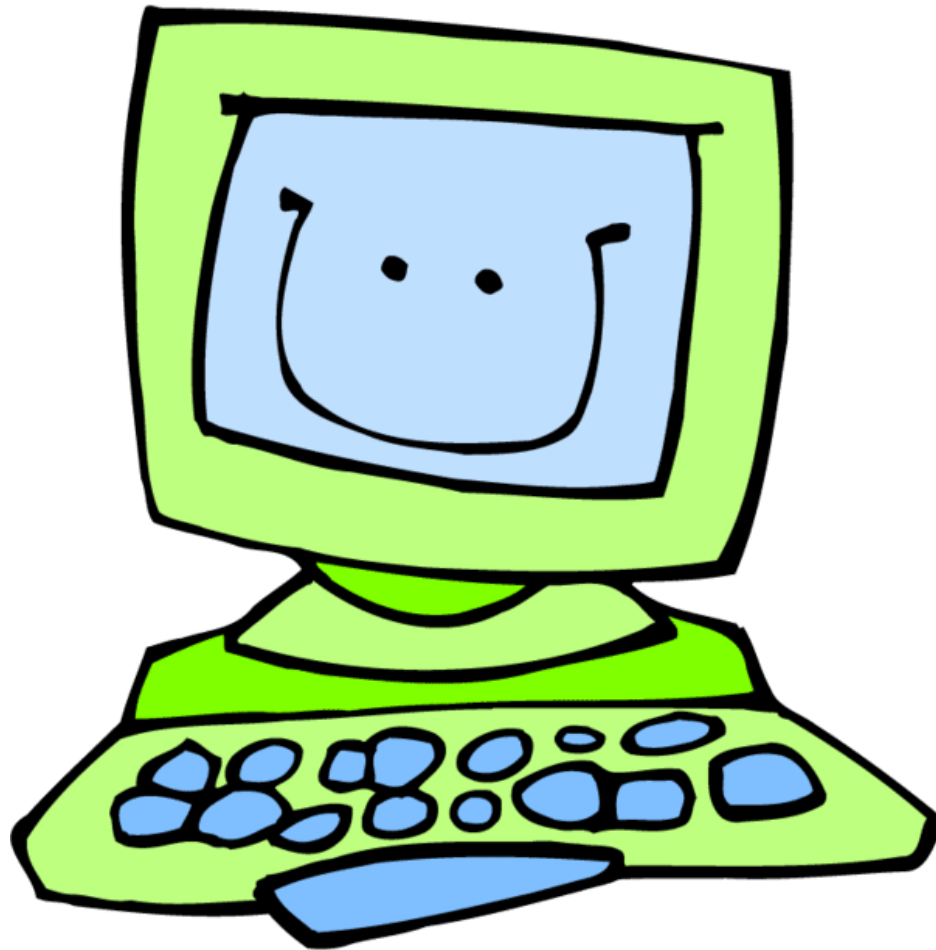


# Introdução à Programação de Computadores

## Aula - Tópico 1

# Por que usar um computador?



# Problema 1

- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis. Dados dois números inteiros positivos  $A$  e  $B$ , determine o **quociente** e o **resto** da divisão de  $A$  por  $B$ .

# Algoritmos Estruturados

- Para resolver o Problema 1, precisamos de um algoritmo:

Sequência finita de instruções que, ao ser executada, chega a uma solução de um problema.

# Algoritmos Estruturados

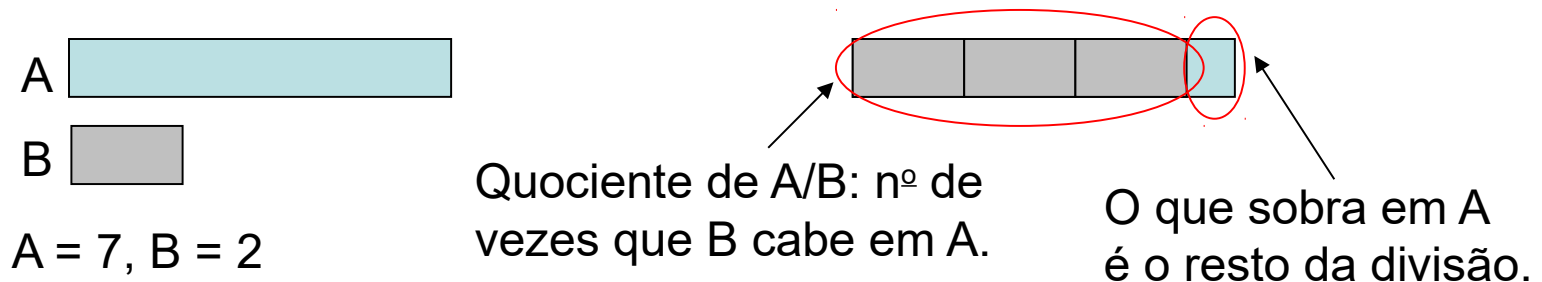
- Para resolver o Problema 1, precisamos de um algoritmo:

Sequência finita de instruções que, ao ser executada, chega a uma solução de um problema.

- Para escrever este algoritmo, podemos usar a seguinte ideia:
  - Representar os números **A** e **B** por retângulos de larguras proporcionais aos seus valores;
  - Verificar quantas vezes **B** cabe em **A**.

# Algoritmos Estruturados

- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis em C. Dados dois números inteiros positivos **A** e **B**, determine o **quociente** e o **resto** da divisão de **A** por **B**.



# Algoritmos Estruturados

- Pode-se escrever este algoritmo como:

```
Sejam A e B os valores dados;  
Atribuir o valor 0 ao quociente (q);  
Enquanto B couber em A:  
{  
    Somar 1 ao valor de q;  
    Subtrair B do valor de A;  
}  
Atribuir o valor final de A ao resto (r).
```

# Algoritmos Estruturados

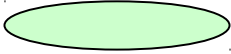

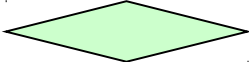
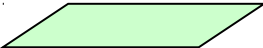

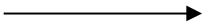
- Pode-se escrever este algoritmo como:

```
Sejam A e B os valores dados;  
Atribuir o valor 0 ao quociente (q);  
Enquanto B <= A:  
{  
    Somar 1 ao valor de q;  
    Subtrair B do valor de A;  
}  
Atribuir o valor final de A ao resto (r).
```



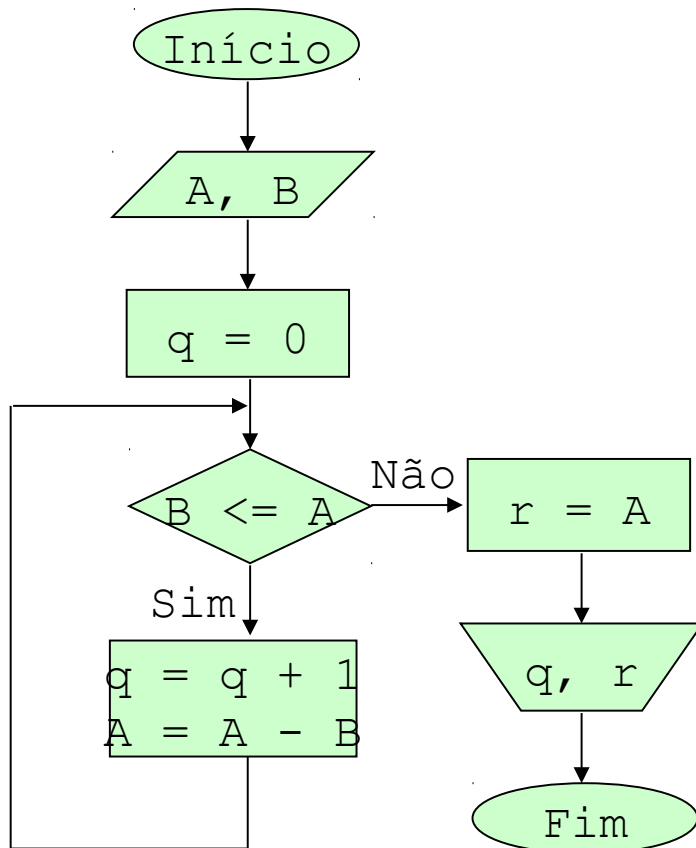
# Fluxograma

- É conveniente representar algoritmos por meio de **fluxogramas** (diagrama de blocos).
- Em um fluxograma, as operações possíveis são representadas por meio de figuras:

Figura	Usada para representar
	Início ou fim.
	Atribuição.
	Condição.
	Leitura de dados.
	Apresentação de resultados.
	Fluxo de execução.

