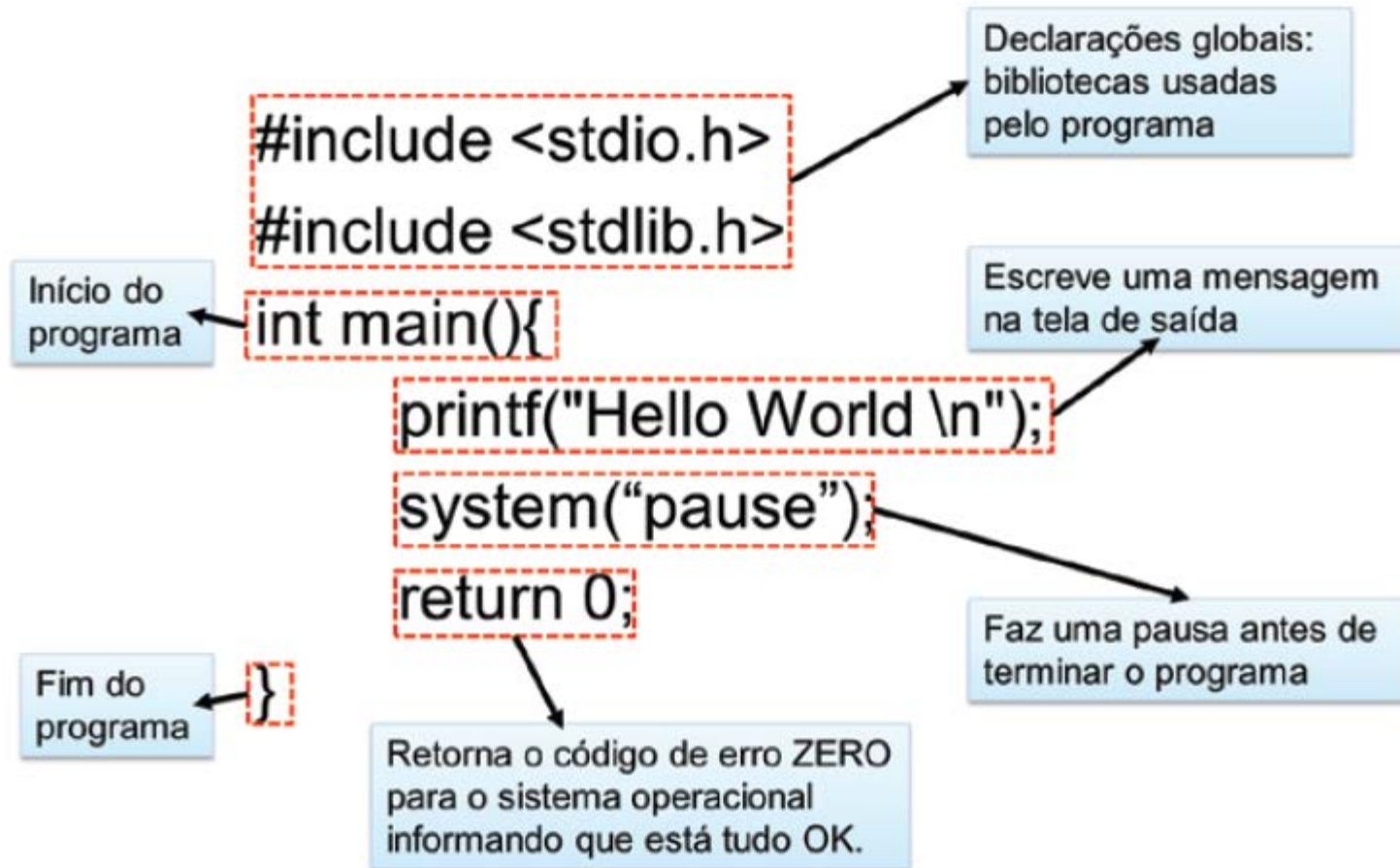
```
#include <stdio.h>
#include <stdlib.h>
int main(){
    printf("Hello World \n");
    system("pause");
    return 0;
}
```



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.

```
*main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      /* a função printf() serve para escrever na
7       tela */
8
9      printf("Hello world!\n");
10
11     // Faz uma pausa no programa (em Windows)
12     system("pause");
13
14     // Retorna 0 (zero) para o sistema operacional
15     return 0; // FIM DO PROGRAMA
16 }
17
```

Comentário com uma ou mais linhas

Início: /*

Fim: */

Comentário em uma única linha

Início: //

Fim: sem símbolos, é o próprio final da linha

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
 - ▶ Um nome de uma pessoa
 - ▶ O valor da temperatura atual
 - ▶ A quantidade de alunos em uma sala de aula
 - ▶ Se um assento de uma aeronave está ocupado



