


Linguagem C

Variáveis e expressões



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

Algoritmos

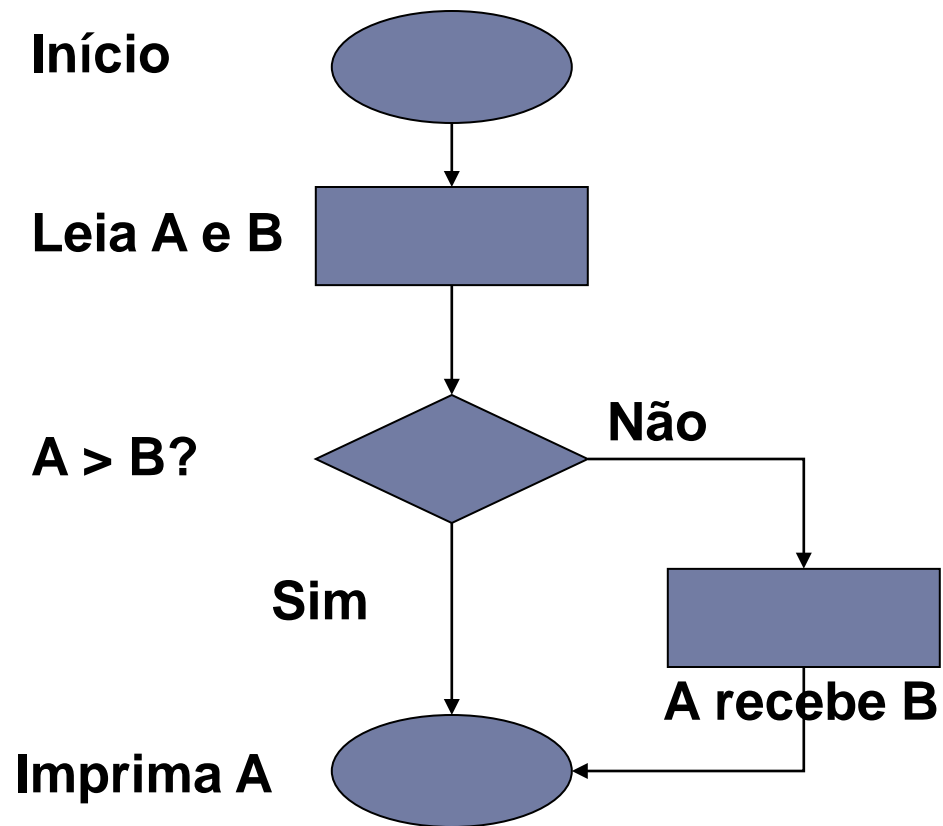
- ▶ O algoritmo é a lógica do nosso problema.
 - ▶ É a sequência de passos que eu faço desenvolvo (na cabeça ou no papel) antes de escrever o programa
 - ▶ Podem existir vários algoritmos diferentes para resolver o mesmo problema



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



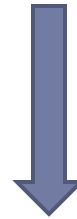
Linguagem C

- ▶ **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

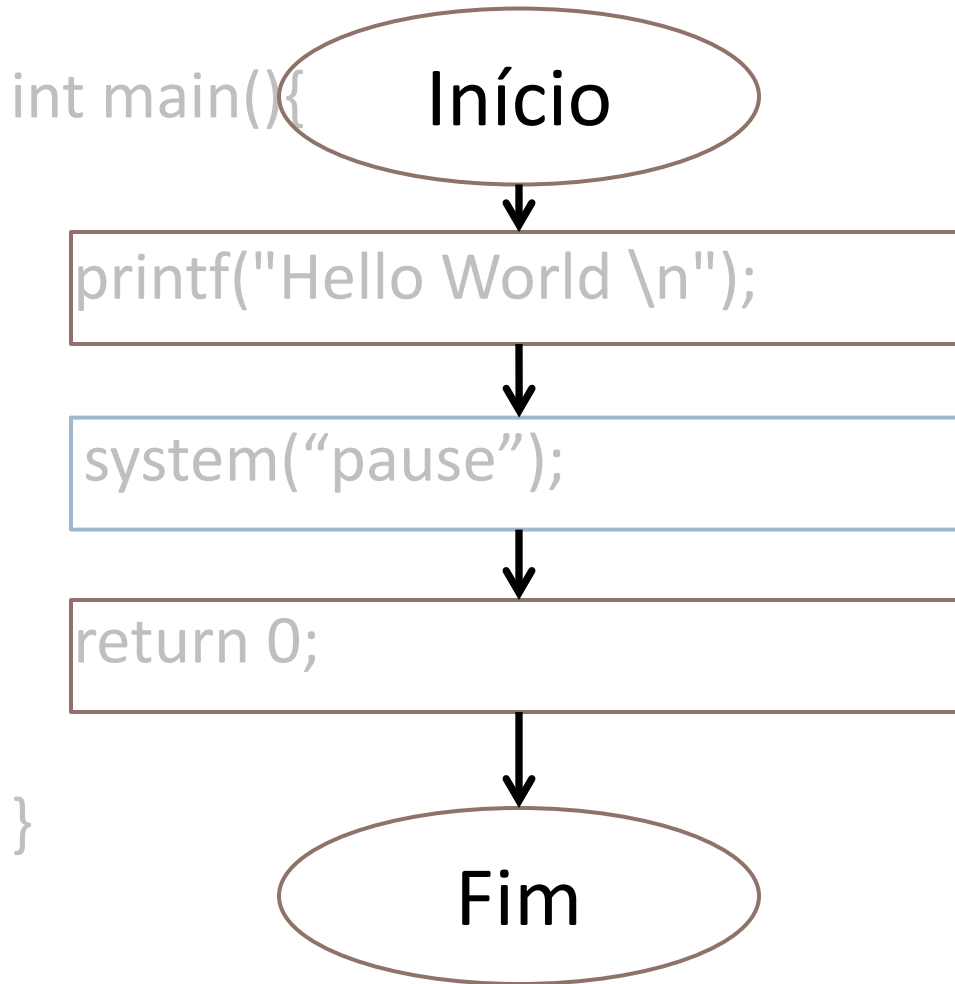


Primeiro programa em C

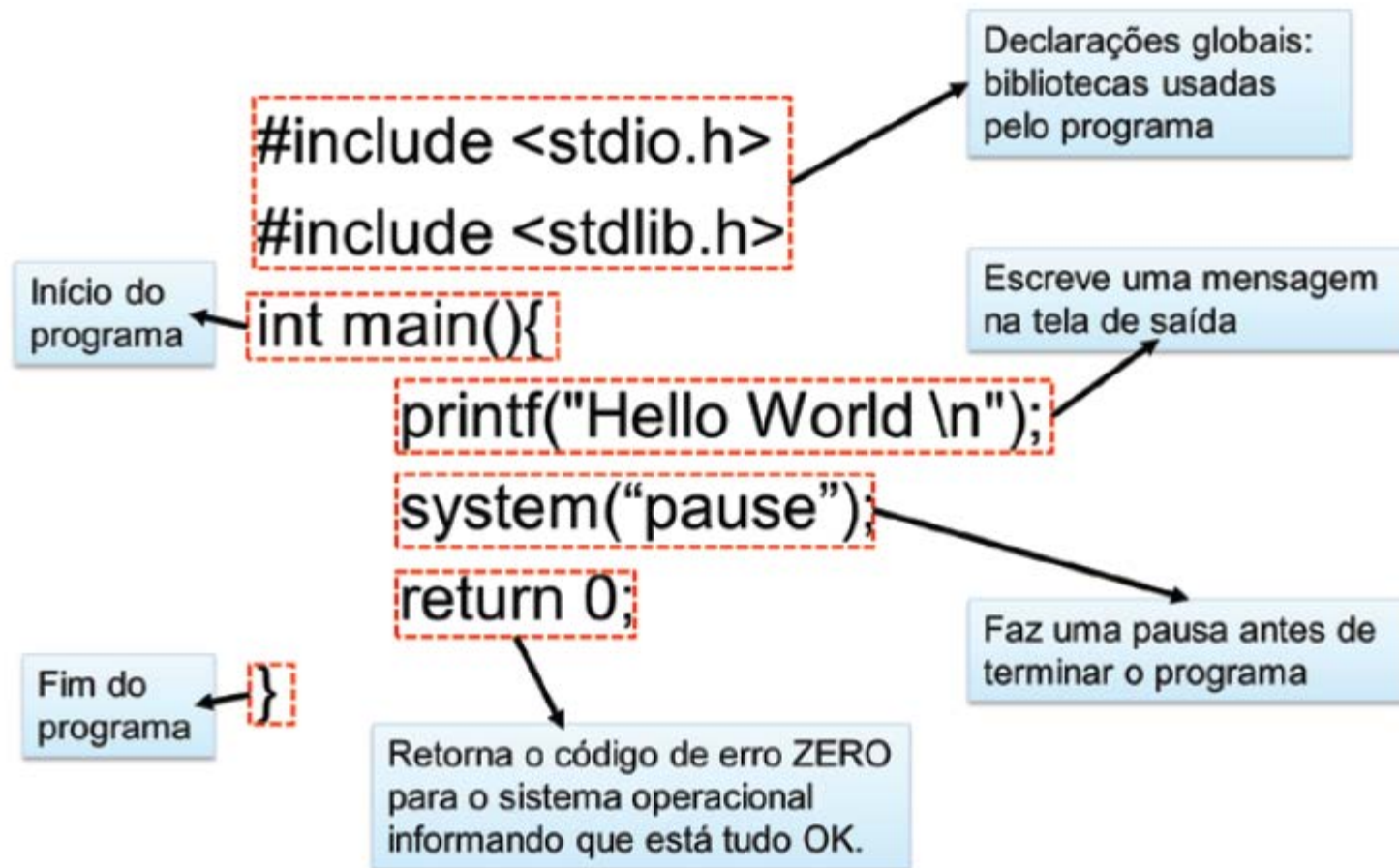
```
#include <stdio.h>
#include <stdlib.h>
int main(){
    printf("Hello World \n");
    system("pause");
    return 0;
}
```