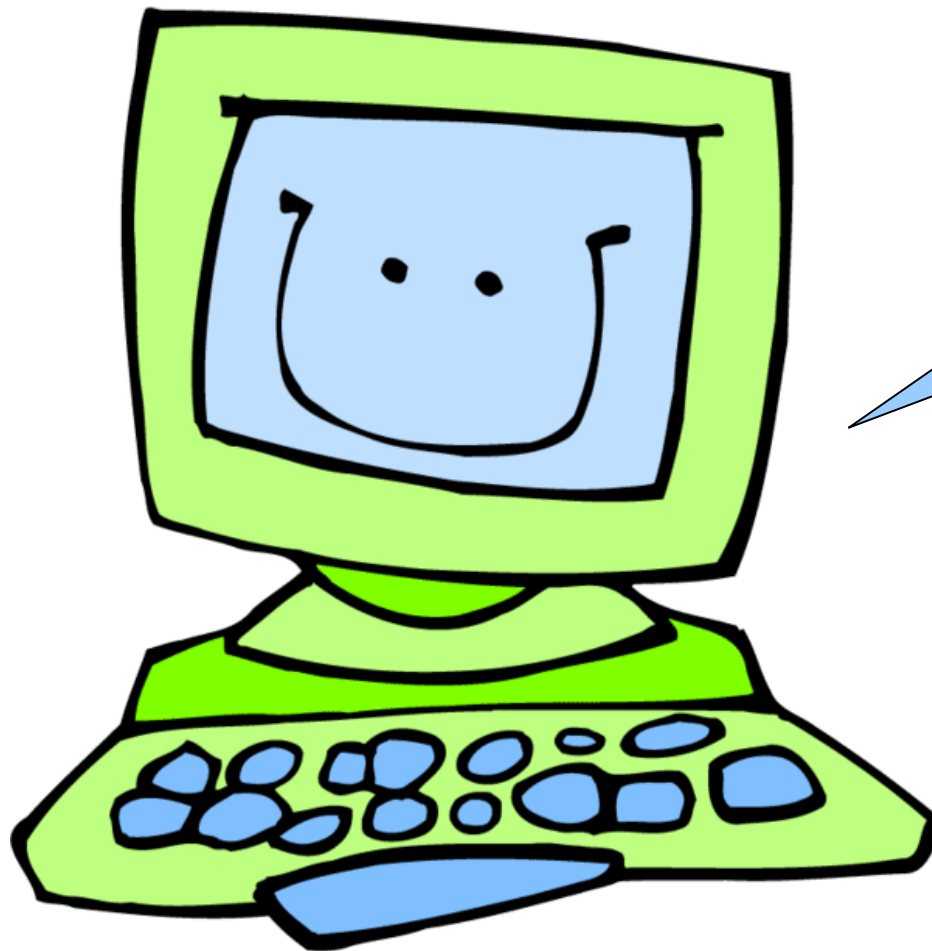
# Fluxograma

- **Exemplo:** o algoritmo para o Problema 1 pode ser representado pelo seguinte fluxograma.



Atenção: observe que um algoritmo não inclui detalhes, tais como declaração de variáveis, mensagens a serem exibidas, etc.

# Como conversar com um computador?



0101001001010100101011  
0010100110101011111010  
0010101101010100001010

# Como conversar com um computador?

- Considere o seguinte problema:
  - Determinar o valor de  $y = \text{seno}(1,5)$ .

# Como conversar com um computador?

- Considere o seguinte problema:
  - Determinar o valor de  $y = \text{seno}(1,5)$ .
  - Escrever um programa:

000101010111010111

001010111010101111

011101011101011100

# Como conversar com um computador?

- Considere o seguinte problema:
  - Determinar o valor de  $y = \text{seno}(1,5)$ .
  - Escrever um programa:

mensagem para o computador:

calcula  $\text{seno}(1,5)$  e armazena em  $y$

imprime\_na\_tela( $y$ )

PAUSA

# Problema 1

- Considere o seguinte problema:
  - Determinar o valor de  $y = \text{seno}(1,5)$ .

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Definições

- Para resolver um problema de computação é preciso escrever um **texto**.
- Este texto, como qualquer outro, obedece **regras de sintaxe**.
- Estas regras são estabelecidas por uma **linguagem de programação**.
- Este texto é conhecido como:

# Programa



# Definições

- Neste curso, será utilizada a **linguagem C**.
- A **linguagem C** é subconjunto da **linguagem C++** e, por isso, geralmente, os **ambientes de programação** da linguagem C são denominados ambientes C/C++.
- Um **ambiente de programação** contém:
  - Editor de programas: viabiliza a escrita do programa.
  - Compilador: verifica se o texto digitado obedece à sintaxe da linguagem de programação e, caso isto ocorra, traduz o texto para uma sequência de instruções em **linguagem de máquina**.



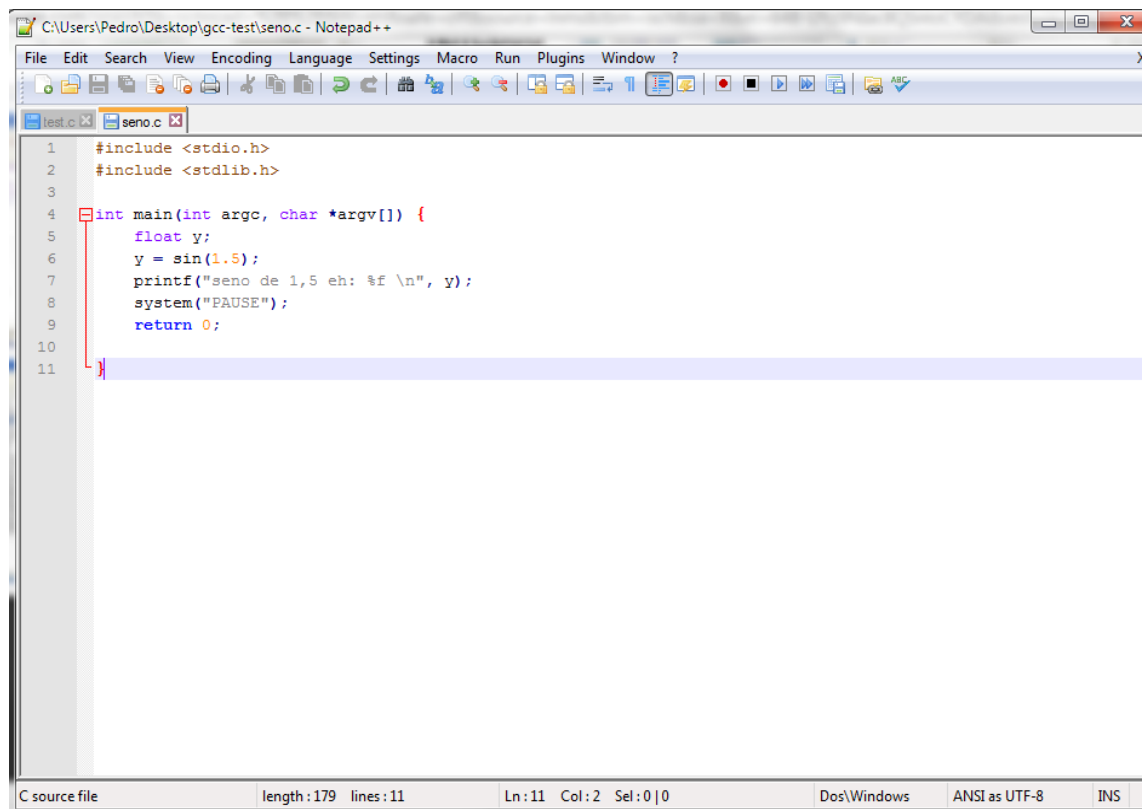
Código binário

# Definições

- Que ambiente de programação iremos utilizar?
  - Existem muitos, por exemplo: Microsoft Visual C++, Borland C++ Builder, Code Blocks, DEV-C++ etc.

# Definições

- Que ambiente de programação iremos utilizar?
  - Existem muitos, por exemplo: Microsoft Visual C++, Borland C++ Builder, Code Blocks, DEV-C++ etc.
- Não recomendo nenhum (notepad++ OU textpad e gcc)