Declaração de Variáveis

▶ Tipos de dados

- ▶ Um nome de uma pessoa
 - ▶ Uma cadeia de caracteres (“Bruno” - 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)



Declaração de Variáveis

► Declaração de variáveis em C

► <tipo de dado> nome-da-variável



Variáveis

▶ Propriedades

▶ Nome

- ▶ Pode ter um ou mais caracteres
- ▶ Nem tudo pode ser usado como nome

▶ Tipo

- ▶ Conjunto de valores aceitos

▶ Escopo (*tema para uma outra aula*)

- ▶ 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 (*Case sensitive*)



Observações sobre declaração de variáveis

- ▶ Não utilizar espaços nos nomes
 - ▶ Exemplo: nome do aluno, temperatura do sensor,
- ▶ Não utilizar acentos ou símbolos
 - ▶ Exemplos: garça, tripé, °, ⊕
- ▶ 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

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



Variáveis

▶ Quais nomes de variáveis estão corretos:

- ▶ Contador
- ▶ contador l
- ▶ comp!
- ▶ .var
- ▶ Teste_123
- ▶ _teste
- ▶ int
- ▶ int l
- ▶ lcontador
- ▶ -x
- ▶ Teste-123 x&



Observações sobre declaração de variáveis

- ▶ Não utilizar espaços nos nomes
 - ▶ Exemplo: nome do aluno: caracter
- ▶ Não utilizar acentos ou símbolos
 - ▶ Exemplos: garça, tripé, °, ⊕
- ▶ 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

- ▶ **Corretos:**

- ▶ Contador, contador l, Teste_ l23, _teste, int l

- ▶ **Errados**

- ▶ comp!, .var, int, l contador, -x, Teste- l23, 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'!***

```
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**: um inteiro cujo tamanho depende do processador, tipicamente 16 ou 32 bits

```
int NumeroAlunos;
```

```
int Idade;
```

```
int NumeroContaCorrente;
```

```
int N = 10; // o variável N recebe o valor 10
```



Tipos básicos em C

▶ Números Reais (\mathbb{R})

- ▶ Tipos: *float*, *double* e *long double*
- ▶ Pode-se escrever números usando notação científica

```
TempoTotal = 0.000000003295;
```

```
TempoTotal = 3.2950e-009; // notação científica, que  
                        // equivale à 3,295x10-9
```

- ▶ parte decimal usa ponto e não vírgula!
- ▶ Parte dos bits armazena a mantissa, e outra parte o expoente
- ▶ São números chamados de ‘ponto flutuante’ devido à forma como são representados
 - 3.2950e-009
 - 3.2950e-008
 - 3.2950e-011
- ▶ Parte decimal pode ‘flutuar’, mover, relativo aos dígitos significantes (mantissa, números antes do expoente)



Tipos básicos em C

- ▶ **Números Reais**

- ▶ **float**: um número real com precisão simples

```
float Temperatura; // pode receber, por exemplo, 23.30
float MediaNotas; // pode receber, por exemplo, 7.98
float TempoTotal; // pode receber 0.0000000032 (s) ou
                  // 3.2000e-009 (notação científica)
                  // que equivale à  $3,2 \times 10^{-9}$ 
```

- ▶ **double**: um número real com precisão dupla

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



Variáveis

Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)	
signed char	8-bit	-128 to 127	
unsigned char	8-bit	0 to 255	
char	8-bit	* Mesmo que unsigned/signed char, a depender do sistema	
short int	16-bit	-32,768 to 32,767	short; signed short int; signed short
unsigned short int	16-bit	0 to 65,535	unsigned short
int	32-bit	-2,147,483,648 to 2,147,483,647	signed int; signed
unsigned int		0 to 4,294,967,295	unsigned



Variáveis

Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)	
<code>long long int</code>	64-bit	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	<code>long long;</code> <code>signed</code> <code>long long</code> <code>int;</code> <code>signed</code> <code>long long</code>
<code>unsigned long long int</code>	64-bit	0 to 18,446,744,073,709,551,615	<code>unsigned</code> <code>long long</code>

This type is not part of C89, but is both part of C99 and a GNU C extension.

<https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Primitive-Types>



Variáveis

Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)		
float	32-bit	FLT_MIN	= 1.175494e-38	
		FLT_MAX	= 3.402823e+38	
double	64-bit	FLT_MIN	= 2.225074e-308	
		FLT_MAX	= 1.797693e+308	
long double	96-bit	FLT_MIN	= 3.362103e-4932	
		FLT_MAX	= 1.189731e+4932	

All floating point data types are signed to precisely represent numbers such as, for example, 4.2. For this reason, we recommend that you consider not comparing real numbers for exact equality with the == operator, but rather check that real numbers are within an acceptable tolerance.

<https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Primitive-Types>



Variáveis

de valores
A 127
255
A 127
A 2.147.483.647
4.967.295
A 32.767
A 32.767
5.535
A 32.767
A 2.147.483.647
4.967.295
A 2.147.483.647
3,402823E+038
A 1,797693E+308
3,4E+4932

Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)
signed char	8-bit	-128 to 127
unsigned char	8-bit	0 to 255
char	8-bit	* Mesmo que unsigned/signed char, a dependo do sistema
short int	16-bit	-32,768 to 32,767
unsigned short int	16-bit	0 to 65,535
int	32-bit	-2,147,483,648 to 2,147,483,647
unsigned int	32-bit	0 to 4,294,967,295

<https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Primitive-Types>

you should use the char data type specifically for storing ASCII

- ▶ characters (such as ``m'`), including escape sequences (such as ``n'`).

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; /* em pseudolinguagem  
               representamos assim: x <- 5 */  
    int y;  
    y = x + 3;  
}
```

