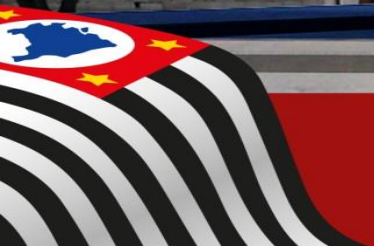


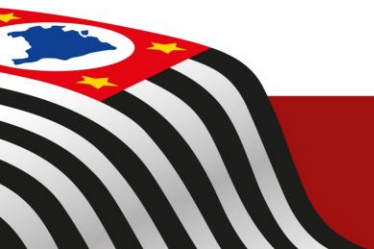
## Linguagem de Programação

- Estrutura de um programa –
  - Tipos de dados, Identificadores, declaração de variáveis, palavras reservadas, operações básicas, comandos de entrada e saída
  - Material: LP\_Aula02



## Tipos de Dados Básicos em C

Há 05 tipos de dados básicos em C: **char**, **int**, **float**, **double** e **void**.  
Outros tipos de dados são baseados nesses tipos.



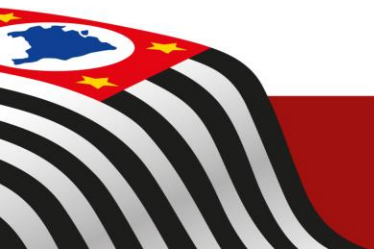
# Tipos de dados definidos no padrão ANSI

Tipo	Tamanho aproximado em bits	Faixa mínima
char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16	-32.767 a 32.767
unsigned int	16	0 a 65.535
signed int	16	O mesmo que int
short int	16	O mesmo que int
unsigned short int	16	0 a 65.535
signed short int	16	O mesmo que short int
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	O mesmo que long int.
unsigned long int	32	0 a 4.294.967.295
float	32	Seis dígitos de precisão
double	64	Dez dígitos de precisão
long double	80	Dez dígitos de precisão



## Modificando os tipos de dados

Segundo Herbert(1997) “um modificador é usado para alterar o significado de um tipo básico para adaptá-lo mais precisamente às necessidades de diversas situações”.

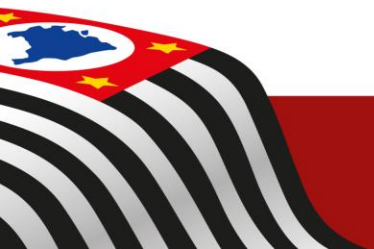


## Lista de Modificadores

Modificadores: **signed, short, long e unsigned.**

Os modificadores podem ser aplicados aos tipos básicos caractere e inteiro.

Contudo long também pode ser aplicado a double.



## Identificadores

Os nomes de variáveis, funções, rótulos e vários outros objetos definidos pelo usuário são chamados de identificadores.

Variam de 1 a vários caracteres.

Exemplos de nomes de identificadores.

### Correto

count

test23

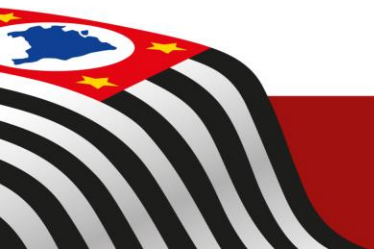
high\_balance

### Incorreto

1count

hi!there

high...balance



## Padrão ANSI para identificadores

- Pelo menos os 6 primeiros caracteres devem ser significativos se o identificador estiver envolvido em um processo externo de linkedição. (Nomes externos)
- Se o identificador não for usado em um processo de linkedição, os 31 primeiros caracteres serão significativos. (Nomes internos)



## Variáveis

- Sempre que desejamos guardar um valor, que não fixo, declaramos variáveis.
- Um variável é um nome que damos a uma determinada posição de memória para conter um valor de um determinado tipo de dados.