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
 - ▶ contador1
 - ▶ comp!
 - ▶ .var
 - ▶ Teste_123
 - ▶ _teste
 - ▶ int
 - ▶ int1
 - ▶ 1contador
 - ▶ -x
 - ▶ Teste-123 x&



Observações sobre declaração de variáveis

- ▶ Não utilizar espaços nos nomes
 - ▶ Exemplo: nome do aluno: caractere
- ▶ 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 dentro do mesmo escopo (isso será discutido adiante)



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'!***

```
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
```



Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)	<i>alias</i>
signed char	8-bit (1 byte)	-128 to 127	
unsigned char	8-bit (1 byte)	0 to 255	
char	8-bit (1 byte)	* Mesmo que unsigned/signed char, a depender do sistema	
short int	16-bit (2 bytes)	-32,768 to 32,767	short; signed short int; signed short
unsigned short int	16-bit (2 bytes)	0 to 65,535	unsigned short
int	32-bit (4 bytes)	-2,147,483,648 to 2,147,483,647	signed int; signed
unsigned int	32-bit (4 bytes)	0 to 4,294,967,295	unsigned



Tipos

Tipo (<i>type</i>)	Bits	Intervalo de valores (<i>range</i>)	
<code>long int</code>	32 ou 64	Depende do sistema (ver <i>int</i> ou <i>Long Long int</i>)	<code>long</code> , <code>signed long int</code>
<code>unsigned long int</code>	32 ou 64	Depende do sistema (ver <i>int</i> ou <i>Long Long int</i>)	<code>unsigned long</code>
<code>long long int</code>	64-bit (8 bytes)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	<code>long long</code> ; <code>signed long long int</code> ; <code>signed long long</code>
<code>unsigned long long int</code>	64-bit (8 bytes)	0 to 18,446,744,073,709,551,615	<code>unsigned long long</code>

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



Tipos

18,446,744,073,709,551,615

Dezoito quintilhões, quatrocentos e quarenta e seis quatrilhões, setecentos e quarenta e quatro trilhões, setenta e três bilhões, setecentos e nove milhões, quinhentos e cinquenta e um mil e seiscentos e quinze

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>



Tipos

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	

Todos os números em ponto flutuante possuem sinal

O uso da representação binária para números reais faz com que nem todos os números possam ser representados precisamente (por exemplo, 4.2). Por isso, é recomendado não fazer comparações com números reais para igualdade exata (usando `==`, como em `a==4.2`). É melhor verificar se o número está em um intervalo aceitável

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



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 pseudoliguagem  
               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*)
 - ▶ Subsequências 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**

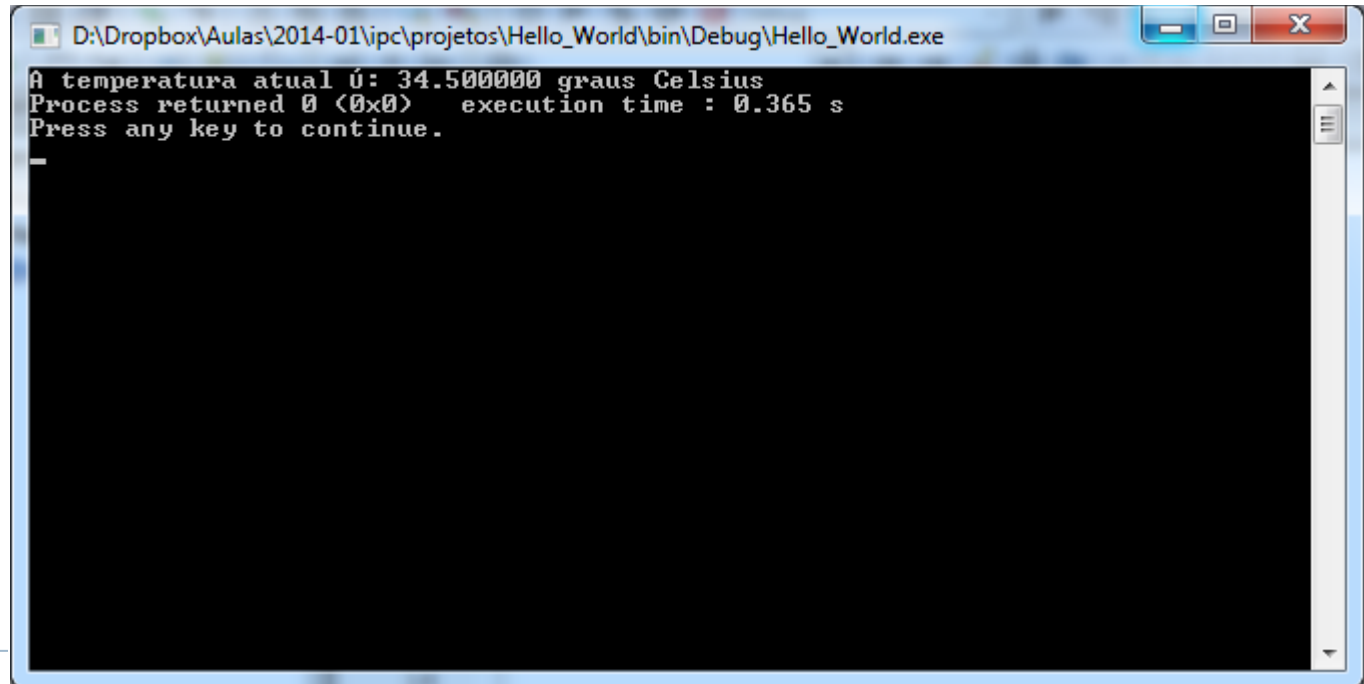
```
float TemperaturaAtual = 34.5;  
printf("A temperatura atual é: ");  
printf("%f", TemperaturaAtual);  
printf(" graus Celsius");
```



Comando de saída

► printf()

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

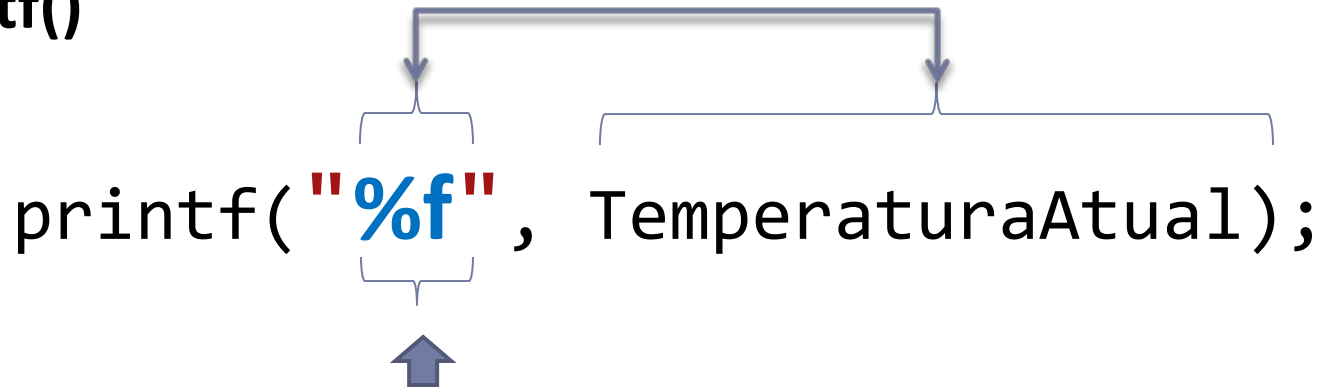


The screenshot shows a Windows command prompt window titled "D:\Dropbox\Aulas\2014-01\ipc\projetos\Hello_World\bin\Debug\Hello_World.exe". The output of the program is displayed as follows:

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

Comando de saída

► printf()



The diagram illustrates the structure of the `printf()` command. It shows the code `printf("%f", TemperaturaAtual);`. A bracket under the opening double quote and another bracket under the closing double quote are connected by a horizontal line with downward-pointing arrows at each end, indicating that the text between the quotes is the format specifier. A separate bracket is placed under the text `TemperaturaAtual`, which is the argument being passed to the function.

```
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 format string of the printf function and pointing to its arguments. The first arrow starts at the first '%f' and points to the variable 'peso'. The second arrow starts at the second '%f' and points to the variable 'altura'. The third arrow starts at the third '%f' and points to the variable 'IMC'.

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 entrada

- ▶ Comandos de entrada servem para obtermos informações dos dispositivos de entrada ligados ao computador
- ▶ Nos algoritmos indicamos os comando de entrada como “leia” (*read*) um determinado valor
- ▶ 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



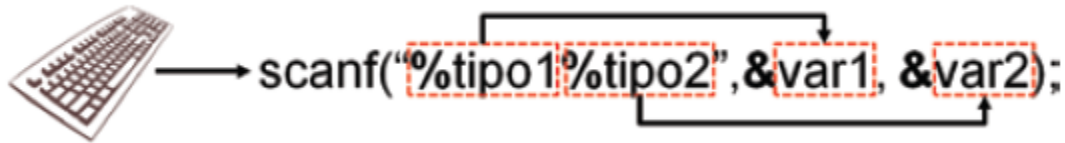
Comando de entrada

- ▶ 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 decimais
 - ▶ %f – leitura de float (número real)
 - ▶ %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



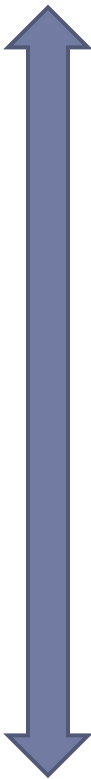
Menor Precedência

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



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 (*escape sequences*).

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"
-------	---



Constantes

▶ Caractere

- ▶ Um caractere constante é na verdade um valor inteiro
- ▶ Ele é representado por aspas simples
 - ▶ Exemplo: 'a'; '?'; 'z' no exemplo abaixo
 - ▶ `char a = 'z';`
- ▶ Ao atribuir esse valor a um 'char' é feita

▶ Numero Inteiro

- ▶ Se está dentro da precisão do `int`, é um inteiro
 - ▶ Exemplo: 122; 145; 12 no exemplo abaixo
 - ▶ `qte = 12*unidades;`



Constantes

▶ Numero Inteiro

- ▶ Se não está dentro da precisão do `int`, é um `long` (ou outro tipo de maior precisão)
 - ▶ Exemplo: 5000000000;
 - ▶ `qte = 12*unidades;`
- ▶ Constante `long` também pode ser indicada pelo sufixo letra `l` ou `L` após o número inteiro
 - ▶ Exemplo: `1l`; `1500L`
- ▶ Constante `unsigned int` usa-se o sufixo `u` ou `U`
 - ▶ Exemplo: `10U`
- ▶ Constante `unsigned long`, use-se `U` e `L`
 - ▶ Exemplo: `194UL`
 - ▶ `unsigned long i = k*100ul;`



Constantes

▶ Número real

- ▶ Constante que utiliza casas decimais são consideradas double
 - ▶ 3.1415; 1.6180; 1e10; 2.1e-30
- ▶ Caso necessite de uma constante float, use o sufixo f ou F (tem que ter a casa decimal!)
 - ▶ `float f,f2;`
 - ▶ `f2 = 10.0f*f;`
- ▶ Use o sufixo l ou L para especificar long double (no printf/scanf use %Lf), L maiúsculo
- ▶ `long double k = raio*10.8L;`



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;`

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

- ▶ `x << 2`

- ▶ `(x*4)`

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

- ▶ `x >> 2`

- ▶ `(x/4)`

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 seqüência 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
```