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



Atribuição

► Como “ler em voz alta”

```
int x = 5; // x recebe 5 (e não 'x é igual a 5')  
y = x + 3; // y recebe x mais 3  
y = y + 5; // y recebe y mais 5  
y = y + x; // y recebe y mais x
```

► Em pseudocódigo

```
x <- 5  
x <- x + 3;  
y <- y + 5;  
y <- y + x;
```



Erros comuns

- ▶ O símbolo ponto (.) é um separador decimal, e não separador de milhar

```
int Habitantes;
```

```
Habitantes = 110.000; // 110 mil habitantes  
printf("%d", Habitantes); // resposta: 110 habitantes (e não 110 mil)
```



Comando de saída

- ▶ **printf()**

- ▶ *print formatted*

- ▶ Sintaxe: **printf**("format",arg1,...)

- ▶ format – texto a ser mostrado e formatações

- ▶ arg1, arg2, ... – valores a serem mostrados

- ▶ Exemplos:

- ```
printf("Hello World");
```

- ```
printf("Faculdade de Computação - Universidade  
Federal de Uberlândia"); // obs: tudo em uma linha só  
// para usar mais de uma linha use barra invertida \
```



Comando de saída

▶ **printf()**

- ▶ Como mostrar o valor contido em uma variável?
- ▶ Utilizar especificadores de formato (*format specifiers*)
 - ▶ Subseqüências iniciando com o símbolo de porcentagem:

%

- ▶ Um símbolo é usado para cada tipo de variável a ser mostrada
 - ▶ Por ex., a letra **f** é usada para mostrar valores do tipo **float**

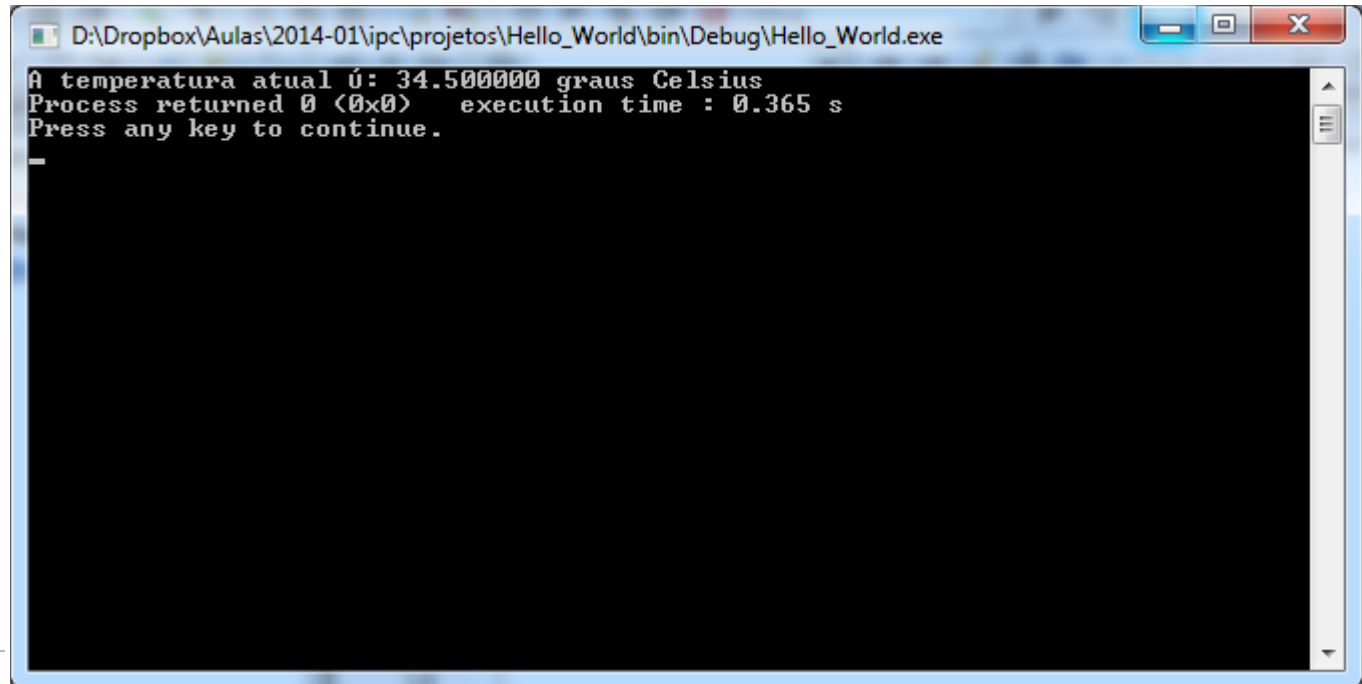
```
float TemperaturaAtual = 34.5;  
printf("A temperatura atual é: ");  
printf("%f", TemperaturaAtual);  
printf(" graus Celsius");
```