### Exemplos:

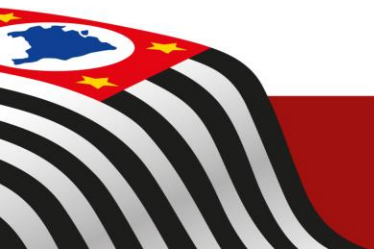
```
int i;                /* i é uma variável do tipo inteiro */  
char chl, novo_char; /* chl e novo_char são vars do tipo char */  
float pi, raio, perimetro;  
double total, k123;
```





## Onde declaramos as variáveis ?

- Variáveis podem ser declaradas em 03 lugares básicos:
  - dentro das funções,
  - na definição dos parâmetros das funções
  - e fora de todas as funções, ou seja, locais, parâmetros formais e variáveis globais.



## Nota sobre a declaração

A declaração de uma variável deve ser feita antes de sua utilização e antes de qualquer instrução.

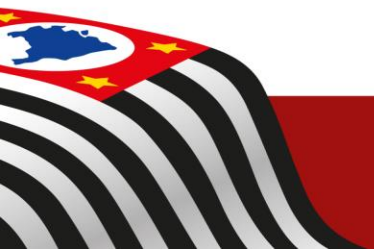
```
main()  
{  
    Declaração de variáveis ←  
  
    Instrução1;  
    Instrução2;  
}
```



## Nome de Variáveis

Conjunto de regras para definição de nomes de variáveis:

- O nome de uma variável deve ser constituído por letras do alfabeto (maiúsculas e minúsculas).
- Maiúsculas e minúsculas representam caracteres diferentes, logo variáveis distintas.
- O primeiro caracteres não pode ser um dígito. Pode ser uma letra, ou o caractere underscore.
- Uma variável não pode ter por nome uma palavra reservada da linguagem C.



## Exemplos

```
int idade;          /* Correto */
int Num_Cliente;    /* Correto */
float alB2c3;       /* Correto */
float 7a2b3c;       /* INCORRETO: primeiro caractere é um dígito */
char float;         /* INCORRETO: utilizou-se uma palavra reservada */
double vinte%;      /* INCORRETO: utilizou-se caractere inadmissível */
char sim?não;       /* INCORRETO: utilizou-se caractere inadmissível */
int _alfa;          /* Correto, mas não aconselhável */
int _123;           /* Correto, mas não aconselhável */
                  /* Notar que o primeiro caractere não é um dígito */
                  /* mas sim o underscore */
char Num, NUM;      /* Correto, pois o C é case sensitive. */
                  /* Será aconselhável ??? */
```

DAMAS (2007, p. 33)



## Atribuição

- Sempre que uma variável é declarada, estamos solicitando ao compilador para reservar um espaço em memória para armazená-la.
  - Esse espaço passará a ser referenciado por esse nome da variável.
- Nota: Quando uma variável é declarada fica sempre com um valor, **o qual é o resultado do estado aleatório dos bits que a constituem.**





## [...] Atribuição

Uma variável poderá ser iniciada com um valor através de uma operação de atribuição.

```
int num = -17;    /* num é declarada do tipo int e automaticamente */  
                  /* iniciada com o valor -17 */  
  
int n1=3, n2=5;   /* n1 e n2 são declaradas e ficam com os valores */  
                  /* 3 e 5 respectivamente */  
  
int a = 10, b, c = -123, d;  
                  /* a e c são automaticamente iniciadas com os  
                  * valores 10 e -123.  
                  * b e d ficam com um valor aleatório ("lixo")  
                  * porque não foram iniciadas.  
                  */
```