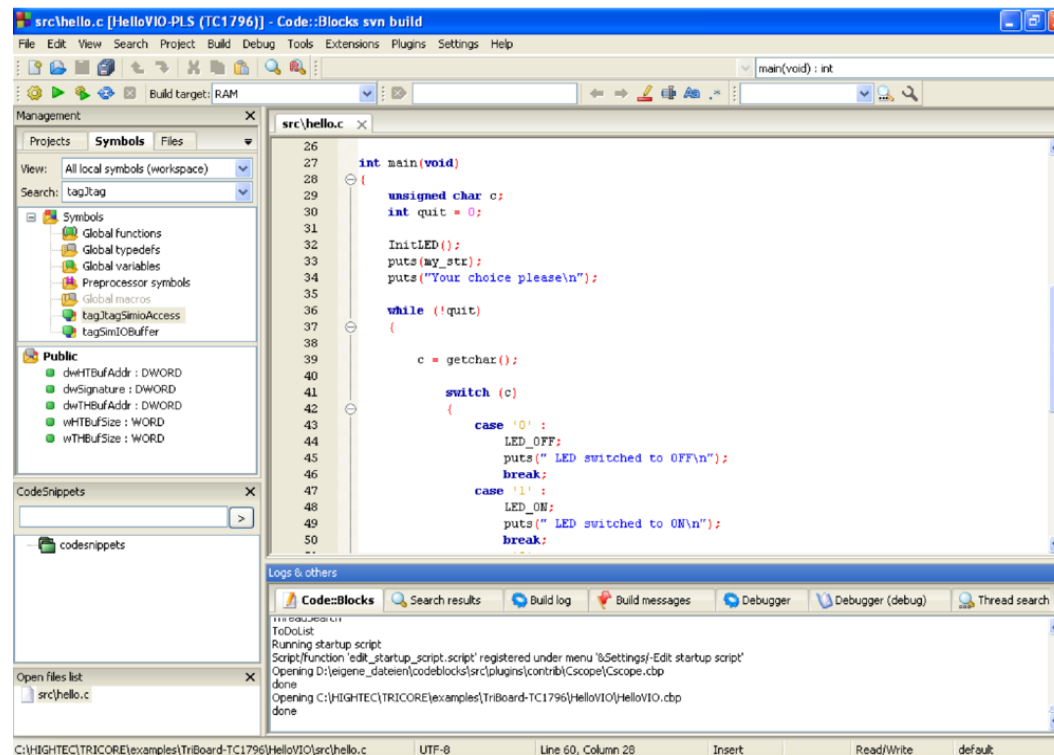
The image shows a Notepad++ window titled "C:\Users\Pedro\Desktop\gcc-test\seno.c - Notepad++". The editor contains a C program with the following code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[]) {
5     float y;
6     y = sin(1.5);
7     printf("seno de 1,5 eh: %f \n", y);
8     system("PAUSE");
9     return 0;
10 }
11
```

The status bar at the bottom indicates "C source file", "length: 179 lines: 11", "Ln: 11 Col: 2 Sel: 0 | 0", "Dos\Windows", "ANSI as UTF-8", and "INS".

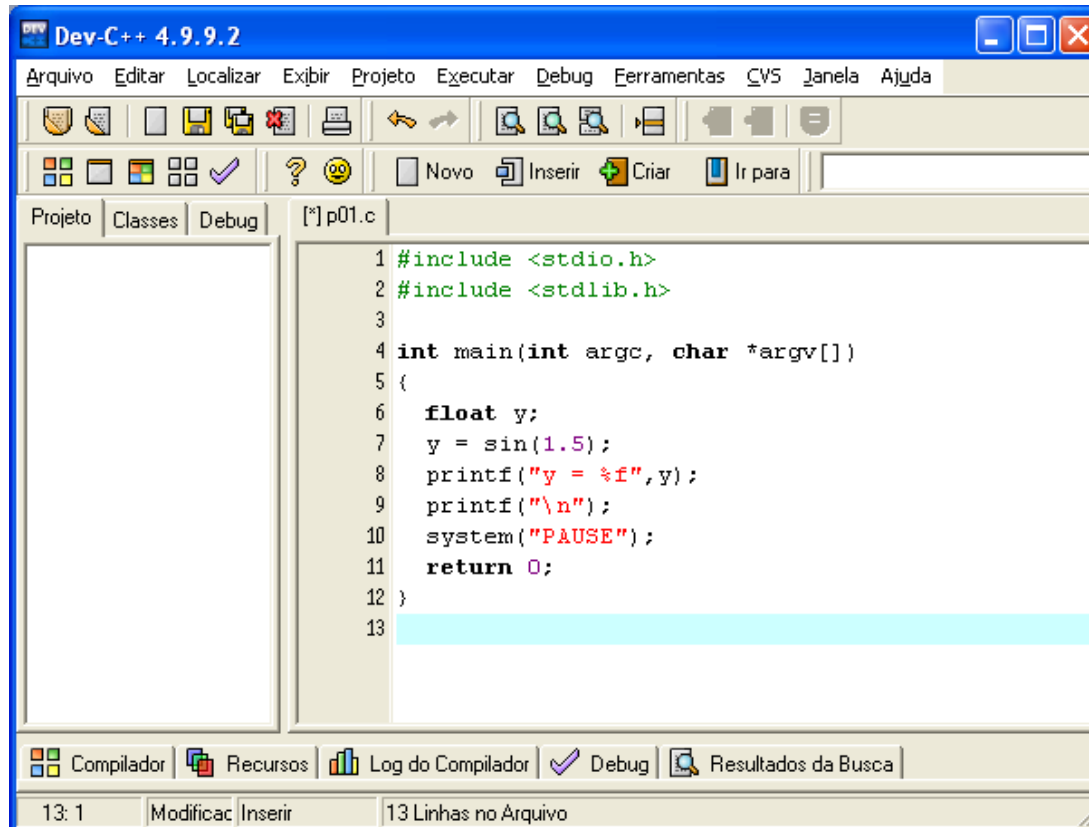
# Definições

- Que ambiente de programação iremos utilizar?
  - Existem muitos, por exemplo: Microsoft Visual C++, Borland C++ Builder, Code Blocks, DEV-C++ etc.
- Mas pode-se usar o Code Blocks (at your own risk!)



# Definições

- Que ambiente de programação iremos utilizar?
  - Existem muitos, por exemplo: Microsoft Visual C++, Borland C++ Builder, Code Blocks, DEV-C++ etc.
- Ou DEV-C++ (at your own risk!)



# Definições

- Porque o compilador traduz o programa escrito na linguagem de programação para a linguagem de máquina?

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char* argv[]) {
5     float y;
6     y = sin(1.5);
7     printf("seno de 1.5 eh: %f", y);
8     printf("\n");
9     system("PAUSE");
10    return 0;
11 }
```



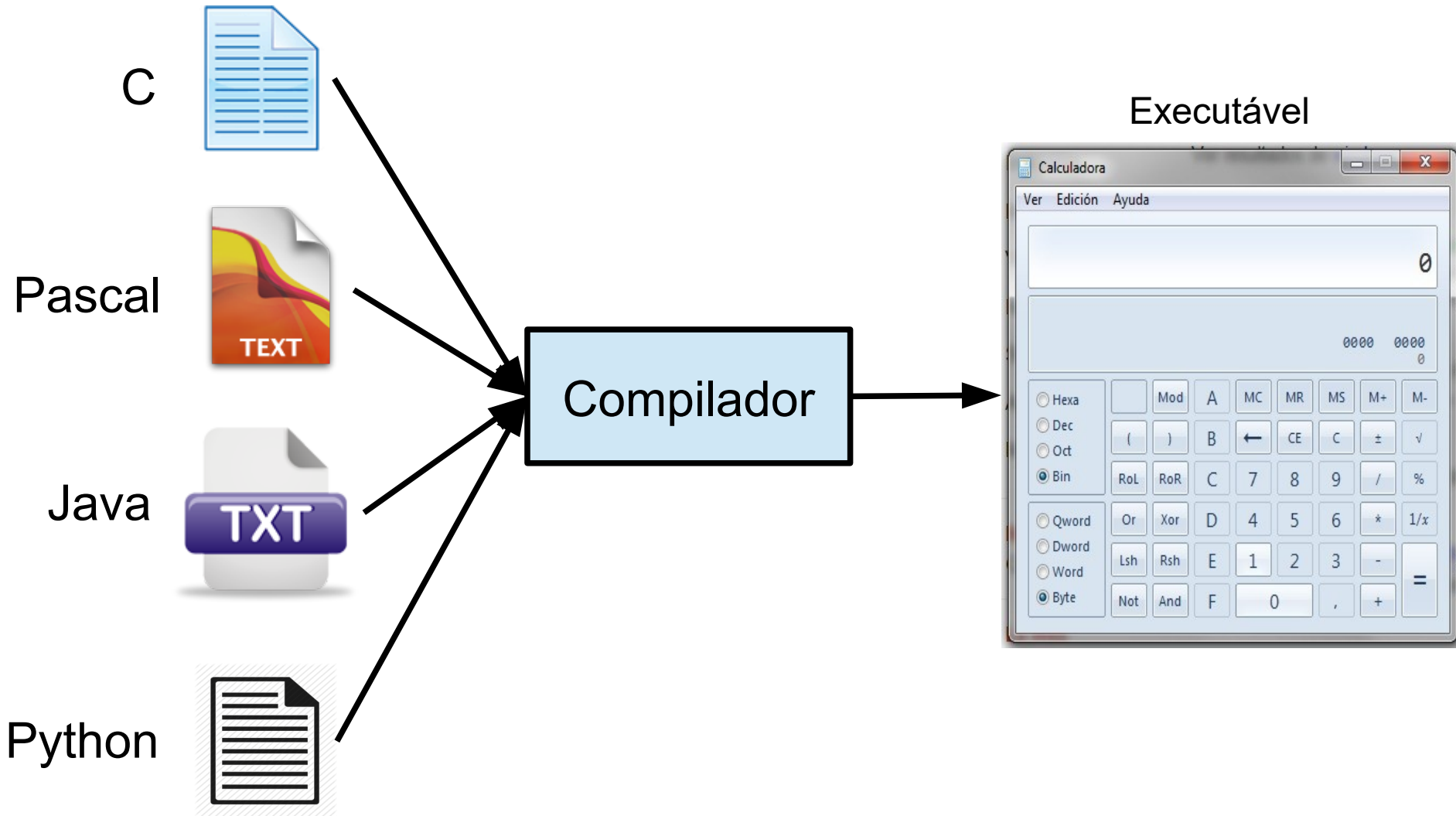
Compilador



```
0101010110100010011
1000101010111101111
1010100101100110011
0011001111100011100
0101010110100010011
1000101010111101111
1010100101100110011
0011001111100011100
```

- Os computadores atuais só conseguem executar instruções que estejam escritas na forma de códigos binários.
- Um programa em linguagem de máquina é chamado de programa executável.