Comando de saída

► printf()

```
float TemperaturaAtual = 34.5;  
printf("A temperatura atual é: ");  
printf("%f", TemperaturaAtual);  
printf(" graus Celsius");
```



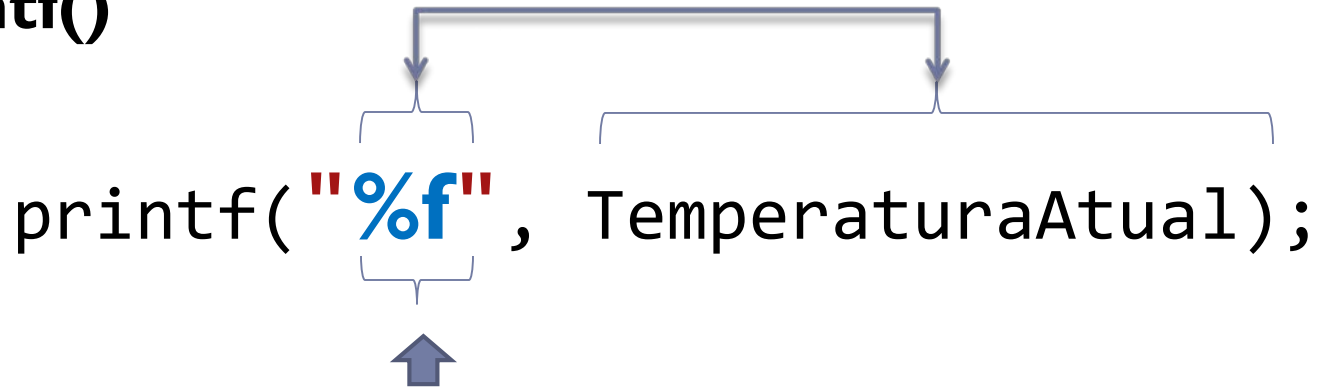
The screenshot shows a Windows command prompt window titled "D:\Dropbox\Aulas\2014-01\ipc\projetos\Hello_World\bin\Debug\Hello_World.exe". The window contains the following text:

```
A temperatura atual é: 34.500000 graus Celsius  
Process returned 0 (0x0)   execution time : 0.365 s  
Press any key to continue.  
-
```

The output matches the code shown in the previous block, demonstrating the use of the `printf` function to format and print floating-point numbers and strings.

Comando de saída

► printf()



The diagram illustrates the components of the `printf` command. The command is `printf("%f", TemperaturaAtual);`. A bracket under the opening quote of the format string points to a callout box. A bracket under the variable `TemperaturaAtual` points to the second arrow of a horizontal line with two downward-pointing arrows, indicating the argument being passed.

```
printf("%f", TemperaturaAtual);
```

O especificador de formato fica dentro da região entre aspas do comando `printf`

Comando de saída

► **printf()**

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

```
float TemperaturaAtual = 34.5;
```

```
printf("A temperatura atual é %f graus Celsius: ", TemperaturaAtual);
```



Comando de saída

▶ **printf()**

- ▶ Podemos mostrar mais de uma variável
 - ▶ Sintaxe: **printf**("format",arg1,...)
 - ▶ A sintaxe diz que podemos ter vários argumentos (arg1, arg2, ...)