## Operações sobre inteiros

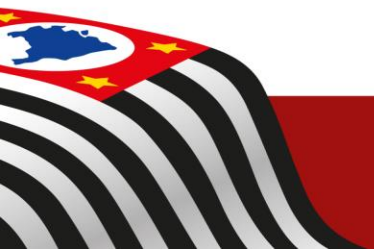
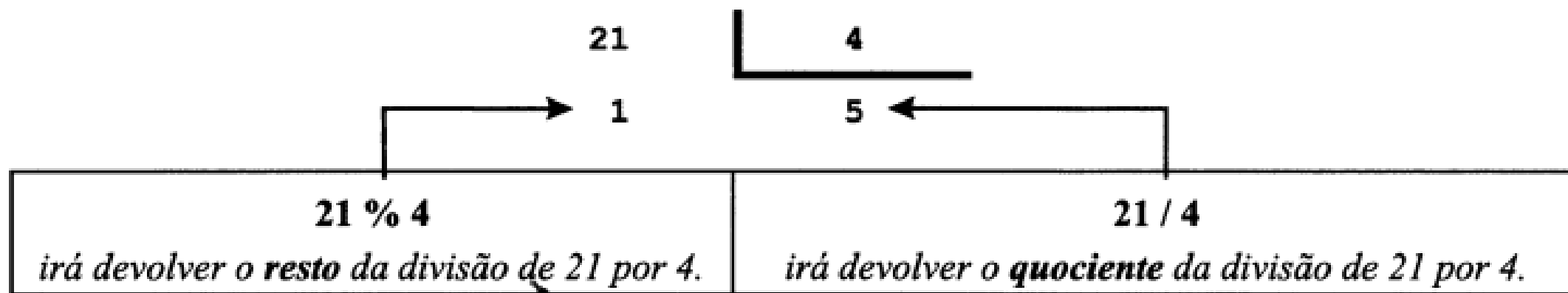
Operação	Descrição	Exemplo	Resultado
+	Soma	$21 + 4$	25
-	Subtração	$21 - 4$	17
*	Multiplicação	$21 * 4$	84
/	Divisão Inteira	$21 / 4$	5
%	Resto da Divisão Inteira ( <b>Módulo</b> )	$21 \% 4$	1

NOTA: Qualquer operação entre inteiros retorna um inteiro.



## [...] Operações sobre inteiros

Assim, da divisão entre 21 e 4 não irá resultar 5,25, como se poderia pensar, uma vez que o resultado de uma operação entre dois inteiros (21 e 4) tem sempre como resultado um inteiro.




## Formato da escrita de um inteiro (%d)

Vamos então colocar o símbolo `%d` no local onde queremos escrever os inteiros:

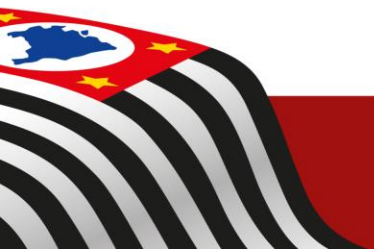
O valor de `num` = `%d` e o valor seguinte = `%d\n`

Falta apenas indicar ao *printf* quais os valores que terá que colocar nos locais assinalados por `%d`.

```
printf("O valor de num =  e o valor seguinte =    \n", num, num+1);
```



Função de escrita: `printf`.



## Função para Leitura de Caracteres

A função usada para a leitura de valores é a função **scanf**.

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int num;
6:
7:     printf("Introduza um N°: ");
8:     scanf("%d", &num);
9:     printf("O N° introduzido foi %d\n", num);
10: }
```

Para ler qualquer variável do tipo int, char, float ou double utiliza-se a função scanf precedendo cada variável com um &.

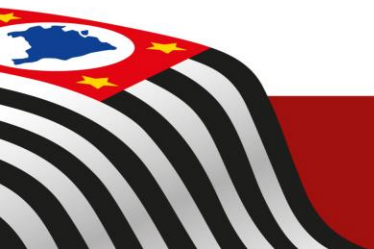


## Inteiros e variações

C disponibiliza uma maneira de saber qual a dimensão de um inteiro, para isso deve-se usar o operador sizeof.

A sintaxe do operador sizeof é

```
sizeof <expressão> ou sizeof ( <tipo> )
```



## Ex: Uso do sizeof

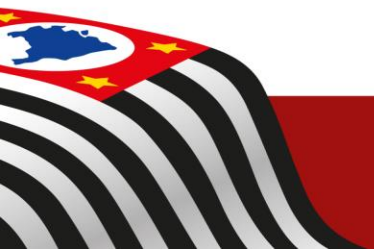
```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("O Tamanho em bytes de um Inteiro = %d\n", sizeof(int));
6: }
```

**Resultado:**  
**O tamanho em bytes de um Inteiro**  
**= 4**  
**(SO de 32 bits)**



## Compile e Teste

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("O Tamanho em bytes de um char    = %d\n", sizeof(char));
6:     printf("O Tamanho em bytes de um int     = %d\n", sizeof(int));
7:     printf("O Tamanho em bytes de um float   = %d\n", sizeof(float));
8:     printf("O Tamanho em bytes de um double = %d\n", sizeof(double));
9: }
```



## Declaração de Inteiros

Podemos usar os seguintes prefixos:

OBS: valores em um SO de 16 bits.