# Definições



# Erros de sintaxe

- **Atenção!**

- O **programa executável** só será gerado se o texto do programa não contiver **erros de sintaxe**.
- Exemplo: considere uma **string**. Ah?! O que é isso?! Uma **sequência de caracteres delimitada por aspas**.
- Se isso é uma string e se tivéssemos escrito:

```
printf("\y = %f, y);
```

- O compilador iria apontar um erro de sintaxe nesta linha do programa e exibir uma mensagem tal como:

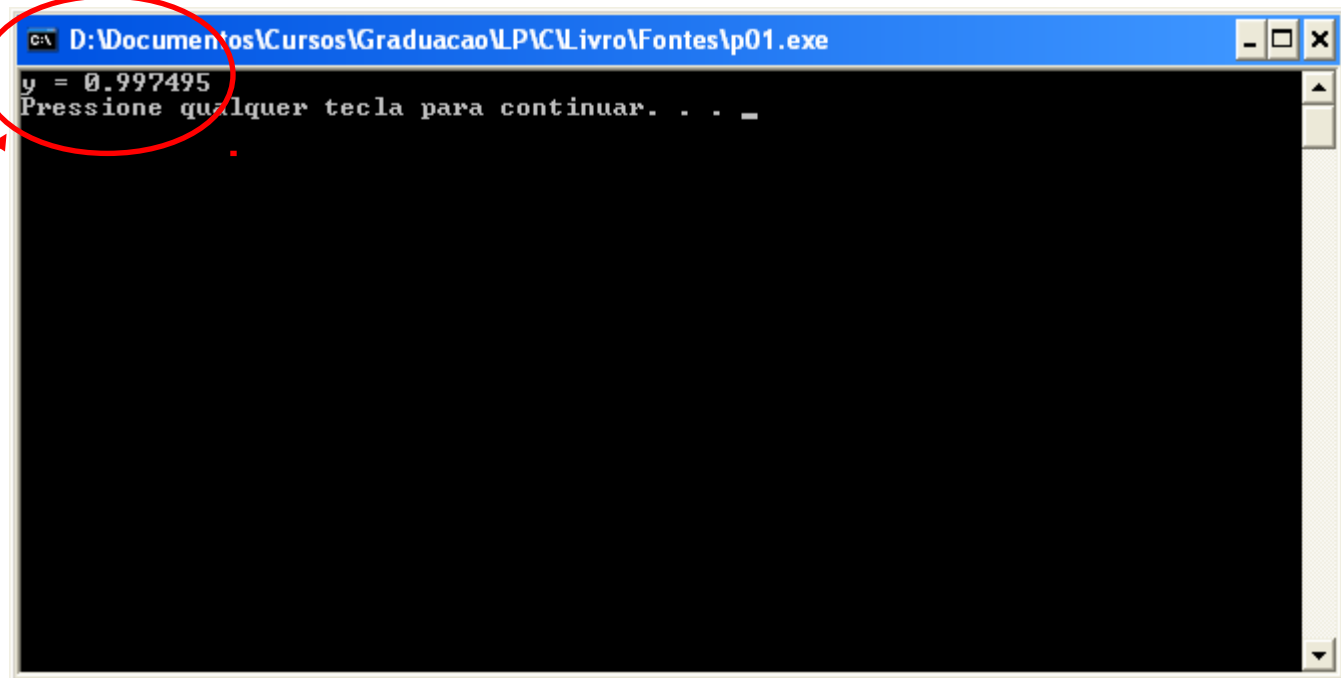
```
undetermined string or character constant
```



# Erros de sintaxe

- Se o nome do programa é **p1.c**, então após a **compilação** será produzido o programa executável **p1.exe** (ou **a.exe**).
- Executando-se o programa **p1.exe**, o resultado será:

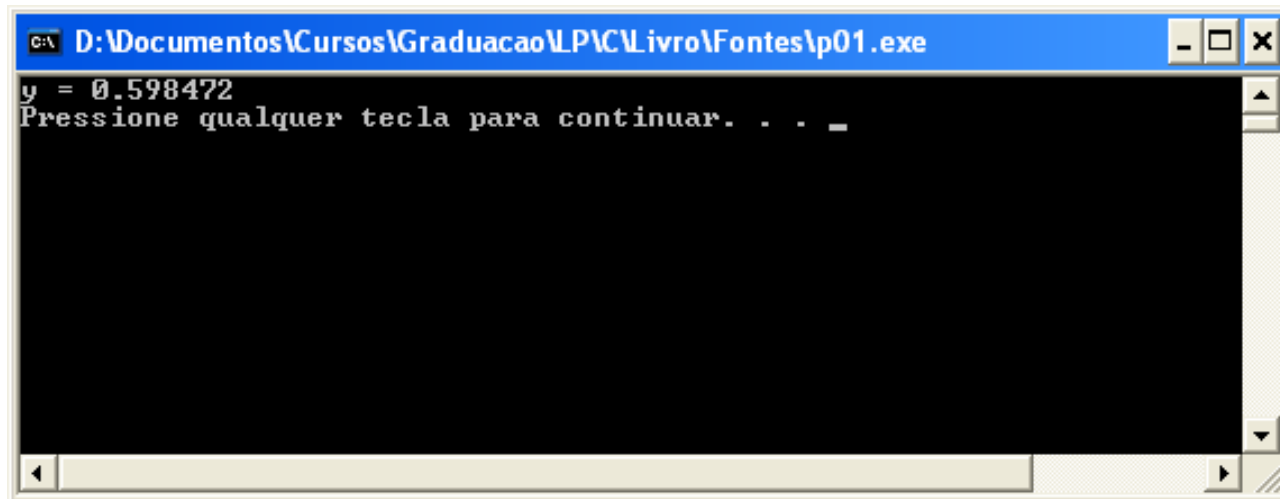
Problema  
Resolvido!



```
D:\Documentos\Cursos\Graduacao\LP\CLivro\Fontes\p01.exe
y = 0.997495
Pressione qualquer tecla para continuar. . . _
```

# Erros de lógica

- Atenção!
  - Não basta obter o programa executável!! Será que ele está correto?
  - Se ao invés de: `Y = sin(1.5);`
  - Tivéssemos escrito: `Y = sin(2.5);`
  - O compilador também produziria o programa p1.exe, que executado, iria produzir:



A screenshot of a Windows command prompt window. The title bar shows the file path: `C:\D:\Documentos\Cursos\Graduacao\LPIC\Livro\Fontes\p01.exe`. The window content displays the output of a program: `y = 0.598472` followed by the prompt `Pressione qualquer tecla para continuar. . . _`. The window has standard Windows controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

# Erros de lógica

- Embora um resultado tenha sido obtido, ele **não é correto**.
- Se um programa executável não produz os resultados corretos, é porque ele contém **erros de lógica** ou **bugs**.
- O processo de identificação e correção de erros de lógica é denominado **depuração (debug)**.
- O nome de um texto escrito em uma linguagem de programação é chamado de **programa-fonte**.  
Exemplo: o programa **p1.c** é um **programa-fonte**.

# Arquivos de cabeçalho

- Note que o programa-fonte p1.c começa com as linhas:

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

- Todo programa-fonte em linguagem C começa com linhas deste tipo.
- O que elas indicam?
  - Dizem ao compilador que o programa-fonte vai utilizar arquivos de cabeçalho (extensão `.h`, de `header`).
  - E daí? O que são estes arquivos de cabeçalho?
  - Eles `contêm informações` que o compilador precisa para construir o programa executável.

# Arquivos de cabeçalho

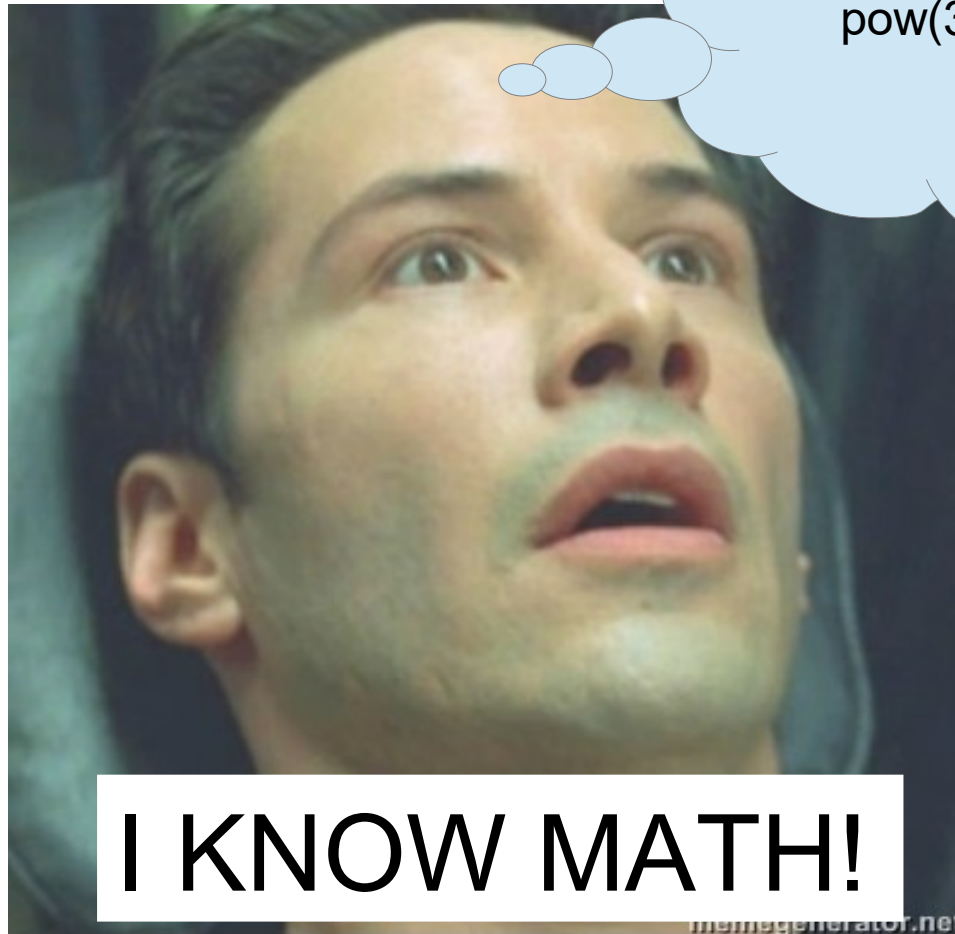
```
#include <kungfu.h>
```



# Arquivos de cabeçalho

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

$\sin(1.5) = 0.9975$   
 $\sqrt{429} = 20.71$   
 $\text{pow}(3,5) = 243$   
...