// cálculo do IMC (índice de massa corporal): $\text{Peso(kg)} / (\text{Altura}^2)$

```
float peso = 82.5;
```

```
float altura = 1.70;
```

```
float IMC;
```

```
IMC = peso / (altura*altura);
```

```
printf("Peso: %f, Altura: %f, IMC: %f", peso, altura, IMC);
```



The diagram consists of three curved arrows originating from the variable names in the printf statement and pointing to their respective values in the calculation above. One arrow starts from 'peso' and points to '82.5'. Another arrow starts from 'altura' and points to '1.70'. A third arrow starts from 'IMC' and points to the result of the division 'IMC = peso / (altura*altura);'.

Comando de saída

► printf()

```
// conceitos de uma universidade
char conceito_bom, conceito_ruim;
conceito_bom = 'A'; // Note que atribuição para o tipo char precisa de aspas simples
conceito_ruim = 'F';
printf("O melhor conceito é %c e o pior é %c", conceito_bom, conceito_ruim);
```

> Saída: O melhor conceito é A e o pior é F



Comando de saída

► printf()

```
// preço de produtos
int qte = 10;
float preco_unitario = 2.50;
printf("Quantidade de produtos: %d \n", qte);
printf("Preço unitário(R$): %f \n", preco_unitario);
printf("Valor total (R$): %f \n", preco_unitario*qte);
```

➤ Saída:

```
Quantidade de produtos: 10
Preço unitário (R$): 2.500000
Valor total (R$): 25.000000
```



Comando de saída

▶ **printf()**

- ▶ Comando que realiza a impressão dos dados do programa

- ▶ `printf("tipo de saída", lista de variáveis)`



← `printf("texto %tipo_de_saída texto" expressão);`

▶ Alguns tipos de saída

- ▶ `%c` – escrita de **um** caractere
 - ▶ `%d` – escrita de números inteiros
 - ▶ `%f` – escrita de número reais
 - ▶ `%s` – escrita de **vários** caracteres



Comando de saída

▶ Ex:

- ▶ Saída de um único valor inteiro

```
printf("%d",x);
```

- ▶ Saída de mais de um único valor

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

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

- ▶ No tipo de saída, pode-se formatar toda a saída

- ▶ `printf("Total = %d",x+y);`



Comando de entrada

- ▶ Comandos de entrada servem para obtermos informações dos dispositivos de entrada ligados ao computador
- ▶ Nos algoritmos indicamos os comando de entrada como “leia” um determinado valor; (“read”)
- ▶ Exemplos:
 - ▶ Teclado
 - ▶ Usuário deve digitar uma informação para prosseguir com o uso de um programa (e.g., nome, idade, endereço)
 - ▶ Ler nome; Ler endereço
 - ▶ Leitor de código de barras
 - ▶ O software do supermercado fica aguardando que um código de barras passe pelo leitor de código de barras para então buscar o produto e seu preço para lançar na nota fiscal
 - ▶ Ler código de barras



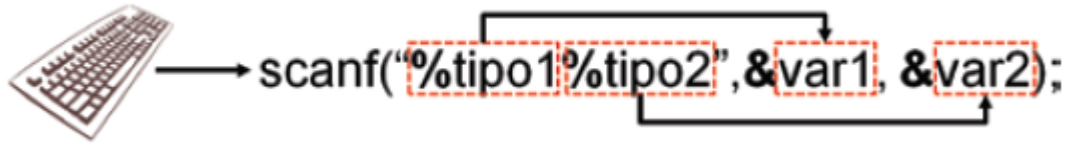
Comando de entrada

- ▶ Em C, o comando que permite ler dados da entrada padrão (no caso o teclado) é o `scanf()`
- ▶ **`scanf()`**
- ▶ Sintaxe: **`scanf("format",&name1,...)`**
 - ▶ `format` – especificador de formato da entrada que será lida
 - ▶ `&name1, &name2, ...` – endereços das variáveis que irão receber os valores lidos



Comando de entrada

- ▶ Temos, igual ao comando printf, que especificar o tipo (formato) do dado que será lido
- ▶ scanf("tipo de entrada", lista de variáveis)



- ▶ Alguns “tipos de entrada”
 - ▶ %c – leitura de **um** caractere
 - ▶ %d – leitura de números inteiros
 - ▶ %f – leitura de float
 - ▶ %s – leitura de **vários** caracteres