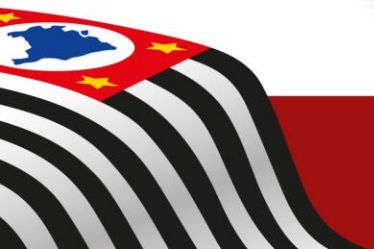
- **short** — Inteiro pequeno (2 *bytes*)
- **long** — Inteiro grande (4 *bytes*)
- **signed** — Inteiro com sinal (nºs negativos e positivos)
- **unsigned** — Inteiro sem sinal (apenas nºs positivos)



## Exemplo com Prefixo para Inteiros

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     short int idade;           /* ou short idade    */
6:     int montante;
7:     long int n_conta;         /* ou long n_conta  */
8:
9:     printf("Qual a Idade: "); scanf("%hd",&idade);
10:    printf("Qual o montante a depositar: "); scanf("%d",&montante);
11:    printf("Qual o n° de conta: "); scanf("%ld",&n_conta);
12:
13:    printf("Uma pessoa de %hd anos depositou $%d na conta %ld\n",
14:           idade, montante, n_conta);
15: }
```

Observe os campos em negrito.



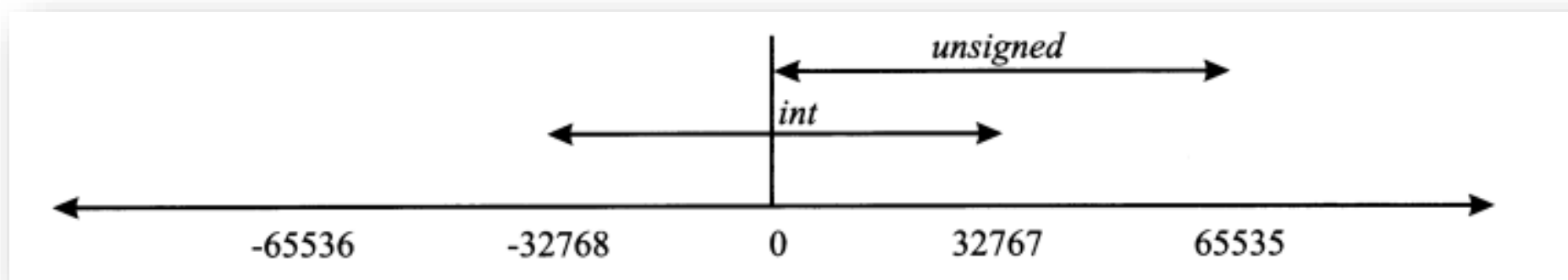


## signed e unsigned

Uma variável do tipo inteiro admite valores positivos e negativos.

Se um inteiro for armazenado em 2 bytes os seus valores podem variar entre -32768 e 32767.

Se desejar valores apenas positivos use o prefixo unsigned.

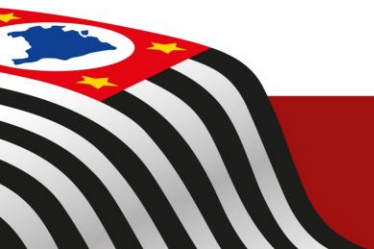


**Atenção1: o formato para leitura e escrita de inteiros sem sinal é %u ao invés de %d**

**Atenção2: - 2147483648 0 2147483647 ( $2^{32}$ ) [Sistema de 32 bits]**

## Limites que um inteiro pode variar

Tipo de Variável	Nº de Bytes	Valor Mínimo	Valor Máximo
int	2	-32 768	32 767
short int	2	-32 768	32 767
long int	4	-2 147 483 648	2 147 483 647
unsigned int	2	0	65 535
unsigned short int	2	0	65 535
unsigned long int	4	0	4 294 967 295



## float ou double

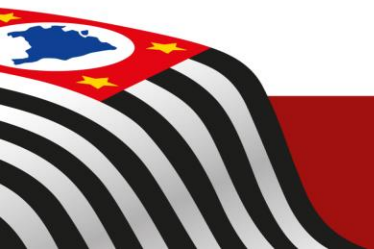
São usadas para armazenar valores numéricos com parte fracionária.

float ocupa 04 bytes. (precisão simples)

double ocupa 08 bytes. (precisão dupla)



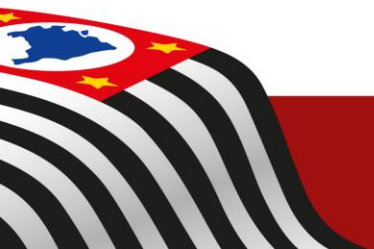
The diagram shows the number 123456.789. The digits 123456 are enclosed in a box and labeled 'Parte inteira' (Integer part). The decimal point '.' is labeled 'Ponto' (Point). The digits 789 are enclosed in a box and labeled 'Parte decimal' (Decimal part).



## Exemplo

1) Escreva um programa que calcule o perímetro e a área de uma circunferência.

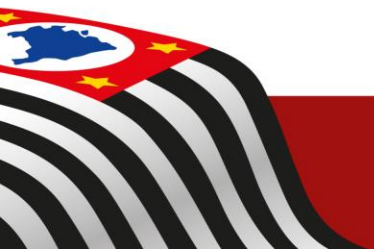
- Área =  $\text{PI} * \text{Raio} * \text{Raio}$
- Perímetro =  $2 * \text{PI} * \text{Raio}$



## Outro Exemplo

Programa que realiza a conversão de toneladas para quilos e gramas escrevendo o resultado em notação tradicional (aaaa.bbbb) e científica (aaa E±bb).

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     float  quilos = 1.0E3; /* Uma tonelada são 1000 quilos */
6:     double gramas = 1.0e6; /* Uma tonelada são 1 000 000 de gramas */
7:     float n_toneladas;
8:
9:     printf("Quantas toneladas comprou: "); scanf("%f", &n_toneladas);
10:    printf("Nº de Quilos = %f = %e\n", n_toneladas * quilos,
           n_toneladas * quilos);
11:    printf("Nº de gramas = %f = %E\n", n_toneladas * gramas,
           n_toneladas * gramas);
12: }
```