Tópicos Adicionais

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



Sintaxe

- ▶ `int scanf(char *format, ...)`
 - ▶ Lê os dados da entrada padrão (*stdin* – standard input)
 - ▶ Interpreta os dados de acordo com o indicado em *format*
 - ▶ Armazena os resultados na lista de argumentos (representada pelo “...” da sintaxe)
 - ▶ Todo argumento deve ser um ponteiro (tema de outra aula)
 - ▶ Retorna o número de itens da lista de argumentos corretamente identificados e atribuídos
 - ▶ Pode ser usado para saber quantos itens foram encontrados



Sintaxe

- ▶ `int scanf(char *format, ...)`
 - ▶ `char *format`
 - ▶ É uma *string* que contém as especificações para conversões, podendo conter
 - ▶ Caracteres de *espaços em branco* são todos ignorados
 - Os “caracteres de *espaços em branco*” são o espaço (), a tabulação (\t) e nova linha(\n)
 - A exceção ocorre com o %c, que lê esses caracteres
 - ▶ Caracteres comuns (letras, números, símbolos – exceto %), em que é esperado casar (igualar) com o próximo caractere diferente do espaço em branco do fluxo de entrada (*input stream*).
 - ▶ Caractere % seguido do caractere de conversão



-
- ▶ `%[*][width][length]specifier`

<http://www.cplusplus.com/reference/cstdio/scanf/>




```

#include <stdio.h>

int main()
{
    char letra1,letra2;
    printf("Lendo char com scanf !\n");

    printf("Informe o primeiro char: ");
    scanf("%c",&letra1);

    printf("Informe o segundo char: ");
    scanf("%c",&letra2);

    printf("\n\nObserve que a segunda letra nao foi lida.");
    printf("\nIsso ocorre pois a tecla ENTER (\\n) foi");
    printf(" considerada um caractere\n");
    printf("\nMostrando os valores lidos\n");

    printf("\nLetra 1: >>>%c<<<", letra1);
    printf("\nLetra 2: >>>%c<<<", letra2);

    printf("\nObserve um foi inserido um ENTER ");
    printf("ao mostrar a letra 2 >>> <<<, \n");
    printf("provando que foi lido o ENTER (\\n) \n\n");

    return 0;
}

```

```

clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
❯ clang-7 -pthread -lm -o main main.c
❯ ./main
Lendo char com scanf !
Informe o primeiro char: 4
Informe o segundo char:

Observe que a segunda letra nao foi lida.
Isso ocorre pois a tecla ENTER (\\n) foi considerada um caractere

Mostrando os valores lidos

Letra 1: >>>4<<<
Letra 2: >>>
<<<
Observe um foi inserido um ENTER ao mostrar a letra 2 >>> <<<,
provando que foi lido o ENTER (\\n)

❯

```

Explicando

- ▶ Quando fazemos um **scanf**, realizamos um pedido ao sistema operacional para que busque uma informação do dispositivo de entrada, no caso o teclado
- ▶ O que é digitado não é imediatamente passado ao programa, mas fica armazenado em um *buffer* no sistema operacional (SO). Um *buffer* é um espaço temporário de memória.
- ▶ Isso pode ser notado quando pressionamos o *backspace* ao digitar – em nosso código em C não foi programado que quando o *backspace* é pressionado a letra anterior deve ser apagada. Outro software é quem faz essa tarefa.



-
- ▶ Ao pressionar ENTER, o SO finalmente envia o que foi digitado ao seu programa, para que este processe a entrada.
 - ▶ Note que o caractere ENTER também é enviado ao programa

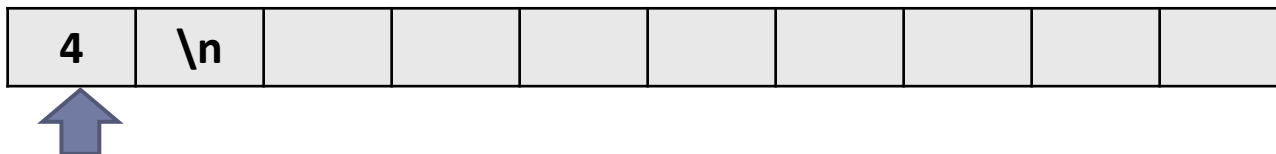
Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER

4	\n								
---	----	--	--	--	--	--	--	--	--

Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER



- ▶ O Buffer é então consumido pelo scanf

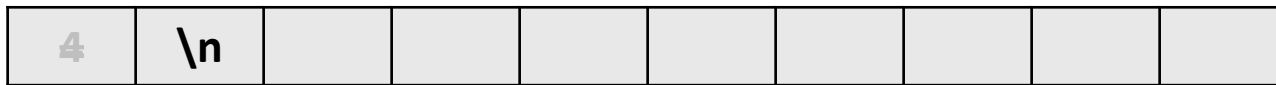
```
printf("Informe o primeiro char: ");  
scanf("%c",&letra1);
```

- ▶ %c faz a leitura de um caractere. Ele joga o valor '4' que está no BUFFER para a variável **letra1**



Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER



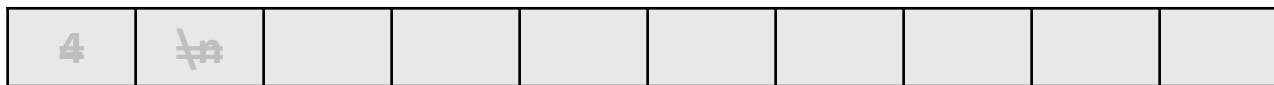
- ▶ Em seguida, como o buffer ainda não está vazio, o próximo scanf é executado

```
printf("Informe o segundo char: ");  
scanf("%c",&letra2);
```
- ▶ %c faz a leitura de um caractere. Ele joga o valor '\n' que está no BUFFER para a variável **letra2**



Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER



► O programa termina!

```
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
❯ clang-7 -pthread -lm -o main main.c
❯ ./main
Lendo char com scanf !
Informe o primeiro char: 4
Informe o segundo char:

Observe que a segunda letra nao foi lida.
Isso ocorre pois a tecla ENTER (\n) foi considerada um caractere

Mostrando os valores lidos

Letra 1: >>>4<<<
Letra 2: >>>
<<<
Observe um foi inserido um ENTER ao mostrar a letra 2 >>> <<<,
provando que foi lido o ENTER (\n)

❯
```

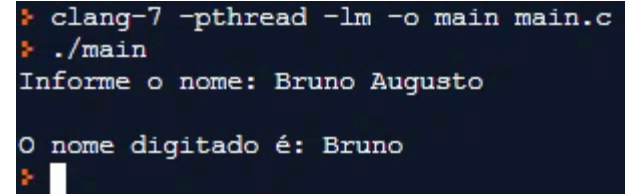
▶ Como solucionar o problema?

- ▶ Temos que apertar “ENTER”



scanf com %s

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
}
```



```
* clang-7 -pthread -lm -o main main.c  
* ./main  
Informe o nome: Bruno Augusto  
O nome digitado é: Bruno  
*
```

- ▶ *scanf* usa espaço em branco (, \t, \n) como separador
 - ▶ Se dois nomes são digitados, somente um nome é armazenado
 - ▶ Observe que no exemplo só “Bruno” foi lido
 - ▶ Mas o outro (“Augusto”) permanece no buffer
 - ▶ O terminador de string ‘\0’ é inserido no vetor ‘nome’
-

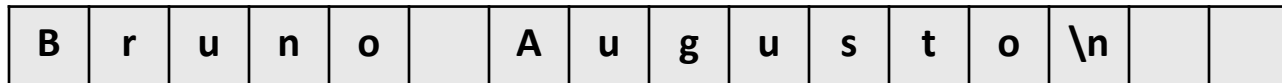


scanf com %s

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
}
```

```
❖ clang-7 -pthread -lm -o main main.c  
❖ ./main  
Informe o nome: Bruno Augusto  
  
O nome digitado é: Bruno  
❖
```