Comando scanf() - Exemplo

```
// declaração das variáveis
```

```
float peso;
```

```
float altura;
```

```
float IMC;
```

```
// Obtendo os dados do usuário
```

```
printf("Informe o peso: ");
```

```
scanf("%f",&peso);
```

```
printf("Informe a altura: ");
```

```
scanf("%f",&altura);
```

```
// calculando o ICM e mostrando o resultado
```

```
IMC = peso / (altura*altura);
```

```
printf("Peso: %f, Altura: %f, IMC: %f", peso, altura, IMC);
```



Comando scanf() - Exemplo

- ▶ Como “ler em voz alta”

```
scanf("%f",&peso); // leia um valor real (do tipo  
float) e armazene no endereço da variável peso
```

- ▶ O símbolo & indica qual é o endereço da variável que vai receber os dados lidos
 - ▶ peso – variável peso
 - ▶ &peso – endereço da variável peso



Comando de entrada

- ▶ **Ex:**

- ▶ Leitura de um único valor

```
int x;
```

```
scanf("%d",&x);
```

- ▶ Podemos ler mais de um valor em um único comando

```
int x,y;
```

```
scanf("%d%d",&x,&y);
```

- ▶ Obs: na leitura de vários valores, separar com espaço, TAB, ou Enter.



Tipos Booleanos em C

- ▶ Um tipo booleano pode assumir dois valores:
 - ▶ verdadeiro ou falso (true ou false)
- ▶ Na linguagem C não existe o tipo de dado booleano.
- ▶ Para armazenar esse tipo de informação, use-se uma variável do tipo `int` (número inteiro)
 - ▶ Valor 0 significa falso / números + ou – : verdadeiro
- ▶ Exemplos:

```
int AssentoOcupado = 1; // verdadeiro (o assento está  
                        // ocupado)
```

```
int PortaAberta = 0; // falso (a porta está fechada)
```



Operadores

- ▶ Os operadores são usados para desenvolver operações matemáticas, relacionais e lógicas.
- ▶ Podem ser
 - ▶ Unários
 - ▶ Binários
 - ▶ Bit a bit (será visto em outra aula)



Operadores

► Operadores Unários

Op	Uso	Exemplo
+	mais unário ou positivo	+X
-	menos unário (número oposto)	- X
!	NOT ou negação lógica	! x
&	Endereço	&x

Existem outros operadores que serão vistos em outras aulas



Operador Unário

```
int num;  
int oposto_num;  
num = -10;  
oposto_num = -num;  
printf("Num. %d => Valor oposto %d", num, oposto_num);
```

> Saída: Num. -10 => Valor oposto 10



Operador Unário

```
int ligado;  
ligado = 1; // valor booleano (verdadeiro)  
  
printf("Ligado: %d; Desligado %d", ligado, !ligado);
```

> Saída: Ligado: 1; Desligado 0



Operadores

► Binários

Operador	Descrição	Exemplo
+	Adição de dois números	$z = x + y;$
-	Subtração de dois números	$z = x - y;$
*	Multiplicação de dois números	$z = x * y;$
/	Quociente de dois números	$z = x / y;$
%	Resto da divisão	$z = x \% y;$



Operadores: cuidados com símbolos iguais

Op	Uso	Exemplo
+	mais unário ou positivo	+x
-	menos unário ou negação	-x
!	NOT ou negação lógica	!x
&	Endereço	&x

// operador unário. y recebe o
// negativo de x

y = -x;

// operador binário: z é
// é o valor k menos y

z = k - y;

Op	Descrição	Exemplo
+	Adição de dois números	z = x + y;
-	Subtração de dois números	z = x - y;
*	Multiplicação de dois números	z = x * y;
/	Quociente de dois números	z = x / y;
%	Resto da divisão	z = x % y;

Operadores Relacionais e Lógicos

- ▶ Operadores relacionais: comparação entre variáveis
- ▶ Esse tipo de operador retorna *verdadeiro* (1) ou *falso* (0)

Op	Descrição	Exemplo
>	Maior do que	Idade > 6
>=	Maior ou igual a	Nota >= 60
<	Menor do que	Valor < Temperatura
<=	Menor ou igual a	Velocidade <= MAXIMO
==	Igual a	Opcao == 'a'
!=	Diferente de	Opcao != 's'



Operadores Relacionais e Lógicos

▶ Exemplos

Expressão	Resultado
▶ <code>x=5; x > 4</code>	verdadeiro (1)
▶ <code>x=5; x == 4</code>	falso (0)
▶ <code>x=5; y=3; x != y</code>	verdadeiro (1)
▶ <code>x=5; y=3; x != (y+2)</code>	falso (0)



Operadores Relacionais e Lógicos

- ▶ Operadores lógicos: operam com valores lógicos e retornam um valor lógico *verdadeiro* (1) ou *falso* (0)