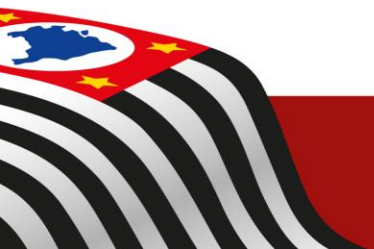




## Explicação de Exemplo Anterior

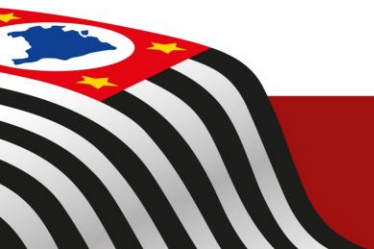
- *No programa anterior, declaramos duas variáveis reais quilos e gramas para conter os valores mil e um milhão, os quais podem ser escritos no formato tradicional (1 000.0 e 1 000 000.0) ou na notação científica (1.0E3 e 1.0E6).*
- *É solicitada ao usuário a introdução de um determinado n° de toneladas.*
- *Em seguida são escritos o n° de quilos e o n° de gramas correspondentes ao n° de toneladas introduzidas (no formato tradicional e no formato científico).*

DAMAS (2007, p. 44)



## Operações sobre Reais

Qualquer operação que inclua um dos operandos do tipo real obtém um resultado real.

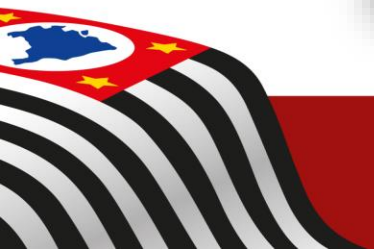


## Operações disponíveis

Operação	Descrição	Exemplo	Resultado
+	Soma	21.3 + 4.1	25.4
-	Subtração	21.7 - 4.8	16.9
*	Multiplicação	21.2 * 4.7	99.64
/	Divisão Real	21.0 / 4.0	5.25
%	Não faz sentido aplicar a reais	n.a.	n.a.