BUFFER



nome

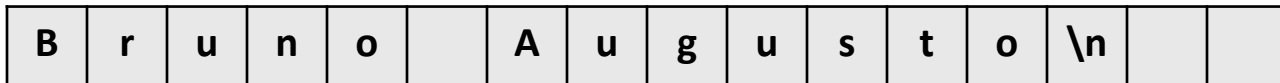


scanf com %s

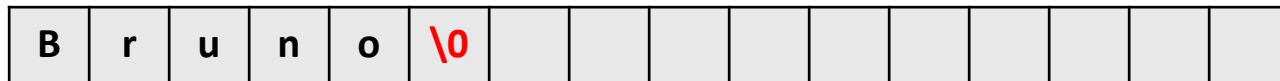
```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
}
```

```
❖ clang-7 -pthread -lm -o main main.c  
❖ ./main  
Informe o nome: Bruno Augusto  
  
O nome digitado é: Bruno  
❖
```

BUFFER



nome

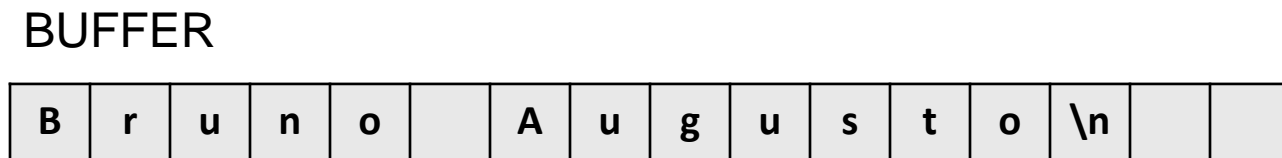


scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 0;
c ← ' ' (espaço)



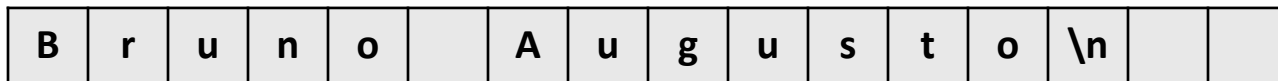
scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 1;
c ← 'A'

BUFFER

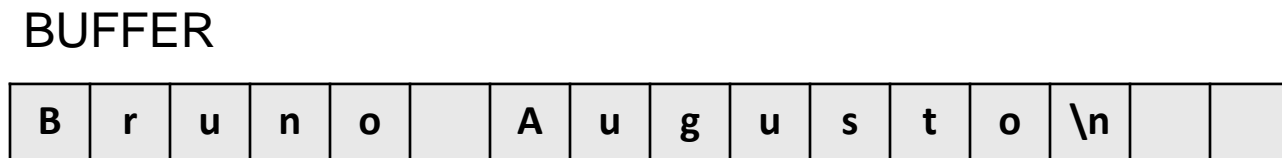


scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 2;
c ← 'u'



scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 8;
c ← '\n'

BUFFER

B	r	u	n	o		A	u	g	u	s	t	o	\n		
---	---	---	---	---	--	---	---	---	---	---	---	---	----	--	--



scanf com %s

- Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

BUFFER

B	r	u	n	o		A	u	g	u	s	t	o	\n		
---	---	---	---	---	--	---	---	---	---	---	---	---	----	--	--

i = 9;
Fim do buffer,
scanf chama o SO
para nova entrada



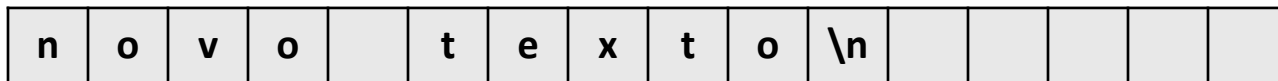
scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 10;
C ← 'n'

BUFFER



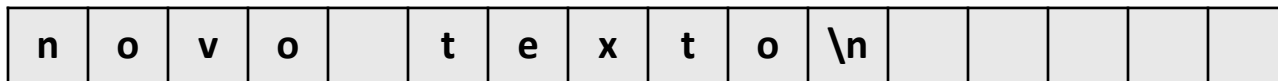
scanf com %s

- ▶ Se novos comandos *scanf* são chamados, o buffer será consumido até o final

```
int main(void) {  
    char nome[100];  
  
    printf("Informe o nome: ");  
    scanf("%s", nome);  
  
    printf("\nO nome digitado é: %s\n", nome);  
  
    char c;  
    // chamando o scanf 20x  
    for (int i = 0; i < 20; i++){  
        scanf("%c", &c);  
        printf("Próximo caractere no buffer: %c\n", c);  
    }  
}
```

i = 11;
C ← 'o'

BUFFER



```

int main(void) {
    char nome[100];

    printf("Informe o nome: ");
    scanf("%s", nome);

    printf("\nO nome digitado é: %s\n", nome);

    char c;
    // chamando o scanf 20x
    for (int i = 0; i < 20; i++){
        scanf("%c", &c);
        printf("Próximo caractere no buffer: %c\n", c);
    }
}

```

```

> clang-7 -pthread -lm -o main main.c
> ./main
Informe o nome: Bruno Augusto

O nome digitado é: Bruno
Próximo caractere no buffer:
Próximo caractere no buffer: A
Próximo caractere no buffer: u
Próximo caractere no buffer: g
Próximo caractere no buffer: u
Próximo caractere no buffer: s
Próximo caractere no buffer: t
Próximo caractere no buffer: o
Próximo caractere no buffer:

novo texto
Próximo caractere no buffer: n
Próximo caractere no buffer: o
Próximo caractere no buffer: v
Próximo caractere no buffer: o
Próximo caractere no buffer:
Próximo caractere no buffer: t
Próximo caractere no buffer: e
Próximo caractere no buffer: x
Próximo caractere no buffer: t
Próximo caractere no buffer: o
Próximo caractere no buffer:

> 

```

Voltando ao scanf com %c

- ▶ Vimos que dois comandos scanf com %c seguidos faz com que o segundo ‘consume’ do buffer o ‘\n’, não permitindo o usuário digitar um valor
- ▶ Mais de uma solução pode ser usada para resolver este problema
- ▶ A ideia é consumir o ‘\n’ ou qualquer outro caractere em branco antes de chamar o próximo *scanf*. Isso pode ser feito da seguintes formas
 - ▶ Solução 1: Adicionando um espaço em branco antes do “%c”
 - ▶ Solução 2: Realizando a chamada do comando getchar()



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

```
int main()
{
    char letra1,letra2;
    printf("Lendo char com scanf !\n");

    printf("Informe o primeiro char: ");
    scanf("%c",&letra1);

    printf("Informe o segundo char: ");
    scanf("%c",&letra2);

    printf("\n\nObserve que a segunda letra nao foi lida.");
    printf("\nIsso ocorre pois a tecla ENTER (\\n) foi");
    printf(" considerada um caractere\n");
    printf("\nMostrando os valores lidos\n");

    printf("\nLetra 1: >>>%c<<<", letra1);
    printf("\nLetra 2: >>>%c<<<", letra2);

    printf("\nObserve um foi inserido um ENTER ");
    printf("ao mostrar a letra 2 >>> <<<, \n");
    printf("provando que foi lido o ENTER (\\n) \n\n");

    return 0;
}
```

Solução 1:
Adicionando um espaço em
branco antes do %c



```
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
> clang-7 -pthread -lm -o main main.c
> ./main
Lendo char com scanf !
Informe o primeiro char: 4
Informe o segundo char:

Observe que a segunda letra nao foi lida.
Isso ocorre pois a tecla ENTER (\n) foi considerada um caractere

Mostrando os valores lidos

Letra 1: >>>4<<<
Letra 2: >>>
<<<
Observe um foi inserido um ENTER ao mostrar a letra 2 >>> <<<,
provando que foi lido o ENTER (\n)

> |
```

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

```
int main()
{
    char letra1,letra2;
    printf("Lendo char com scanf !\n");

    printf("Informe o primeiro char: ");
    scanf("%c",&letra1);

    printf("Informe o segundo char: ");
    scanf(" %c",&letra2);

    printf("\n\nObserve que a segunda letra nao foi lida.");
    printf("\nIsso ocorre pois a tecla ENTER (\\n) foi");
    printf("\nconsiderada um caractere\n");
    printf("\nMostrando os valores lidos\n");

    printf("\n\nObserve um foi inserido um ENTER ");
    printf("ao mostrar a letra 2 >>> <<<, \n");
    printf("provando que foi lido o ENTER (\\n) \n\n");

    return 0;
}
```

Solução 1:
Adicionando um espaço em branco antes do %c

DEPOIS

Foi adicionado um pequeno espaço em branco

```
➤ ./main
Lendo char com scanf !
Informe o primeiro char: 4
Informe o segundo char: 5

Observe que agora a segunda letra foi lida.
Isso ocorre pois o espaço em branco indica ao
scanf que ele deve consumir espaços, tabulações e \n

Mostrando os valores lidos

Letra 1: >>>4<<<
Letra 2: >>>5<<<

➤ □
```

```
#include <stdio.h>

int main()
{
    char letra1,letra2;
    printf("Lendo char com scanf !\n");

    printf("Informe o primeiro char: ");
    scanf("%c",&letra1);

    printf("Informe o segundo char: ");
    scanf(" %c",&letra2);

    printf("\n\nObserve que a segunda letra nao foi lida.");
    printf("\nIsso ocorre pois a tecla ENTER (\\n) foi");
    printf(" considerada um caractere\n");
    printf("\nMostrando os valores lidos\n");

    printf("\nLetra 1: >>>%c<<<", letra1);
    printf("\nLetra 2: >>>%c<<<", letra2);

    printf("\nObserve um foi inserido um ENTER ");
    printf("ao mostrar a letra 2 >>> <<<, \n");
    printf("provando que foi lido o ENTER (\\n) \n\n");

    return 0;
}
```