Op	Função	Exemplo
&&	AND (E)	(c >= '0' && c <= '9')
	OR (OU)	(a == 'F' b != 32)
!	NOT	!continuar



Operadores Relacionais e Lógicos

► Tabela verdade

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 Relacionais e Lógicos

▶ Exercício

- ▶ Diga se as seguintes expressões serão verdadeiras ou falsas:

$((10 > 5) \ || \ (5 > 10))$

$(!(5 == 6) \ \&\& \ (5 != 6) \ \&\& \ ((2 > 1) \ || \ (5 <= 4)))$



Expressões

- ▶ Expressões são combinações de variáveis, constantes e operadores.
- ▶ Exemplos:

`Anos = Dias/365.25;`

`i = i+3;`

`c= a*b + d/e;`

`c= a*(b+d)/e;`



Precedência dos Operadores

Maior Precedência




Menor Precedência

Operators (grouped by precedence)

structure member operator	<i>name.member</i>
structure pointer	<i>pointer->member</i>
increment, decrement	++ , --
plus, minus, logical not, bitwise not	+ , - , ! , ~
indirection via pointer, address of object	*pointer , &name
cast expression to type	(type) expr
size of an object	sizeof
multiply, divide, modulus (remainder)	* , / , %
add, subtract	+ , -
left, right shift [bit ops]	<< , >>
comparisons	> , >= , < , <=
comparisons	== , !=
bitwise and	&
bitwise exclusive or	^
bitwise or (incl)	
logical and	&&
logical or	
conditional expression	<i>expr₁ ? expr₂ : expr₃</i>
assignment operators	+= , -= , *= , ...
expression evaluation separator	,

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Operators (grouped by precedence)



structure member operator	<i>name.member</i>
structure pointer	<i>pointer->member</i>
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, - , ! , ~
indirection via pointer, address of object	* <i>pointer</i> , & <i>name</i>
cast expression to type	(<i>type</i>) <i>expr</i>
size of an object	sizeof
multiply, divide, modulus (remainder)	*, /, %
add, subtract	+, -
left, right shift [bit ops]	<<, >>
comparisons	>, >=, <, <=
comparisons	==, !=
bitwise and	&
bitwise exclusive or	^
bitwise or (incl)	
logical and	&&
logical or	
conditional expression	<i>expr</i> ₁ ? <i>expr</i> ₂ : <i>expr</i> ₃
assignment operators	+=, -=, *=, ...
expression evaluation separator	,

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Observe que o operador unário – (negativo) tem precedência sobre o operador binário – (subtração)

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



Slides Extras (não passei na aula do dia 26/05).
Poderão fazer parte de outras aulas



Material Complementar

- ▶ **Vídeo aulas**
 - ▶ Aula 01: Introdução
 - ▶ Aula 02: Declaração de Variáveis
 - ▶ Aula 03: Printf
 - ▶ Aula 04: Scanf
 - ▶ Aula 05: Operadores de Atribuição
 - ▶ Aula 06: Constantes
 - ▶ Aula 07: Operadores Aritméticos
 - ▶ Aula 08: Comentários
 - ▶ Aula 09: Pré e Pós Incremento
 - ▶ Aula 10: Atribuição Simplificada
 - ▶ Aula 11: Operadores Relacionais
 - ▶ Aula 12: Operadores Lógicos



Comando de entrada

► **getchar()**

- Comando que realiza a leitura de um único caractere

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      char c;
05      c = getchar();
06      printf("Caractere: %c\n", c);
07      printf("Codigo ASCII: %d\n", c);
08      system("pause");
09      return 0;
10  }
```



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:

```
#define PI 3.1415
```

▶ Usando **const**

- ▶ Usando **const**, a declaração não precisa estar no início do código.

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



Constantes char

- ▶ A linguagem C utiliza vários códigos chamados códigos de barra invertida.

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)



Constantes char

01	#include <stdio.h>
02	#include <stdlib.h>
03	int main(){
04	printf("Hello World\n");
05	printf("Hello\nWorld\n");
06	printf("Hello \\World\n");
07	printf("\"Hello World\"\n");
08	system("pause");
09	return 0;
10	}

Saída	Hello World Hello World Hello \ World "Hello World"
-------	---



Conversão de tipos