**I KNOW MATH!**

# Arquivos de cabeçalho

Como assim?

- Observe que o programa p1.c inclui algumas **funções**, tais como:  
**sin** – função matemática seno.  
**printf** – função para exibir resultados.
- Por serem muito utilizadas, a linguagem C mantém funções como estas em **bibliotecas**.
- Atenção! O conteúdo de um arquivo de cabeçalho também é um texto.

# Arquivos de cabeçalho

- Ao encontrar uma instrução `#include` em um programa-fonte, o compilador traduz este texto da mesma forma que o faria se o texto tivesse sido digitado no programa-fonte.
- Portanto, as linhas:

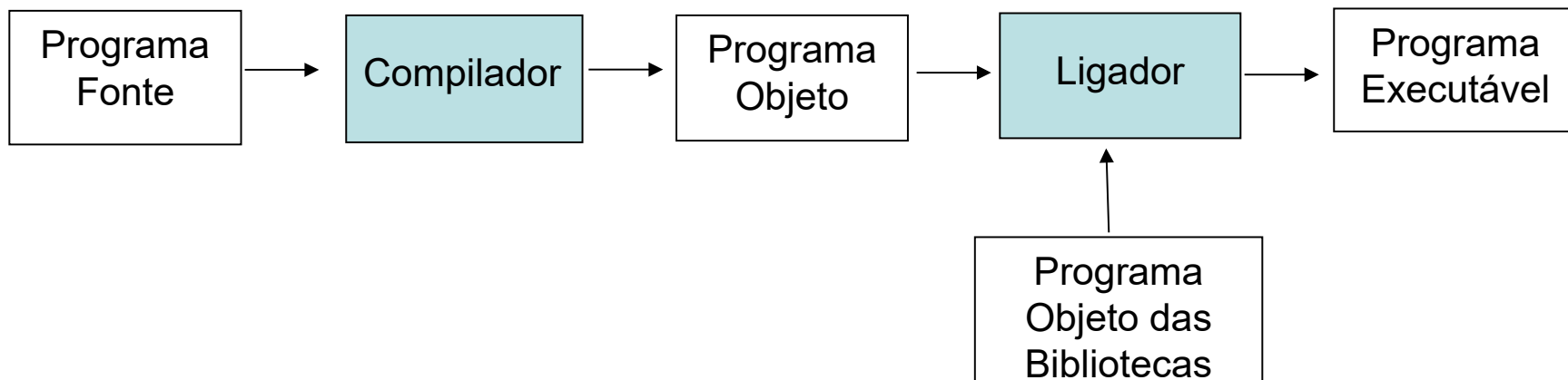
```
#include <stdio.h>  
#include <math.h>
```

indicam ao compilador que o programa `p1.c` utilizará as instruções das bibliotecas `stdio` e `stdlib`.



# Processo de compilação

- O processo de compilação, na verdade, se dá em duas etapas:
  - Fase de tradução: programa-fonte é transformado em um programa-objeto.
  - Fase de ligação: junta o programa-objeto às instruções necessárias das bibliotecas para produzir o programa executável.



# Função main

- A próxima linha do programa é:

```
int main(int argc, char *argv[])
```

- Esta linha corresponde ao cabeçalho da função **main** (a função principal, daí o nome **main**).
- O texto de um programa em Linguagem C pode conter muitas outras funções e **SEMPRE** deverá conter a **função main**.

int	main	(int argc, char *argv[])
-----	------	--------------------------

Indica o tipo do valor  
produzido pela função.

Nome da  
Função.

Lista de parâmetros  
da função.

# Função `main`

- A Linguagem C é *case sensitive*. Isto é, considera as letras maiúsculas e minúsculas diferentes.
- Atenção!
  - O nome da função principal deve ser escrito com letras minúsculas: `main`.
  - `Main` ou `MAIN`, por exemplo, provocam erros de sintaxe.
- Da mesma forma, as palavras `int` e `char`, devem ser escritas com letras minúsculas.

# Tipos de dados

- A solução de um problema de cálculo pode envolver vários tipos de dados.
- Caso mais comum são os **dados numéricos**:
  - **Números inteiros** (2, 3, -7, por exemplo).
  - **Números com parte inteira e parte fracionária** (1,234 e 7,83, por exemplo).
- Nas linguagens de programação, dá-se o nome de **número de ponto flutuante** aos números com parte inteira e parte fracionária.
- Da mesma forma que instruções, os dados de um programa devem ser representados em **notação binária**.
- Cada tipo de dado é representado na memória do computador de uma forma diferente.

# Notação Decimal

- $19.625 =$

$$1 \times 10^1 + 9 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3} =$$

$$10 + 9 + 0.6 + 0.02 + 0.005 = 19.625$$

# Notação Binária

- $10011.101 =$

$$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} =$$

$$16 + 0 + 0 + 2 + 1 + 0 + 0.5 + 0 + 0.125 = 19.625$$

- $19 / 2 =$

$1 \quad 9 / 2 =$

$1 \quad 4 / 2 =$

$0 \quad 2 / 2 =$

$0 \quad 1 / 2 =$

$1 \quad 0$

Condição de parada

# Notação Binária

- $0.625 \times 2 =$

**1** +  $0.25 \times 2 =$

**0** +  $0.5 \times 2 =$

**1** +  $0.0$

**0.101**

- $0.6 = ?$

preciso de quantos bits depois do “.” ?

# Notação Binária

- $0.625 \times 2 =$

**1** +  $0.25 \times 2 =$

**0** +  $0.5 \times 2 =$

**1** +  $0.0$

**0.101**

- $0.6 \times 2 =$

$1 + 0.2 \times 2 =$

$0 + 0.4 \times 2 =$

$0 + 0.8 \times 2 =$

$1 + 0.6 \times 2 = \dots$

$0.100110011001\dots$



# Armazenamento no computador

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15	b16
#E1	0	0	1	0	0	1	0	0	0	0	1	0	1	1	1	1
#E2	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0
#E3	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1
#E4	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1

# Representação de números inteiros

- Existem várias maneiras de representar números inteiros no sistema binário.
- Forma mais simples é a **senal-magnitude**:
  - O bit mais significativo corresponde ao sinal e os demais correspondem ao valor absoluto do número.
- Exemplo: considere uma representação usando cinco **dígitos binários** (ou **bits**).

Decimal

Binário

Desvantagens:

+5

00101

- Duas notações para o zero (+0 e -0).
- A representação dificulta os cálculos.

-3

10011

00101

10011

Soma 11000

← Que número é esse?

$5 - 3 = -8$  ???

# Representação de números inteiros

- Outra representação possível, habitualmente assumida pelos computadores, é a chamada **complemento-de-2**:
  - Para números positivos, a representação é idêntica à da forma sinal-magnitude.
  - Para os números negativos, a representação se dá em dois passos:
    1. Inverter os bits 0 e 1 da representação do número positivo;
    2. Somar 1 ao resultado.
  - Exemplo:

<u>Decimal</u>	<u>Binário</u>	
+6	00110	
-6	11001	(bits invertidos)
	1	(somar 1)
	11010	

# Representação de números inteiros

- Note o que ocorre com o zero:

<u>Decimal</u>	<u>Binário</u>
+0	00000
-0	11111 (bits invertidos)
	1 (somar 1)
	00000



Note que o **vai-um** daqui não é considerado, pois a representação usa apenas 5 bits.

- E a soma?

<u>Decimal</u>	<u>Binário</u>
+5	00101
-3	11100 + 1 = 11101

Somando:

00101

11101

00010

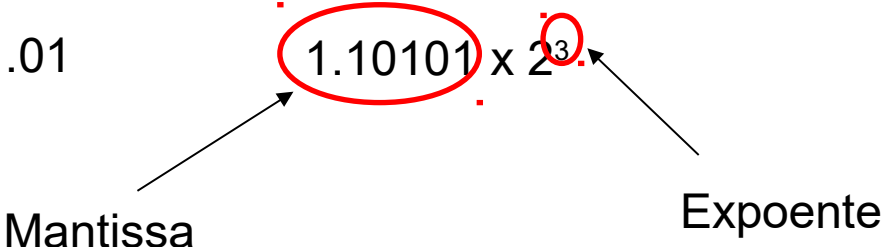
← Que corresponde ao número +2!

# Números de ponto flutuante

- Números de ponto flutuante são os números reais que podem ser representados no computador.
- Ponto flutuante não é um ponto que flutua no ar!
- Exemplo:
  - Representação com ponto fixo: 12,34.
  - Representação com ponto flutuante:  $0,1234 \times 10^2$ .
- Ponto Flutuante ou Vírgula Flutuante?
- A representação com ponto flutuante segue padrões internacionais (IEEE-754 e IEC-559).