Solução 1:
Adicionando um espaço em
branco antes do %c

Lendo char com scanf !
Informe o primeiro char: 4↵

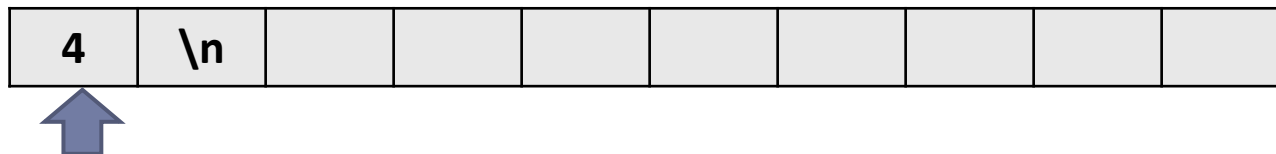
BUFFER



Solução 1:
Adicionando um espaço em branco antes do %c

Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER



- ▶ O Buffer é então consumido pelo scanf

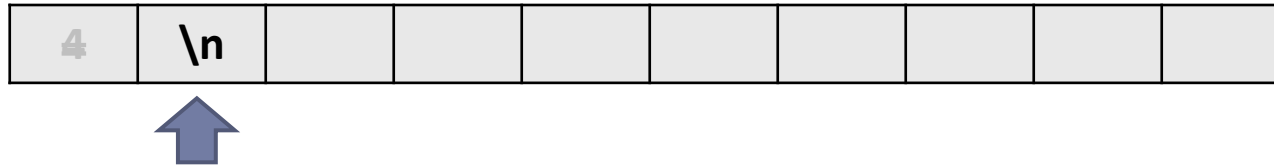
```
printf("Informe o primeiro char: ");  
scanf("%c",&letra1);
```

- ▶ %c faz a leitura de um caractere. Ele joga o valor '4' que está no BUFFER para a variável **letra1**

Solução 1:

Adicionando um espaço em branco antes do %c

BUFFER



- ▶ Em seguida, como o buffer ainda não está vazio, o próximo scanf é executado

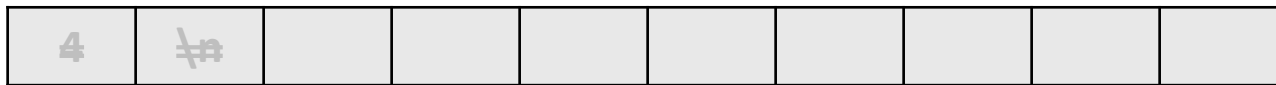
```
printf("Informe o segundo char: ");  
scanf(" %c", &letra2);
```

- ▶ O espaço em branco no início da string do *scanf* faz com que o scanf consuma o '\n' do buffer, esvaziando o buffer
- ▶ `scanf(" %c")` faz o *scanf* consumir todo o buffer até encontrar um caractere diferente de espaço em branco (' '\t, \n)

Solução 1:
Adicionando um espaço em branco antes do %c

Lendo char com scanf !
Informe o primeiro char: 4↵

BUFFER



- ▶ Ainda falta o *scanf* processar o %c, mas como o buffer já esvaziou, uma nova chamada ao SO é feita para que o usuário digite a nova letra

```
printf("Informe o segundo char: ");  
scanf(" %c",&letra2);
```

Solução 1:

Adicionando um espaço em branco antes do %c

- ▶ Por que então não podemos colocar o espaço em branco depois do %c no primeiro scanf?

```
char letra1, letra2;  
printf("Lendo char com scanf !\n");  
  
printf("Informe o primeiro char: ");  
scanf("%c", &letra1);  
  
printf("Informe o segundo char: ");  
scanf(" %c", &letra2);
```

- ▶ Se isso for feito, você poderá apertar várias vezes o ENTER, espaços em branco, tabulações que o cursor não vai para a leitura do segundo caractere. Ele só fará essa leitura quando você digitar um símbolo que não seja um caractere em branco e depois pressionar ENTER. Isso ocorre pois o espaço em branco dentro do scanf faz com que o comando scanf seja finalizado somente quando encontrar no buffer um caractere diferente de espaço em branco (' ', \t, \n)

Solução 2: Utilizando o `getchar()`

- ▶ A função `getchar()` tem a seguinte sintaxe
 - ▶ `int` `getchar` (`void`);
- ▶ Essa função lê um (somente um!) caractere do buffer
- ▶ Ela retorna o caractere lido ou EOF (por isso o retorno é inteiro (`int`) e não é caractere)
 - ▶ EOF geralmente é o valor -1
 - ▶ Pode-se fazer um *cast* para `unsigned char` para obter o caractere
- ▶ Somente um caractere é lido, por essa razão pode não ser uma boa solução (qualquer espaço em branco a mais digitado pode interferir no resultado)

Solução 2: Utilizando o getchar()

```
int main()
{
    char letra1,letra2;
    printf("Lendo char com scanf !\n");

    printf("Informe o primeiro char: ");
    scanf("%c",&letra1);
    int c = getchar();

    printf("Informe o segundo char: ");
    scanf("%c",&letra2);

    printf("\n\nObserve que agora a segunda letra foi lida.");
    printf("\nIsso ocorre pois o getchar consumiu o \\n e \\n");
    printf("esvaziou o buffer");

    printf("\n\nMostrando os valores lidos\n");
    printf("\nLetra 1: >>>%c<<<", letra1);
    printf("\nLetra 2: >>>%c<<<", letra2);

    printf("\nValor de c, lido no getchar (int): %d",c);
    printf("\nValor de c, lido no getchar (char): %c",c);
    printf("\n\n");

    return 0;
}
```



Especificadores de formato %d e %i

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```



Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```



Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16


```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```



Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10



O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);
```

```
    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);
```

```
    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

Quando %i é usado e o usuário digita um número com um zero à esquerda, isso é interpretado como um número octal
 $(010)_8 = 1 \times 8^1 + 0 \times 8^0 = 8$

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);
```

```
    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);
```

```
    printf("O primeiro valor lido é %d\n", val1);
    scanf("%i", &val2);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Isso não ocorreu quando usamos **%d** na primeira leitura. O especificador **%d** indica que o número que será digitado será **decimal**, podendo ter zeros à esquerda que serão ignorados

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10



O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);

    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);

    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16



```
// diferença %d e %i no scanf
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Leitura do primeiro valor: ");
    scanf("%d", &val1);
```

```
    printf("Leitura do segundo valor: ");
    scanf("%i", &val2);
```

Quando %i é usado e o usuário digita um número iniciando com 0x (zero-x), isso é interpretado como um número hexadecimal
 $(0x10)_{16} = 1 \times 16^1 + 0 \times 16^0 = 16$

```
    printf("Leitura do terceiro valor: ");
    scanf("%i", &val3);
```

```
    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);
```

```
    return 0;
```

```
}
```

Leitura do primeiro valor: 010
Leitura do segundo valor: 010
Leitura do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16



```
// diferença das constantes inteiras nas atribuições
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;

    printf("Atribuição do primeiro valor: 10\n");
    val1 = 10;

    printf("Atribuição do segundo valor: 010\n");
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal

    printf("Atribuição do terceiro valor: 0x10\n\n");
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal

    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);

    return 0;
}
```

Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

// diferença das constantes inteiras nas atribuições
`#include <stdio.h>`

```
int main(void) {  
    int val1, val2, val3;  
  
    printf("Atribuição do primeiro valor: 10\n");  
    val1 = 10;  
  
    printf("Atribuição do segundo valor: 010\n");  
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal  
  
    printf("Atribuição do terceiro valor: 0x10\n\n");  
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal  
  
    printf("O primeiro valor lido é %d\n", val1);  
    printf("O segundo valor lido é %d\n", val2);  
    printf("O terceiro valor lido é %d\n", val3);  
  
    return 0;  
}
```



Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença das constantes inteiras nas atribuições  
#include <stdio.h>
```

```
int main(void) {  
    int val1, val2, val3;  
  
    printf("Atribuição do primeiro valor: 10\n");  
    val1 = 10;  
  
    printf("Atribuição do segundo valor: 010\n");  
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal  
  
    printf("Atribuição do terceiro valor: 0x10\n\n");  
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal  
  
    printf("O primeiro valor lido é %d\n", val1);  
    printf("O segundo valor lido é %d\n", val2);  
    printf("O terceiro valor lido é %d\n", val3);  
  
    return 0;  
}
```

Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença das constantes inteiras nas atribuições  
#include <stdio.h>
```

```
int main(void) {  
    int val1, val2, val3;
```

```
    printf("Atribuição do primeiro valor: 10\n");  
    val1 = 10;
```


```
    printf("Atribuição do segundo valor: 010\n");  
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal
```

```
    printf("Atribuição do terceiro valor: 0x10\n\n");  
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal
```

```
    printf("O primeiro valor lido é %d\n", val1);  
    printf("O segundo valor lido é %d\n", val2);  
    printf("O terceiro valor lido é %d\n", val3);
```

```
    return 0;
```

```
}
```



Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença das constantes inteiras nas atribuições
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Atribuição do primeiro valor: 10\n");
    val1 = 10;
```

```
    printf("Atribuição do segundo valor: 010\n");
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal
```

```
    printf("Atribuição do terceiro valor: 0x10\n\n");
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal
```

```
    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);
```

```
    return 0;
```

```
}
```

Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10



O primeiro valor lido é 10
O segundo valor lido é 8
O terceiro valor lido é 16

```
// diferença das constantes inteiras nas atribuições
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Atribuição do primeiro valor: 10\n");
    val1 = 10;
```

```
    printf("Atribuição do segundo valor: 010\n");
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal
```

```
    printf("Atribuição do terceiro valor: 0x10\n\n");
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal
```

```
    printf("O primeiro valor lido é %d\n", val1);
    printf("O segundo valor lido é %d\n", val2);
    printf("O terceiro valor lido é %d\n", val3);
```

```
    return 0;
```

```
}
```

Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10

O segundo valor lido é 8

O terceiro valor lido é 16

```
// diferença das constantes inteiras nas atribuições
#include <stdio.h>
```

```
int main(void) {
    int val1, val2, val3;
```

```
    printf("Atribuição do primeiro valor: 10\n");
    val1 = 10;
```

```
    printf("Atribuição do segundo valor: 010\n");
```

```
    val2 = 010; // iniciar com 0 (zero) indica que o número é Octal
```

```
    printf("Atribuição do terceiro valor: 0x10\n\n");
```

```
    val3 = 0x10; // iniciar com 0x (zero-x) indicar que o número é Hexadecimal
```

```
    printf("O primeiro valor lido é %d\n", val1);
```

```
    printf("O segundo valor lido é %d\n", val2);
```

```
    printf("O terceiro valor lido é %d\n", val3);
```

```
    return 0;
```

```
}
```