### Exemplos:

21 / 4 → 5 /\* Divisão inteira \*/  
21.0 / 4 → 5.25 /\* Como 21.0 é um real, o valor 4 é alterado para 4.0 \*/  
21 / 4. → 5.25 /\* Como 4. é um real, o valor 21 é alterado para 21.0 \*/  
21.0 / 4.0 → 5.25 /\* Divisão real \*/



## char

O tipo ***char*** permite armazenar UM ÚNICO CARACTERE .

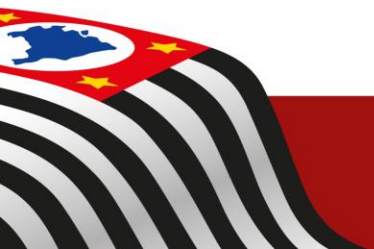
Um ***char*** é sempre armazenado num byte, ou seja, é possível representar 256 (0 a 255) símbolos.

00000000

— Todos os *bits* com 0 (valor 0).

11111111

— Todos os *bits* com 1 (valor 255).



## Declaração de um tipo *char*

A declaração de uma variável do tipo *char* segue a sintaxe já conhecida:

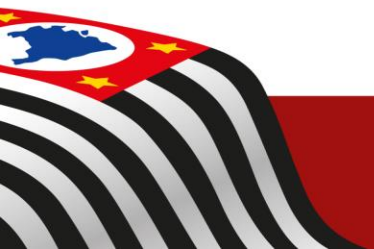
```
char var1, ch, var2;
```

Para realizar uma carga automática de uma variável do tipo *char* deverá ser colocado o caractere a atribuir entre **Aspas simples** (Ex: 'A'), e não entre **Aspas**.

**Nota:** A representação de um caractere individual é sempre realizada entre aspas simples ('A', '2', '\n').

A utilização de aspas para a representação de um caractere "A" é um erro comum, está totalmente incorreta e pode levar a algumas surpresas não muito agradáveis.

DAMAS (2007, p. 46)



## O que faz o programa abaixo ?

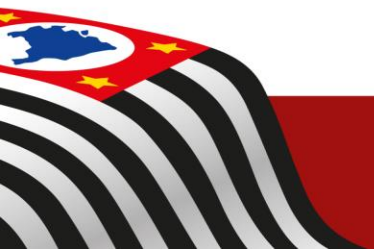
```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("%cello Wo%cld%c", 'H', 'r', '\n');
6: }
```



## Programa que lê um caractere e o imprime

```
1: #include <stdio.h>
2:
3: main()
4: { char ch;
5:   printf("Introduza um Caractere: ");
6:   scanf("%c", &ch);
7:   printf("O caractere introduzido foi '%c'\n",ch);
8: }
```

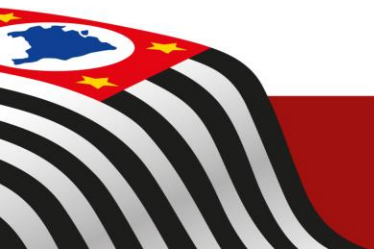
Alternativa ao uso do scanf é a função getchar. Ela é invocada sem qualquer parâmetro, lê um caractere e o devolve como resultado da função, evitando a escrita de parâmetros, formatos, &ch, etc...





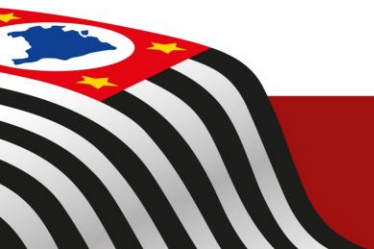
## *getchar()*

```
1: #include <stdio.h>
2:
3: main()
4: { char ch;
5:     printf("Introduza um Caractere: ");
6:     ch = getchar();
7:     printf("O caractere introduzido foi '%c'\n",ch);
8: }
```



## getchar() X scanf()

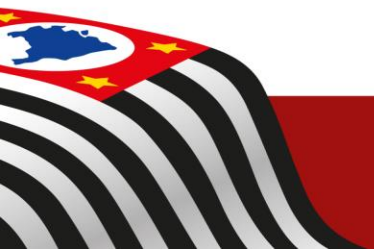
A diferença é que o ***scanf()*** é uma função genérica de leitura enquanto ***getchar()*** é específica para a leitura de caractere.