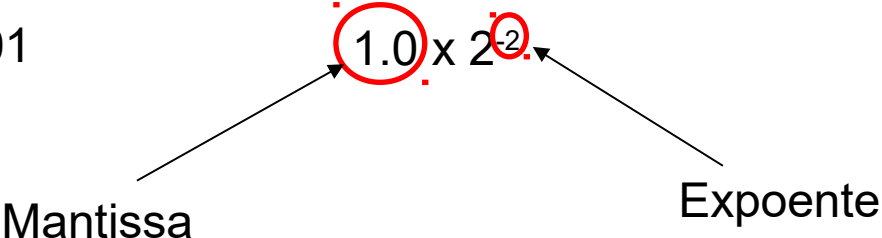
# Números de ponto flutuante

- A representação com ponto flutuante tem três partes: o **sinal**, a **mantissa** e o **expoente**.
- No caso de computadores, a mantissa é representada na forma normalizada, ou seja, na forma  $1.f$ , onde  $f$  corresponde aos demais bits.
- Ou seja, o primeiro bit sempre é 1.
- Exemplo 1:

<u>Decimal</u>	<u>Binário</u>	<u>Binário normalizado</u>
+13.25	1101.01	$1.10101 \times 2^3$
		

# Números de ponto flutuante

- Exemplo 2:

<u>Decimal</u>	<u>Binário</u>	<u>Binário normalizado</u>
+0.25	0.01	$1.0 \times 2^{-2}$
		

- Existem dois formatos importantes para os números de ponto flutuante:
  - Precisão simples (SP).
  - Precisão dupla (DP).

# Números de ponto flutuante

- Precisão Simples

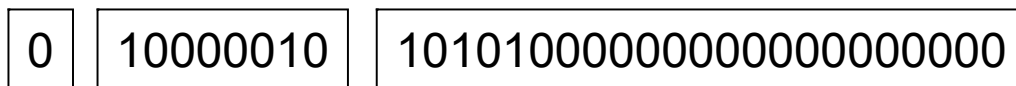
- Ocupa 32 bits: 1 bit de sinal, 23 bits para a mantissa e 8 bits para o expoente (representado na notação **excesso-de-127**).

- Exemplo:

**Ponto flutuante**

$$1.10101 \times 2^3$$

**Representação SP**

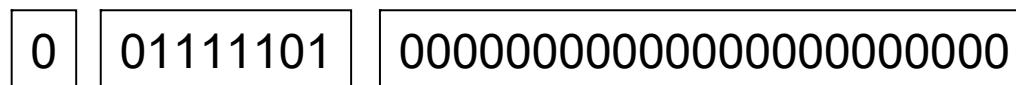


127 é o novo 0  
130 é o novo 3

**Ponto flutuante**

$$1.0 \times 2^{-2}$$

**Representação SP**



- O primeiro bit da mantissa de um número de ponto flutuante não precisa ser representado (sempre 1).



# Números de ponto flutuante

- Precisão Simples - Valores especiais

IEEE 754 - Single Precision			Valor	
s	e	m		
0	0000 0000	000 0000 0000 0000 0000 0000	+0	Zero
1	0000 0000	000 0000 0000 0000 0000 0000	-0	
0	1111 1111	000 0000 0000 0000 0000 0000	+Inf	Infinito Positivo
1	1111 1111	000 0000 0000 0000 0000 0000	-Inf	Infinito Negativo
0	1111 1111	010 0000 0000 0000 0000 0000	+NaN	Not a Number
1	1111 1111	010 0000 0000 0000 0000 0000	-NaN	

5/0

-3/0

0/0 ou  $\infty/\infty$

# Números de ponto flutuante

- Observações – Precisão Simples:
  - Dado que para o expoente são reservados 8 bits, ele poderá ser representado por 256 ( $2^8$ ) valores distintos (0 a 255).
  - Usando-se a notação **excesso-de-127**, tem-se:
    - para um expoente igual a -127, o mesmo será representado por 0 (**valor especial! Número Zero**).
    - para um expoente igual a 128, o mesmo será representado por 255 (**valor especial! Infinito**).
  - Conclusão, os números normalizados representáveis possuem expoentes entre -126 e 127.

# Números de ponto flutuante

- Precisão Dupla

Ocupa 64 bits: 1 bit de sinal, 52 bits para a mantissa e 11 bits para o expoente (representado na notação **excesso-de-1023**).

- Exemplo: Similar ao abordado para precisão simples...

# Representação de dados não-numéricos

- A solução de um problema pode envolver dados não numéricos.
- Por exemplo, o programa `p1.c` inclui **strings** (sequências de caracteres delimitadas por aspas).

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Representação de dados não-numéricos

- Existem também padrões internacionais para a codificação de caracteres ([ASCII](#), [ANSI](#), [Unicode](#)).
- A Linguagem C adota o padrão ASCII (American Standard Code for Information Interchange):
  - Código para representar caracteres como números.
  - Cada caractere é representado por [1 byte](#), ou seja, uma [seqüência de 8 bits](#).

# Representação de dados não-numéricos

A Linguagem C adota o padrão ASCII (American Standard Code for Information Interchange):

- Código para representar caracteres como números.
- Cada caractere é representado por **1 byte**, ou seja, uma **seqüência de 8 bits**.
- Por exemplo:

Caractere	Decimal	ASCII
'A'	65	01000001
'@'	64	01000000
'a'	97	01100001

# Escrevendo um programa em C

- Escrever um programa em Linguagem C corresponde a escrever o **corpo** da função principal (**main**).
- O **corpo** de uma função sempre começa com abre-chaves **{** e termina com fecha-chaves **}**.

Corpo da  
função

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- Escrever um programa em Linguagem C corresponde a escrever o **corpo** da função principal (**main**).

```
void main(void) {  
    //corpo da função (“//” indica um comentário)  
}
```

```
int main(int argc, char* argv[]) {  
    //corpo da função (“//” indica um comentário)  
    return 0; //ou qualquer número inteiro  
}
```



# Escrevendo um programa em C

**IMPORTANTE:** Todos os comandos do corpo de uma função ou procedimento devem terminar com ponto e vírgula “;”

```
int a;  
a = 45;  
printf(“valor de a: %d”, a);
```

# Escrevendo um programa em C

Para imprimir algo na tela, use a função ***printf*** da biblioteca ***stdio.h***

```
printf("algo");
```

```
printf("Eu tenho %d reais e %d centavos", 4, 20);
```

```
// %d é usado para formatar um número inteiro
```

```
printf("Minha nota foi %f", 9.25);
```

```
// %f é usado para formatar um número ponto flutuante
```

# Meu primeiro programa

Já sabemos escrever o nosso primeiro programa:

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

```
void main() {  
    printf("Alo mundo!");  
}
```

# Meu segundo programa

- Uma conta poupança foi aberta com um depósito de R\$500,00, com rendimentos 1% de juros ao mês. No segundo mês, R\$200,00 reais foram depositados nessa conta poupança. No terceiro mês, R\$50,00 reais foram retirados da conta. Quanto haverá nessa conta no quarto mês?

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

endereço	conteúdo
Rua do Ouro, 12	Edifício Luz
Rua do Ouro, 13	Casa da Maria
Rua do Ouro, 14	Padaria do Zé
Rua do Ouro, 15	Farmácia Legal
Rua do Ouro, 16	Casa do João
Rua do Ouro, 17	Edifício do Sol

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

Cada gaveta tem uma etiqueta e um espaço bem delimitado. No entanto, você pode guardar diversas coisas dentro delas.



# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

endereço	conteúdo
6612	891
6613	'a'
6614	8
6615	16.1
6616	0.4543
6617	298347



# Variáveis

- A partir dos endereços, é possível para o computador saber qual é o valor armazenado em cada uma das posições de memória.
- Como a **memória pode ter bilhões de posições**, é difícil controlar em qual endereço está armazenado um determinado valor!
- Para facilitar o controle sobre onde armazenar informação, os programas utilizam **variáveis**.
- Uma variável corresponde a um **nome simbólico (ou etiqueta)** de uma posição de memória.
- Seu **conteúdo pode variar** durante a execução do programa.