Atribuição do primeiro valor: 10
Atribuição do segundo valor: 010
Atribuição do terceiro valor: 0x10

O primeiro valor lido é 10

O segundo valor lido é 8

O terceiro valor lido é 16

Cuidado!!!! Um erro simples que pode passar despercebido. O programador queria atribuir o número 10, mas como colocou um zero à esquerda, o número armazenado foi 8 (10 em octal é 8)



Codificação de caracteres

Prof. Bruno Augusto Nassif Travençolo

Codificação de caracteres

- ▶ Um **conjunto de caracteres** – conjunto de caracteres de uma determinada língua (e.g., a, b, c, ç, é, ó), símbolos (\$, %, &)
- ▶ Um **conjunto de caracteres codificados** (*code points*) é um conjunto de caracteres em que cada caractere recebe um número único. Exemplo:
 - ▶ Letra 'a' – código: 97; (decimal)
 - ▶ Número zero '0' – código 48 (decimal);
- ▶ Usa-se a seguinte representação para falar dos caracteres
 - ▶ 'a' é o caractere número U+0061
 - ▶ '0' é o caractere número U+0030
 - ▶ (note que os números é hexadecimal)

Codificação de caracteres

- ▶ A codificação de caracteres (***character encoding***) indica como um conjunto de caracteres é mapeado em bytes para manipulação por computadores.

Character encoding

► Existem vários!!

Common character encodings [\[edit \]](#)

- **ISO 646**
 - **ASCII**
- **EBCDIC**
 - **CP037**
 - **CP930**
 - **CP1047**
- **ISO 8859:**
 - **ISO 8859-1** Western Europe
 - **ISO 8859-2** Western and Central Europe
 - **ISO 8859-3** Western Europe and South European (Turkish, Maltese plus Esperanto)
 - **ISO 8859-4** Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
 - **ISO 8859-5** Cyrillic alphabet
 - **ISO 8859-6** Arabic
 - **ISO 8859-7** Greek
 - **ISO 8859-8** Hebrew
 - **ISO 8859-9** Western Europe with amended Turkish character set
 - **ISO 8859-10** Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
 - **ISO 8859-11** Thai
 - **ISO 8859-13** Baltic languages plus Polish
 - **ISO 8859-14** Celtic languages (Irish Gaelic, Scottish, Welsh)
 - **ISO 8859-15** Added the Euro sign and other rationalisations to ISO 8859-1
 - **ISO 8859-16** Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- **CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872**
- **MS-Windows character sets:**
 - **Windows-1250** for Central European languages that use Latin script, (Polish, Czech, Slovak, Hungarian, Slovene, Serbian, Croatian, Bosnian, Romanian and Albanian)
 - **Windows-1251** for Cyrillic alphabets
 - **Windows-1252** for Western languages
 - **Windows-1253** for Greek
 - **Windows-1254** for Turkish
 - **Windows-1255** for Hebrew
 - **Windows-1256** for Arabic
 - **Windows-1257** for Baltic languages
 - **Windows-1258** for Vietnamese
- **Mac OS Roman**
- **KOI8-R, KOI8-U, KOI7**
- **MIK**
- **ISCII**
- **TSCII**
- **VISCII**
- **JIS X 0208** is a widely deployed standard for Japanese character encoding that has several encoding forms.
 - **Shift JIS** (Microsoft [Code page 932](#) is a dialect of Shift_JIS)
 - **EUC-JP**
 - **ISO-2022-JP**
- **JIS X 0213** is an extended version of JIS X 0208.
 - **Shift_JIS-2004**
 - **EUC-JIS-2004**
 - **ISO-2022-JP-2004**
- **Chinese Guobiao**
 - **GB 2312**
 - **GBK** (Microsoft [Code page 936](#))
 - **GB 18030**
- **Taiwan Big5** (a more famous variant is Microsoft [Code page 950](#))
 - **Hong Kong HKSCS**
- **Korean**
 - **KS X 1001** is a Korean double-byte character encoding standard
 - **EUC-KR**
 - **ISO-2022-KR**
- **Unicode** (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane')
 - **UTF-8**
 - **UTF-16**
 - **UTF-32**
- **ANSEL** or **ISO/IEC 6937**

► https://en.wikipedia.org/wiki/Character_encoding

Character encoding

- ▶ Comuns por aqui
 - ▶ ASCII - American Standard Code for Information Interchange
 - ▶ CP-850 Latin-1
 - ▶ Windows-1252
 - ▶ Unicode



ASCII

- ▶ American Standard Code for Information Interchange
- ▶ Normalmente chamamos de Tabela ASCII
- ▶ Usa 7-bits, ou seja, consegue representar 128 caracteres.
- ▶ Os primeiros 32 caracteres (de 0 a 31) são chamados de caracteres de controle. O 127 também é controle.
 - ▶ Muitos desses controles já não fazem sentido atualmente (por exemplo, para controle de impressoras)
 - ▶ Outros podem ser usados (NULL - \0; tab - \t; nova linha - \n; \b - *backspace* e até o \a, que emite um sinal sonoro)
- ▶ Os 95 restantes são caracteres são “imprimíveis”
 - ▶ Letras e símbolos: a, b, A, B, &, #

```
for (char i = 33; i <127;i++){
    printf("%3d %c \t",i,i);
    if (i%8==0) printf("\n");
}
```

Tabela ASCII (>32) - Representação Decimal

33 !	34 "	35 #	36 \$	37 %	38 &	39 '	40 (
41)	42 *	43 +	44 ,	45 -	46 .	47 /	48 0
49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8
57 9	58 :	59 ;	60 <	61 =	62 >	63 ?	64 @
65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H
73 I	74 J	75 K	76 L	77 M	78 N	79 O	80 P
81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X
89 Y	90 Z	91 [92 \	93]	94 ^	95 _	96 `
97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h
105 i	106 j	107 k	108 l	109 m	110 n	111 o	112 p
113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x
121 y	122 z	123 {	124	125 }	126 ~		

```
for (char i =33; i <127;i++){
    printf("%X %c \t",i,i);
    if (i%8==0) printf("\n");
}
```

Tabela ASCII (>32) -Representação Hexadecimal

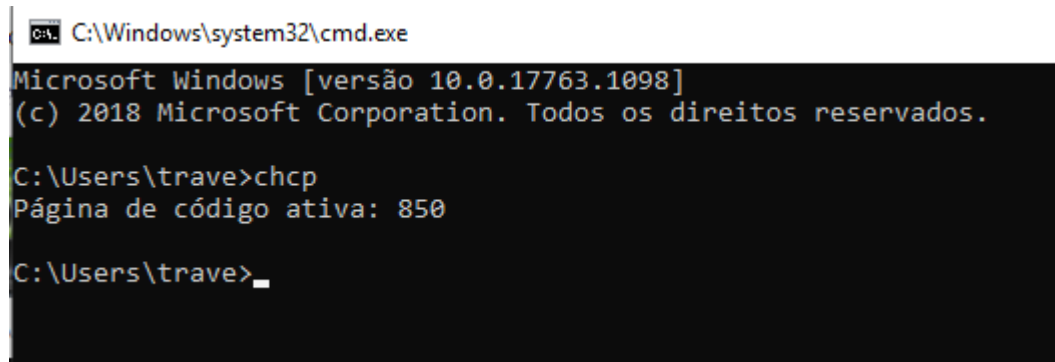
21 !	22 "	23 #	24 \$	25 %	26 &	27 '	28 (
29)	2A *	2B +	2C ,	2D -	2E .	2F /	30 0
31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8
39 9	3A :	3B ;	3C <	3D =	3E >	3F ?	40 @
41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H
49 I	4A J	4B K	4C L	4D M	4E N	4F O	50 P
51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X
59 Y	5A Z	5B [5C \	5D]	5E ^	5F _	60 `
61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h
69 i	6A j	6B k	6C l	6D m	6E n	6F o	70 p
71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x
79 y	7A z	7B {	7C	7D }	7E ~		

Tabela ASCII



Code page 850

- ▶ Usada no DOS (um antigo SO) e ainda presente no shell básico do Windows (cmd)



```
ca. C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.17763.1098]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\trave>chcp
Página de código ativa: 850

C:\Users\trave>_
```

- ▶ 1 byte, dobrando o tamanho da tabela ASCII original
- ▶ Caracteres iniciais (0-127) iguais a da tabela ASCII

Code page 850

```
//somente no console do windows,  
//code page 850  
for (int i =33; i <=255;i++){  
    printf("%3d %c \t",i,i);  
    if (i%8==0) printf("\n");  
}
```

Comando
Windows para
mudar a
codificação:
chcp 850

Página de código ativa: 850

33 !	34 "	35 #	36 \$	37 %	38 &	39 '	40 (
41)	42 *	43 +	44 ,	45 -	46 .	47 /	48 0
49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8
57 9	58 :	59 ;	60 <	61 =	62 >	63 ?	64 @
65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H
73 I	74 J	75 K	76 L	77 M	78 N	79 O	80 P
81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X
89 Y	90 Z	91 [92 \	93]	94 ^	95 _	96 `
97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h
105 i	106 j	107 k	108 l	109 m	110 n	111 o	112 p
113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x
121 y	122 z	123 {	124	125 }	126 ~	127 □	128 Ç
129 ü	130 é	131 â	132 ä	133 à	134 â	135 ç	136 ê
137 ë	138 è	139 ï	140 î	141 ì	142 Ä	143 Å	144 É
145 æ	146 Æ	147 ô	148 ö	149 ò	150 û	151 ù	152 ÿ
153 Ö	154 Ü	155 ø	156 £	157 Ø	158 ×	159 f	160 á
161 í	162 ó	163 ú	164 ñ	165 Ñ	166 ª	167 °	168 ¿
169 ®	170 ¬	171 ½	172 ¼	173 ¡	174 «	175 »	176 ⋮
177 ☒	178 ☒	179	180 †	181 Á	182 Â	183 À	184 ©
185 ¶	186	187 ¶	188 ¶	189 ¢	190 ¥	191 ¶	192 ¤
193 ±	194 ±	195 †	196 —	197 †	198 ã	199 Ã	200 ℓ
201 ₣	202 ₣	203 ₣	204 ₣	205 =	206 ₣	207 ¤	208 ð
209 Ð	210 Ê	211 Ë	212 Ë	213 ¶	214 Î	215 Î	216 Ì
217 ¶	218 ¶	219 ■	220 ■	221 ¶	222 Ì	223 ■	224 Ó
225 ß	226 Ô	227 Ò	228 õ	229 Õ	230 µ	231 þ	232 Þ
233 Ú	234 Ù	235 Ù	236 ý	237 Ý	238 -	239 ´	240
241 ±	242 =	243 ¾	244 ¶	245 §	246 ÷	247 ,	248 °
249 ¨	250 .	251 ¹	252 ³	253 ²	254 ■	255	