```
int Inteiro;  
float Real;
```

```
Real = 1/3;  
printf("%f \n",Real); // resposta 0.00000
```

```
Real = 1/3.0;  
printf("%f \n",Real); // resposta 0.33333
```

```
Inteiro = 3;
```

```
Real = 1/Inteiro;  
printf("%f \n",Real); // resposta 0.00000
```

```
Real = 1/(float)Inteiro;  
printf("%f \n",Real); // resposta 0.33333
```

```
Real = 2.9;  
Inteiro = Real;  
printf("%d \n",Inteiro); // resposta 2
```



Conversões de Tipos na Atribuição

▶ Atribuição entre tipos diferentes

- ▶ O compilador converte automaticamente o valor do lado direito para o tipo do lado esquerdo de “=”
- ▶ Pode haver perda de informação
- ▶ Ex:

```
int x; char ch; float f;
```

```
ch = x; /* ch recebe 8 bits menos significativos de x */
```

```
x = f; /* x recebe parte inteira de f */
```

```
f = ch; /* f recebe valor 8 bits convertido para real */
```

```
f = x; /* idem para inteiro x */
```



Operadores

► Operadores Unários

- : menos unário ou negação	-x
! : NOT ou negação lógica	!x
&: endereço	&x
*: conteúdo (ponteiros)	(*x)
++: pré ou pós incremento	++x ou x++
-- : pré ou pós decremento	-- x ou x --



Operadores

- ▶ **Diferença entre pré e pós incremento/decremento**
 - ▶ $y = x++$: incrementa depois de atribuir
 - ▶ $y = ++x$: incrementa antes de atribuir



Operadores

► Ex:

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

► Resultado

```
11  
10  
12  
12
```



Operadores

- ▶ Operações bit-a-bit: o número é representado por sua forma binária e as operações são feitas em cada bit dele.
 - ▶ <<: desloca à esquerda $x \ll 2$
 - ▶ >>: desloca à direita $x \gg 2$
 - ▶ ^ : ou exclusivo $x \wedge 0xF0$
 - ▶ & : E bit-a-bit $x \& 0x07$
 - ▶ | : OU bit-a-bit $x | 0x80$
 - ▶ ~ : Complementa bit-a-bit $\sim x$



Operadores

- ▶ As operações bit-a-bit ajudam programadores que queiram trabalhar com o computador em "baixo nível".
- ▶ Essas operações só podem ser usadas nos tipos char, int e long.



Operadores

- ▶ Exemplo de operador bit a bit

- ▶ `char x = 0xD5;`

1	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

- ▶ `0x0F`

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

- ▶ `x & 0x0F`

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---



Operadores

▶ Exemplo de operador bit a bit

▶ `char x = 0xD5;`

▶ `x << 2`

▶ `(x*4)`

▶ `x >> 2`

▶ `(x/4)`

1	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---



Operadores Simplificados

- ▶ O C permite simplificar algumas expressões matemáticas

`+=` : soma e atribui

`x += y; <=> x = x + y;`

`-=` : subtrai e atribui

`x -= y; <=> x = x - y;`

`*=` : multiplica e atribui

`x *= y; <=> x = x * y;`

`/=` : divide e atribui quociente

`x /= y; <=> x = x / y;`

`%=` : divide e atribui resto

`x %= y; <=> x = x % y;`

`&=` : E bit-a-bit e atribui

`x &= y; <=> x = x & y;`

`|=` : OU bit-a-bit e atribui

`x |= y; <=> x = x | y;`

`<<=` : shift left e atribui

`x <<= y; <=> x = x << y;`



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

- ▶ Quais os valores de a, b e c após a execução do código abaixo?

```
int a = 10, b = 20, c;
```

```
c = a+++b;
```



Operadores

▶ Exercício

- ▶ Quais os valores de x, y, e z após a execução do código abaixo?

```
int x,y;
```

```
int a = 14, b = 3;
```

```
float z;
```

```
x = a/b;
```

```
y = a%b;
```

```
z = y/x;
```



Modeladores (Casts)

- ▶ Um modelador é aplicado a uma expressão.
- ▶ Força o resultado da expressão a ser de um tipo especificado.
 - ▶ (tipo) expressão
- ▶ Ex:
 - ▶ (float) x;
 - ▶ (int) x * 5.25;



Operador vírgula (,)

- ▶ O operador “,” determina uma lista de expressões que devem ser executadas seqüencialmente.
 - ▶ Ex: $x = (y=2, y+3);$
 - ▶ Nesse caso, o valor 2 é atribuído a y , se somará 3 a y e o retorno (5) será atribuído à variável x .
 - ▶ Pode-se encadear quantos operadores “,” forem necessários.



Modeladores (Casts)

► Ex:

```
int num;  
float f;  
num = 10;  
f = num/7;  
printf ("%f \n", f);  
f = (float)num/7;  
printf ("%f", f);
```

► Resultado

```
1.000000  
1.428571
```