## Importante

Crie esse programa:

- O programa deve solicitar, através da função scanf, um caractere ao usuário e, em seguida, peça outro. Depois de introduzirmos ambos os caracteres, o programa deve mostrar os dois caracteres lidos entre aspas simples.

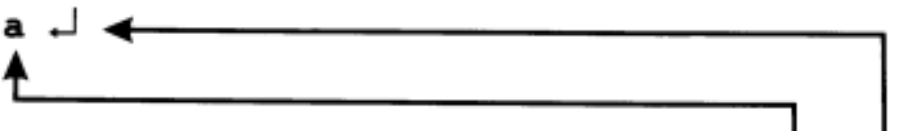


Observe a saída ao executar.

```
1: #include <stdio.h>
2:
3: main()
4: { char ch1, ch2;
5:   printf("Introduza um Caractere: ");
6:   scanf("%c",&ch1);
7:   printf("Introduza outro Caractere: ");
8:   scanf("%c",&ch2);
9:   printf("Os caracteres introduzidos foram '%c' e '%c'\n", ch1,ch2);
10: }
```

**Problemas com buffer.**

Introduza um Caractere: a ↵



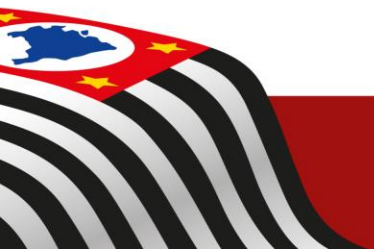
printf("Os caracteres introduzidos foram '%c' e '%c'\n", ch1,ch2);



## Como resolver esse problema

A solução é simples. Use um espaço em branco ' ' imediatamente antes do %c do segundo scanf, mas dentro da string do formato.

```
1: #include <stdio.h>
2:
3: main()
4: { char ch1, ch2;
5:   printf("Introduza um Caractere: ");
6:   scanf("%c",&ch1);
7:   printf("Introduza outro Caractere: ");
8:   scanf(" %c",&ch2);
9:   printf("Os caracteres introduzidos foram '%c' e '%c'\n", ch1,ch2);
10: }
```



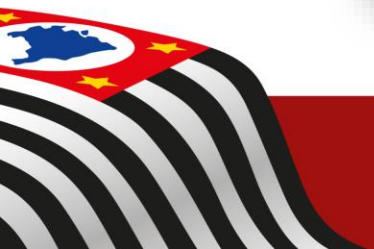
## Caracteres e Inteiros

As operações realizadas com os tipos inteiros podem ser realizadas com os caracteres.

Exemplo formas de colocar o caractere 'A' em uma variável:

### Exemplo:

```
ch = 'A';      /* Formato tradicional */
ch = 65;       /* Caractere cujo código ASCII é 65 */
ch = '\\101';  /* Caractere cujo código ASCII escrito em octal é 101 */
ch = '\\x41';  /* Caractere cujo código ASCII escrito em hexa é 41 */
```



## Execute o seguinte programa:

```
1: #include <stdio.h>
2:
3: main()
4: { char ch;
5:   printf("Introduza um Caractere: ");
6:   scanf("%c",&ch);
7:   printf("O caractere '%c' tem o ASCII nº %d\n", ch , ch);
8: }
```

Nele: Declarou-se um tipo char, fez-se a leitura por meio da **scanf** (poderia ter sido **getchar**). Em seguida escreveu-se o caractere lido (%c) e também o seu código na tabela ASCII, que é armazenado como se fosse um inteiro, embora ocupe só um byte.

```
printf("O caractere '%c' tem o ASCII nº %d\n", ch , ch);
```





# Casting

Sempre que se tem um valor de um determinado tipo e se deseja alterar esse tipo, pode-se indicar o tipo ao qual queremos “promover” esse valor colocando o tipo pretendido entre parênteses antes do valor.

```
printf("O caractere '%c' tem o ASCII nº %d\n", ch, (int) ch);
```