Windows-1252 CP-1252 (code page 1252)

- ▶ Utilizado em sistemas Windows para compatibilidade com sistemas DOS
- ▶ 1 byte, dobrando o tamanho da tabela ASCII original
- ▶ Caracteres iniciais (0-127) iguais a da tabela ASCII

Comando
Windows para
mudar a
codificação:
chcp 1252

Página de código ativa: 1252

33 !	34 "	35 #	36 \$	37 %	38 &	39 '	40 (
41)	42 *	43 +	44 ,	45 -	46 .	47 /	48 0
49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8
57 9	58 :	59 ;	60 <	61 =	62 >	63 ?	64 @
65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H
73 I	74 J	75 K	76 L	77 M	78 N	79 O	80 P
81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X
89 Y	90 Z	91 [92 \	93]	94 ^	95 _	96 `
97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h
105 i	106 j	107 k	108 l	109 m	110 n	111 o	112 p
113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x
121 y	122 z	123 {	124	125 }	126 ~	127 □	128 €
129 □	130 ,	131 f	132 „	133 ...	134 †	135 ‡	136 ^
137 ‰	138 Š	139 <	140 Œ	141 □	142 Ž	143 □	144 □
145 ‘	146 ’	147 “	148 ”	149 •	150 –	151 –	152 ~
153 ™	154 š	155 >	156 œ	157 □	158 ž	159 Ÿ	160
161 ¡	162 č	163 £	164 ¨	165 ¥	166 ¡	167 §	168 ¨
169 ©	170 ª	171 «	172 ¬	173	174 ®	175 ¯	176 °
177 ±	178 ≈	179 ≡	180 ´	181 µ	182 ¶	183 ·	184 ¸
185 ¹	186 °	187 »	188 ¼	189 ½	190 ¾	191 ¿	192 Å
193 Á	194 Â	195 Ã	196 Ä	197 Å	198 Æ	199 Ç	200 È
201 É	202 Ê	203 Ë	204 Ì	205 Í	206 Î	207 Ï	208 Ð
209 Ñ	210 Ò	211 Ó	212 Ô	213 Õ	214 Ö	215 ×	216 Ø
217 Ù	218 Ú	219 Û	220 Ü	221 Ý	222 Þ	223 ß	224 à
225 á	226 â	227 ã	228 ä	229 å	230 æ	231 ç	232 è
233 é	234 ê	235 ë	236 ì	237 í	238 î	239 ï	240 ð
241 ñ	242 ò	243 ó	244 ô	245 õ	246 ö	247 ÷	248 ø
249 ù	250 ú	251 û	252 ü	253 ý	254 þ	255 ÿ	

Unicode - UTF-8

- ▶ Unicode é um conjunto de caracteres universal
 - ▶ Possui caracteres para as maiorias das línguas atuais
- ▶ Os primeiros 65.536 (2^{16} – 2 bytes) constituem o *Basic Multilingual Plane (BMP)*, que inclui os caracteres mais comuns de várias línguas
- ▶ Os caracteres suplementares (*supplementary characters*) possuem por volta de um milhão de posições disponíveis para os caracteres
- ▶ O tamanho usado para armazenar os caracteres é variável e não há uma representação direta entre o valor armazenado e o código do caractere

Unicode - UTF-8

- ▶ Utiliza de um até quatro bytes para armazenar o código do caractere.
- ▶ Os primeiros 128 caracteres do Unicode ocupam um byte e são uma correspondência um-para-um com a tabela ASCII
 - ▶ Isso traz uma boa compatibilidade com programas antigos
- ▶ Os próximos 1.920 caracteres precisam de dois bytes e cobrem caracteres de alfabetos latino, grego, cirílico, hebreu, entre outros.
- ▶ Três bytes são usados para os caracteres restantes, e incluem, entre outros, caracteres chineses, japoneses e coreanos.
- ▶ Quatro bytes são usados em outros planos do Unicode, como por exemplo para símbolos de escrita históricos, símbolos matemáticos e emoji (símbolos pictográficos).

Code page 850 vs CodeBlocks (CP-1252)

- ▶ O editor de texto do Codeblocks no Windows, por padrão, vem configurado com a codificação Windows-1252
- ▶ O console do Windows é executado por padrão com code-page 850
- ▶ Assim, toda acentuação digitada no código do CodeBlocks não é mostrada corretamente no console

```
int main(void) {  
    printf("Codificações são incompatíveis ");  
}
```

Saída:

Codificapşes sÒo incompatÝveis

CP 1552	Código decimal	CP 850
ç	231	þ
õ	245	§
ã	227	ò
í	237	ý

Code page 850 vs CodeBlocks (UTF-8)

- ▶ Pode-se alterar a configuração do code blocks para UTF8 (Settings/Editor/Encoding settings)

Configure editor

General settings

Editor settings

Other editor settings

C/C++ Editor settings

Encoding settings

Encoding

Use encoding when opening files: UTF-8

Use this encoding

- ▶ O console do Windows é executado por padrão com code-page 850

```
int main(void) {  
    printf("Codificações são incompatíveis")  
}
```

Saída:

Codifica ıº ıÁes s ıúo incompat ııveis

UTF-8 (2 bytes)	Código decimal	CP 850
ç	195 – 167	ı º
õ	195 – 181	ı Á
ã	195 – 163	ı ú
í	195 – 173	ı i

Code Blocks UTF-8 e CMD UFT-8!

- ▶ E se configurarmos o nosso código em UTF-8 e o *shell* do Windows (cmd) em UTF-8?
- ▶ Inserir comando `system("chcp 65001");`

```
int main(void) {  
    system("chcp 65001");  
    printf("Codificações são compatíveis!");  
}
```

Saída:

```
Codificações são compatíveis!
```

Code Blocks UTF-8 e CMD UFT-8!

- ▶ O problema de usar UTF-8 é que ao utilizar o `scanf` o sistema também irá considerar a codificação UTF-8. Dessa forma, caracteres como ç, á, ê ocuparão dois bytes no buffer.
- ▶ Os comandos `scanf("%c",)` e `getchar()` processam um byte por vez, podendo então ter o comportamento afetado.
- ▶ Dessa forma, o buffer é processado incorretamente dependendo do caractere digitado.



UTF-8

- ▶ Rodando novamente o programa que em que arrumamos o problema do buffer em `scanf("%c",)`, percebe-se que, se em UTF-8, o programa volta a apresentar novamente problemas com leitura do buffer
- ▶ O caractere ç em UTF-8 ocupa 2 bytes

```
➤ ./main
Lendo char com scanf !
Informe o primeiro char: ç
Informe o segundo char:

Observe que agora a segunda letra foi lida.
Isso ocorre pois o espaço em branco indica ao
scanf que ele deve consumir espaços, tabulações e \n

Mostrando os valores lidos

Letra 1: >>>ç<<
Letra 2: >>> <<

➤
```

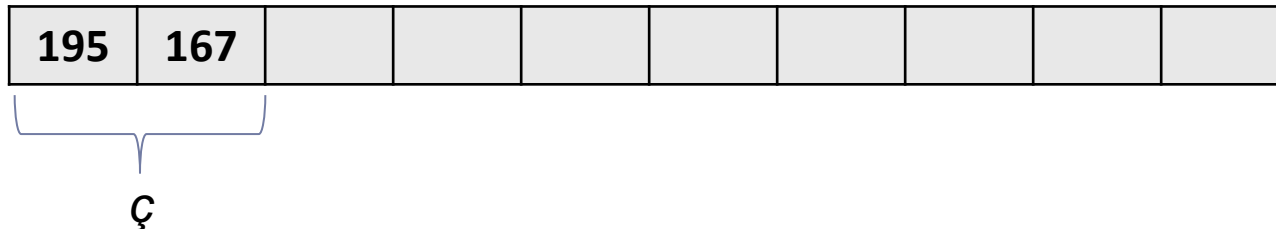

UTF-8

- ▶ O caractere ç em UTF-8 ocupa 2 bytes. Esses dois bytes são consumidos separadamente pelos dois `scanf("%c",)`, `letra1` e `letra2`. Ao mostrar essas variáveis com o `printf` o resultado também não fica correto, pois a codificação é incompatível

```
printf("Informe o primeiro char: ");  
scanf("%c",&letra1);
```

```
printf("Informe o segundo char: ");  
scanf(" %c",&letra2);
```

BUFFER (representação decimal)



UTF-8

- ▶ Neste curso não usaremos essa codificação
- ▶ Caso use, fique atento para utilizar somente os 128 primeiros caracteres, que são idênticos à tabela ASCII e possuem somente um byte
- ▶ Alguns exemplos de uso
- ▶ https://rosettacode.org/wiki/UTF-8_encode_and_decode#C
- ▶ <https://repl.it/@travencolo/utf>





Conversões

Prof. Bruno Augusto Nassif Travençolo

Conversões

- ▶ Alguns operadores podem, dependendo dos seus operandos, causar a conversão do valor de um operando de um tipo para outro
- ▶ Pode ocorrer
 - ▶ Promoção integral
 - ▶ Conversão integral
 - ▶ Inteiros e Pontos flutuantes
 - ▶ Pontos Flutuantes
 - ▶ Conversões aritméticas
 - ▶ Ponteiros e Inteiros (ver livro)
 - ▶ Void (ver livro)
 - ▶ Ponteiros para void (ver livro)

▶ **The C Programming Language**, Second Edition, by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1988).

Promoção Integral

- ▶ `char` e `short int` (com ou sem sinal); tipo enumerado `enum` podem ser usados em expressões em que um `int` pode ser usado. Caso o `int` pode representar todos os valores do tipo original, então o valor é convertido para `int`; caso contrário o valor é convertido para `unsigned int`
- ▶ Esse processo é chamado de promoção integral



```
#include <stdio.h>

int main(void) {
    // integer promotion
    char c[4] = "abc";

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'c'\n");
    printf("Valor de c[0]: %c\n",c[0]);

    printf("Mostrando o conteúdo de a utilizando o especificador 'd'\n");
    printf("Valor de c[0]: %d\n",c[0]);

    printf("o valor de 'a' ASCII é 97!\n");

}
```



%d serve para ler inteiros,
que possuem 4 bytes.



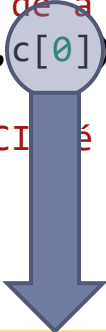
```
#include <stdio.h>

int main(void) {
    // integer promotion
    char c[4] = "abc";

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'c'\n");
    printf("Valor de c[0]: %c\n", c[0]);

    printf("Mostrando o conteúdo de a utilizando o especificador 'd'\n");
    printf("Valor de c[0]: %d\n", c[0]);

    printf("o valor de 'a' em ASCII é 97!\n");
}
```



c[0] é do tipo char, que
possui 1 byte

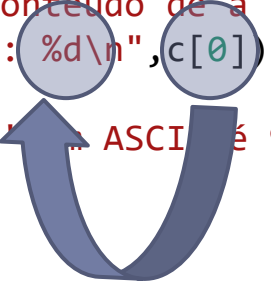
```
#include <stdio.h>

int main(void) {
    // integer promotion
    char c[4] = "abc";

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'c'\n");
    printf("Valor de c[0]: %c\n", c[0]);

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'd'\n");
    printf("Valor de c[0]: %d\n", c[0]);

    printf("o valor de 'a' em ASCII é 97!\n");
}


```

C[0] promovido para int (operação interna, não afetando a variável original) e são lidos os 4 bytes

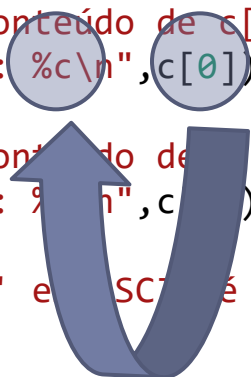

```
#include <stdio.h>

int main(void) {
    // integer promotion
    char c[4] = "abc";

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'c'\n");
    printf("Valor de c[0]: %c\n", c[0]);

    printf("Mostrando o conteúdo de c[0] utilizando o especificador 'd'\n");
    printf("Valor de c[0]: %d\n", c[0]);

    printf("o valor de 'a' em ASCII é 97!\n");
}
```

A diagram illustrating integer promotion. It features a large blue arrow pointing upwards from the variable `c[0]` in the second `printf` statement to the variable `c[0]` in the third `printf` statement. Two small blue circles are placed above the `c[0]` in each statement, connected by the arrow, indicating the transition from a `char` to an `int` during the promotion process.

Uma curiosidade... O mesmo ocorre aqui! O especificador `%c` espera por um valor inteiro, e não por um valor `char`! Ou seja, `c[0]` é promovido para `int` para atender o `%c` e depois convertido para `unsigned char` para ser exibido na tela

Caractere	Tipo do argumento	Exibido como
d,i	int	Notação decimal com sinal
o	unsigned int	Notação octal sem sinal e sem zeros à esquerda
x,X	unsigned int	Notação hexadecimal sem sinal e sem o símbolo 0x ou 0X, sendo abcdef para 0x e ABCDEF para 0X
u	unsigned int	Notação decimal sem sinal
c	int	Um caractere, após conversão para unsigned char
s	char *	Imprime os caracteres até encontrar um '\0' ou até o número de caracteres indicado na precisão
f	double	Notação decimal na forma [-]mmm.ddd, em que o número de dígitos (d) é dado pela precisão (por padrão o valor é 6); 0 remove o separador decimal
e,E	double	Notação científica na forma [-]m.ddddd{e/E}{+/-}xx
g,G	double	Escolhe entre {%e/%E} ou %f (ver documentação)
p	void *	Imprime como ponteiro - depende da implementação
n	int *	O número de caracteres escritos até o momento pela chamada do printf é escrito no argumento (por isso o ponteiro!). Não há conversão de argumentos
%		Não converte argumento, imprime %



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

```
int main(void) {  
    // integer promotion  
    unsigned char n1 = 150;  
    unsigned char n2 = 200;  
    unsigned char soma = n1+n2;  
    unsigned char media = (n1+n2)/2;  
  
    printf("n1=%d, n2=%d, soma = %d (errada!)\n", n1, n2, soma);  
    printf("n1=%d, n2=%d, media = %d\n", n1, n2, media);  
}
```

Promoção integral em operações matemáticas

n1=150, n2=200, soma = 94 (errada!)

n1=150, n2=200, media = 175

Soma não está correta, mas a média está!



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

```
int main(void) {  
    // integer promotion  
    unsigned char n1 = 150;  
    unsigned char n2 = 200;  
    unsigned char soma = n1+n2;  
    unsigned char media = (n1+n2)/2;  
  
    printf("n1=%d, n2=%d, soma = %d (errada!)\n", n1, n2, soma);  
    printf("n1=%d, n2=%d, media = %d\n", n1, n2, media);  
}
```

Promoção integral em operações matemáticas

n1=150, n2=200, soma = 94 (errada!)

n1=150, n2=200, media = 75

Ocorreu a promoção para inteiro para realização de $n1+n2$ (=350). No entanto, o resultado é maior que 255, o máximo suportado por um *unsigned char*, que é o tipo da variável soma



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

```
int main(void) {  
    // integer promotion  
    unsigned char n1 = 150;  
    unsigned char n2 = 200;  
    unsigned char soma = n1+n2;  
    unsigned char media = (n1+n2)/2;  
  
    printf("n1=%d, n2=%d, soma = %d (errada!)\n", n1, n2, soma);  
    printf("n1=%d, n2=%d, media = %d\n", n1, n2, media);  
}
```

Promoção integral em operações matemáticas

n1=150, n2=200, soma = 94 (errada!)

n1=150, n2=200, media = 175

Ocorreu a promoção para inteiro para realização da soma (soma=350). Esse valor em seguida é dividido por 2. Como o resultado da média é menor que 255, a variável média (*unsigned char*) suporta esse valor



Conversões

- ▶ Real para inteiro: parte complementar não inteira é descartada
 - ▶ Não se sabe o comportamento de converter um real negativo para um inteiro sem sinal
- ▶ Inteiro para real: adota-se a representação do número real mais próximo (posterior ou antecessor)
 - ▶ Comportamento indefinido quando o número está fora do intervalo (*out of range*)
- ▶ Real de menor precisão para real de maior ou igual precisão: número inalterado
- ▶ Real de maior para menor precisão: se o número está dentro do intervalo é escolhida a representação mais próxima
 - ▶ Comportamento indefinido se o número está fora do intervalo



Conversões

► Conversões aritméticas (verão antiga)

► Versão do livro do Kernighan & Ritchie

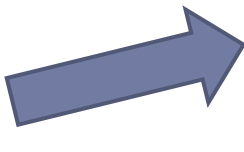
► Sugestão:

<https://en.cppreference.com/w/c/language/conversion>

Se um dos operandos é	O outro é convertido para	
long double	long double	Senão
double	double	Senão
float	float	Senão sabemos que ambos são inteiros
Promoção inteira		
unsigned long int	unsigned long int	
long int	unsigned int	(ou abaixo)
long int	long int	
unsigned int	unsigned int	
int	int	

```
int main(void) {  
    long n1;  
    unsigned long n2,soma;
```

```
    n1 = -5;  
    n2 = 7;  
    soma = n1+n2;
```



Como n2 é *unsigned*, n1 (*signed*) é promovido para *unsigned* (apesar de ter sinal!). O resultado da operação pode ficar comprometido (veja a próxima soma

```
    printf("Valor de n1 (signed long int) : %ld\n",n1);  
    printf("Valor de n2 (unsigned long int): %lu\n",n2);  
    printf("Valor de n1+n2: %lu\n",soma);  
    printf("valor de n1+n2, exibido como se fosse com sinal: %ld\n",soma);
```

```
    n1 = -7;  
    n2 = 5;  
    soma = n1+n2;
```

```
    printf("Valor de n1 (signed long int) : %ld\n",n1);  
    printf("Valor de n2 (unsigned long int): %lu\n",n2);  
    printf("valor de n1+n2: %lu\n",soma);  
    printf("valor de n1+n2, exibido como se fosse com sinal: %ld\n",soma);
```

```
    return 0;  
}
```

```
Valor de n1 (signed long int) : -5  
Valor de n2 (unsigned long int): 7  
Valor de n1+n2: 2  
valor de n1+n2, exibido como se fosse com sinal: 2  
Valor de n1 (signed long int) : -7  
Valor de n2 (unsigned long int): 5  
valor de n1+n2: 18446744073709551614  
valor de n1+n2, exibido como se fosse com sinal: -2
```


Mostrando em bits

-7 + 5 = ?

11111001 (-7) // Notação em complemento de 2
+00000101 (5)

11111110 (-2) // com sinal

11111110 (254) // sem sinal

-5 + 7 = ?

11111011 (-5) // Notação em complemento de 2
+00000111 (7)

00000010 (2) // com e sem sinal