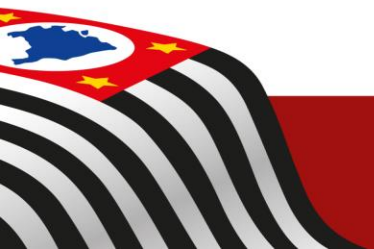
## Execute o seguinte programa:

```
1: #include <stdio.h>
2:
3: main()
4: { int num;
5:   printf("Introduza um Inteiro: ");
6:   scanf("%d",&num);
7:   printf("Foi introduzido %d cujo caractere = '%c'\n",
8:         num, (char) num);
9:   printf("O caractere seguinte = '%c' tem o ASCII nº %d\n",
10:         (char) (num+1) , num+1);
11: }
```

**Responda: Em sua opinião, o que aconteceu ?**



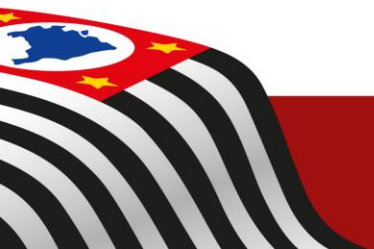
Não se deve realizar a leitura de variáveis de um determinado tipo usando um formato de leitura que não corresponda ao tipo declarado.



## Exercícios Resolvidos

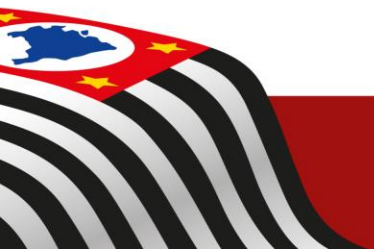
Programa em C que pede ao usuário dois inteiros e apresenta o resultado das operações matemáticas tradicionais.

```
1: #include <stdio.h>
2:
3: main()
4: { int a,b;
5:     printf("Introduza dois Inteiros: ");
6:     scanf("%d%d",&a, &b);
7:     printf("%d + %d = %d\n",a,b,a+b);
8:     printf("%d - %d = %d\n",a,b,a-b);
9:     printf("%d * %d = %d\n",a,b,a*b);
10:    printf("%d / %d = %d\n",a,b,a/b);
11:    printf("%d %% %d = %d\n",a,b,a%b);
12: }
```



Programa que solicita um determinado número de segundos, e em seguida, indica quantas horas, minutos e segundos esse valor representa.

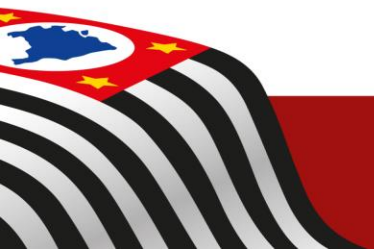
```
3: main()
4: { long int n_segundos;
5:   printf("Introduza o N° de segundos: ");
6:   scanf("%ld",&n_segundos);
7:   printf("Horas    : %d\n", (int) n_segundos/3600);
8:   printf("Minutos  : %d\n", (int) (n_segundos%3600/60));
9:   printf("Segundos: %d\n", (int) n_segundos % 60);
10: }
```



## Mais um

Programa que solicita um determinado número real e exibe sua parte inteira e sua parte fracionária.

```
1: #include <stdio.h>
2:
3: main()
4: { float x;
5:   printf("Introduza um N° real: ");
6:   scanf("%f",&x);
7:   printf("Parte Inteira      : %d\n", (int) x);
8:   printf("Parte Fracionaria: %f\n", x - ((int) x));
9:
10: }
```



# CENTRO PAULA SOUZA

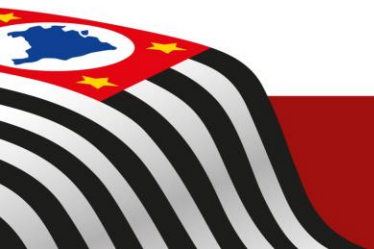
---

## Fatec

Mogi Mirim  
Arthur de Azevedo

Prof. Me. Marcos Roberto de Moraes, o Maromo

# FIM





## Referências Bibliográficas

DAMAS, L. M. D. Linguagem C. LTC, 2007.

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.