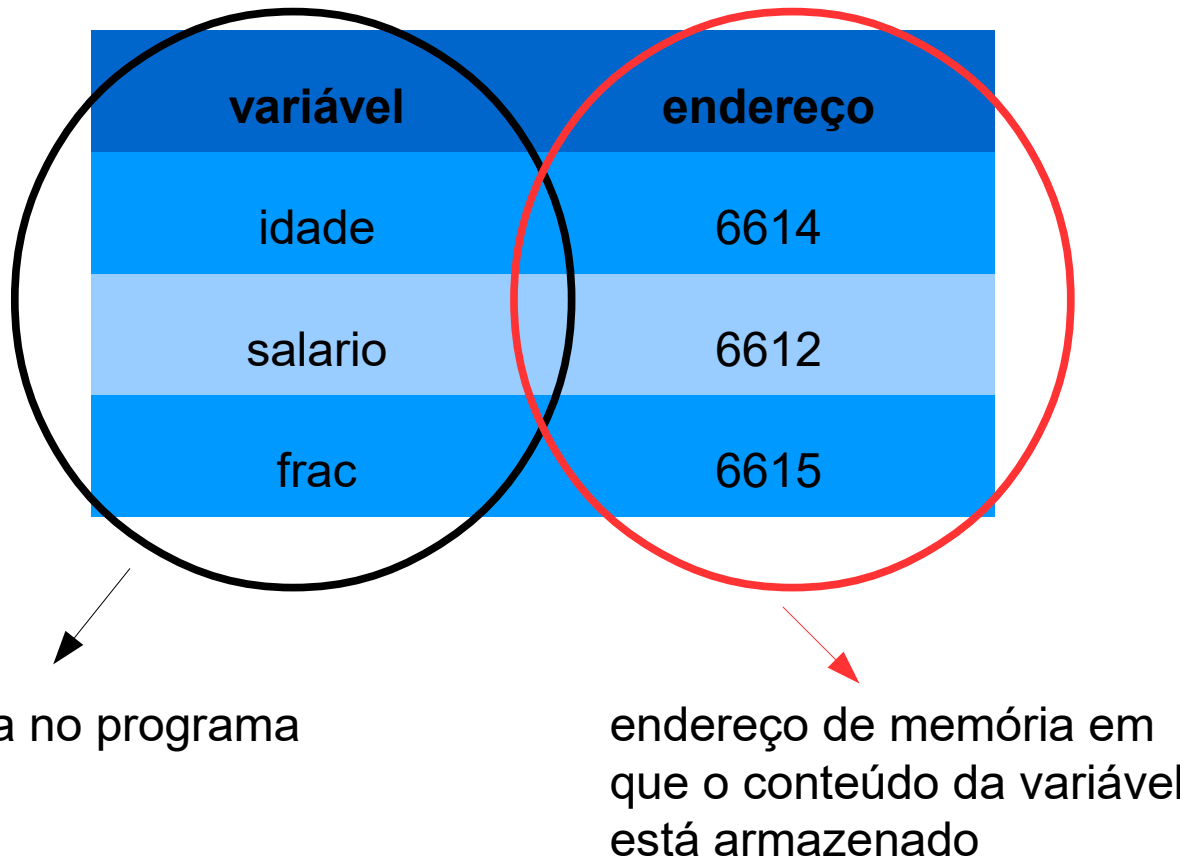
# Variáveis

- Dicionário de variáveis do compilador

variável	endereço
idade	6614
salario	6612
frac	6615

# Variáveis

- Dicionário de variáveis



# Variáveis

## Dicionário de variáveis

<b>variável</b>	<b>endereço</b>
idade	6614
salario	6612
frac	6615



## Memória do computador

<b>endereço</b>	<b>conteúdo</b>
6611	9439.23496
6612	891
6613	'P'
6614	8
6615	0.4543
6616	2365

# Variáveis

- Memória + dicionário de variáveis  
(vamos usar esta representação ao longo do curso!)

endereço	variável	conteúdo
6612	salario	891
6613	c	'a'
6614	idade	8
6615	velocidade	16.1
6616	frac	0.4543
6617	km	298347

# Variáveis

- Exemplo de variável:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

A variável **y** irá armazenar o valor de **sin(1.5)**.

# Variáveis

- Cada variável pode possuir uma quantidade diferente de bytes, uma vez que os tipos de dados são representados de forma diferente.
- Portanto, a cada variável está associado um tipo específico de dados.
- Logo:
  - O tipo da variável define quantos bytes de memória serão necessários para representar os dados que a variável armazena.

# Variáveis

- A Linguagem C dispõe de **quatro tipos básicos de dados**. Assim, as variáveis poderão assumir os seguintes tipos:

tipo	tamanho (bytes)	valor
char	1	Um caractere ou um inteiro de 0 a 127
int	4	um número inteiro
float	4	um número de ponto flutuante (SP)
double	8	um número de ponto flutuante (DP)



# Variáveis

- Dentro do programa, as variáveis são identificadas por seus **nomes**.
- Portanto, um programa deve **declarar** todas as variáveis que irá utilizar.
- Atenção!
  - A declaração de variáveis deve ser feita antes que a **variável seja usada**, para garantir que a quantidade correta de memória já tenha sido reservada para armazenar seu valor.

# Variáveis

- Para assinalar valores à variáveis deve-se usar o **operador de atribuição =**

nome\_da\_variavel = valor;

# Variáveis

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

```
void main(void) {
```

```
    int idade;
```

```
    float salario;
```

```
    char sexo;
```

```
    double divida;
```

```
    idade = 25;
```

```
    salario = 100.5;
```

```
    sexo = 'M';
```

```
    divida = 29999.99;
```

```
    printf("Eu tenho %d anos,", idade);
```

```
    printf(" recebo %f reais por mes,", salario);
```

```
    printf(" sou do sexo %c", sexo);
```

```
    printf(" e tenho uma divida de %f", divida);
```

```
    printf("\n");
```

```
    system("PAUSE");
```

```
}
```

Endereço	Variável	Conteúdo
4812		
4813		
4814		
4815		
4816		
4817		
4818		
4819		

# Variáveis

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

```
void main(void) {
```

```
int idade;
```

```
float salario;
```

```
char sexo;
```

```
double divida;
```

```
idade = 25;
```

```
salario = 100.5;
```

```
sexo = 'M';
```

```
divida = 29999.99;
```

```
printf("Eu tenho %d anos,", idade);
```

```
printf(" recebo %f reais por mes,", salario);
```

```
printf(" sou do sexo %c", sexo);
```

```
printf(" e tenho uma divida de %f", divida);
```

```
printf("\n");
```

```
system("PAUSE");
```

```
}
```

Endereço	Variável	Conteúdo
4812	idade	
4813	salario	
4814	sexo	
4815	divida	
4816		
4817		
4818		
4819		

# Variáveis

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

```
void main(void) {  
    int idade;  
    float salario;  
    char sexo;  
    double divida;
```

```
    idade = 25;  
    salario = 100.5;  
    sexo = 'M';  
    divida = 29999.99;
```

```
    printf("Eu tenho %d anos,", idade);  
    printf(" recebo %f reais por mes,", salario);  
    printf(" sou do sexo %c", sexo);  
    printf(" e tenho uma divida de %f", divida);  
    printf("\n");  
    system("PAUSE");  
}
```

Endereço	Variável	Conteúdo
4812	idade	25
4813	salario	100.5
4814	sexo	'M'
4815	divida	29999.99
4816		
4817		
4818		
4819		

# Escrevendo um programa em C

- A primeira linha do corpo da função principal do programa `p1.c` é:

```
float y;
```

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- Declarando duas ou mais variáveis do mesmo tipo

```
float y, aux, salario;
```

# Escrevendo um programa em C

- Esta linha declara uma variável *y* para armazenar um número de ponto flutuante (SP).
- A declaração de uma variável não armazena valor algum na posição de memória que a variável representa.
- Ou seja, no caso anterior, vai existir uma posição de memória chamada *y*, mas ainda não vai existir valor armazenado nesta posição.



# Escrevendo um programa em C

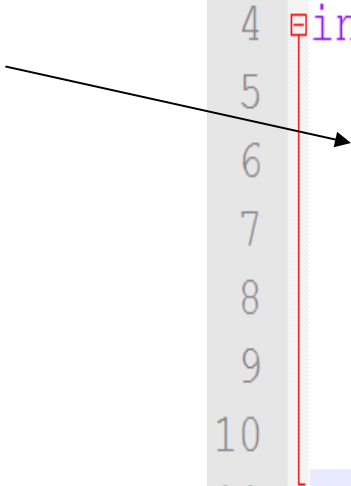
- Um valor pode ser **atribuído** a uma posição de memória representada por uma variável pelo **operador de atribuição =**.
- O operador de atribuição requer à esquerda um nome de variável e à direita, um valor.
- A linha seguinte de **p1.c** atribui um valor a **y**:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- No lado direito do operador de atribuição existe uma referência à função **seno** com um parâmetro **1.5** (uma constante de ponto flutuante representando um valor em **radianos**.)

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```



# Escrevendo um programa em C

- Em uma linguagem de programação chamamos o valor entre parênteses da função, neste exemplo, o valor 1.5, de **parâmetro da função**.
- Da mesma forma, diz-se que **sin(1.5)** é o valor da **função sin** para o parâmetro **1.5**.
- O operador de atribuição na linha **y = sin(1.5)** obtém o valor da função (0.997495) e o armazena na posição de memória identificada pelo nome **y**.
- Esta operação recebe o nome de: **atribuição de valor a uma variável**.

# Escrevendo um programa em C

- Atenção: O valor armazenado em uma variável por uma operação de atribuição depende do tipo da variável.
- Se o tipo da variável for `int`, será armazenado um valor inteiro (caso o valor possua parte fracionária, ela será desprezada).
- Se o tipo da variável for `float` ou `double`, será armazenado um valor de ponto flutuante (caso o valor não possua parte fracionária, ela será nula).

# Escrevendo um programa em C

- Operações matemáticas básicas
  - multiplicação (operador \*)
    - `var3 = var1 * var2;`
  - divisão (operador /)
    - `var3 = var1 / var2;`
  - soma (operador +)
    - `var3 = var1 + var2;`
  - subtração (operador -)
    - `var3 = var1 - var2;`

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- Neste caso, teremos:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	
$b = (1 - 4) / (2 - 5)$	
$a = 2.75 + 1.12$	
$b = a / 2.0$	

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- Neste caso, teremos:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	20
$b = (1 - 4) / (2 - 5)$	1.0
$a = 2.75 + 1.12$	3
$b = a / 2.0$	1.5

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512		
8513		
8514		
8515		
8516		
8517		
8518		
8519		



# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

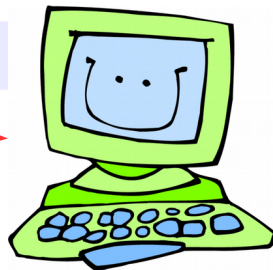
```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

**sin(1.5)**



(processador)

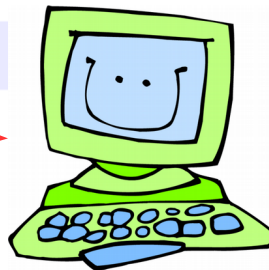
Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512	y	0.997495
8513		
8514		
8515		
8516		
8517		
8518		
8519		

sin(1.5)

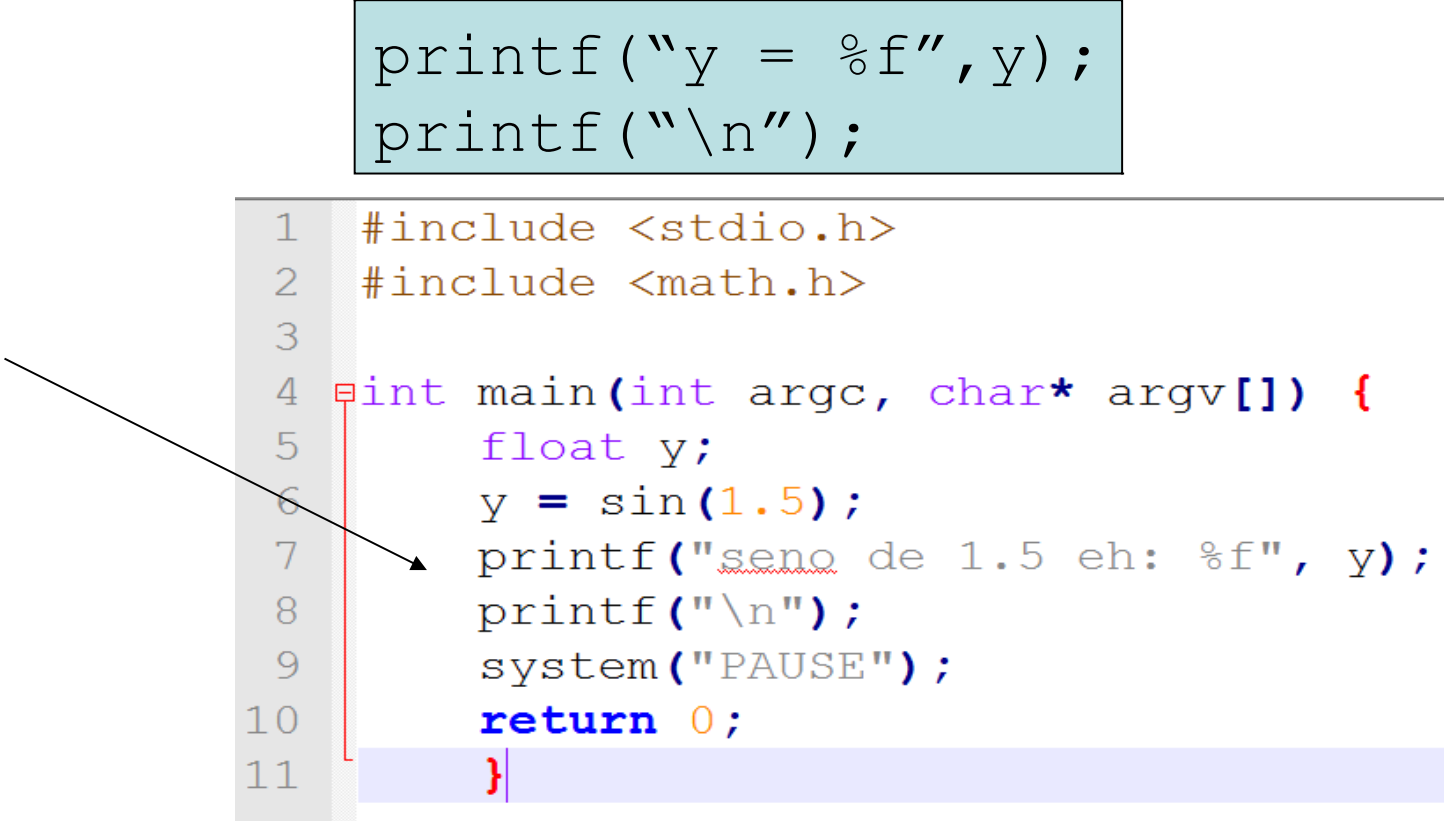


(processador)

# Escrevendo um programa em C

- As próximas linhas do programa `p1.c` são:

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



```
1  #include <stdio.h>  
2  #include <math.h>  
3  
4  int main(int argc, char* argv[]) {  
5      float y;  
6      y = sin(1.5);  
7      printf("seno de 1.5 eh: %f", y);  
8      printf("\n");  
9      system("PAUSE");  
10     return 0;  
11 }
```

- A função `printf` faz parte da biblioteca `stdio`.

# Escrevendo um programa em C

- A função `printf` é usada para exibir resultados produzidos pelo programa e **pode ter um ou mais parâmetros**.
- O primeiro parâmetro da função `printf` é sempre uma **string**, correspondente à sequência de caracteres que será exibida pelo programa.

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

# Escrevendo um programa em C

- Essa sequência de caracteres pode conter alguns **tags** que representam valores. Estes tags são conhecidos como **especificadores de formato**.

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

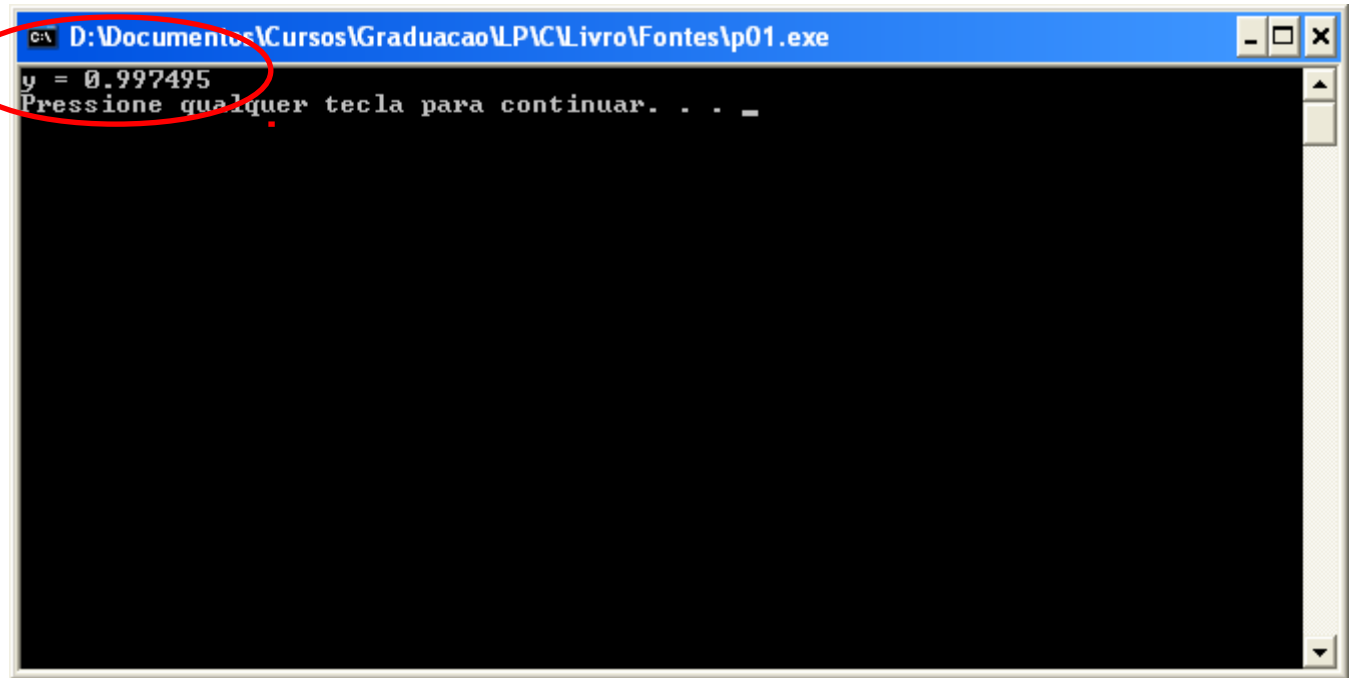
**Especificador  
de formato**

- Um especificador de formato começa sempre com o símbolo **%**. Em seguida, pode apresentar uma **letra** que indica o tipo do valor a ser exibido.
- Assim, **printf("y = %f", y)** irá exibir a letra **y**, um espaço em branco, o símbolo **=**, um espaço em branco, e um valor de ponto flutuante.

# Escrevendo um programa em C

- Veja:

Valor  
armazenado  
em *y*.



```
C:\> D:\Documentos\Cursos\Graduacao\LPIC\Livro\Fontes\lp01.exe
y = 0.997495
Pressione qualquer tecla para continuar. . . _
```



# Escrevendo um programa em C

- Na função `printf`, para cada `tag` existente no primeiro parâmetro, deverá haver um novo parâmetro que especifica o valor a ser exibido.

```
printf("a = %d, b = %c e c = %f", a, 'm', (a+b) );
```

# Escrevendo um programa em C

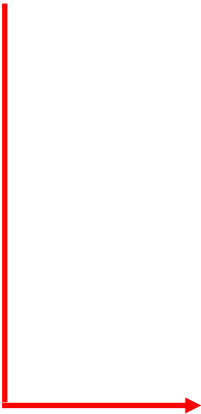
- A linguagem C utiliza o símbolo `\` (barra invertida) para especificar alguns caracteres especiais:

Caractere	Significado
<code>\a</code>	Caractere (invisível) de aviso sonoro.
<code>\n</code>	Caractere (invisível) de nova linha.
<code>\t</code>	Caractere (invisível) de tabulação horizontal.
<code>\'</code>	Caractere de apóstrofo

# Escrevendo um programa em C

- Observe a próxima linha do programa `p1.c`:

```
printf("\n");
```



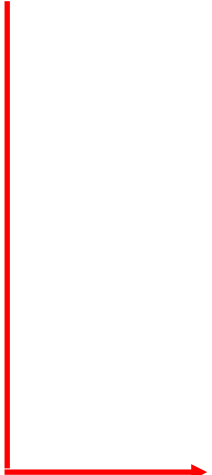
```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

- Ela exibe “o caractere (invisível) de nova linha”. Qual o efeito disso? Provoca uma mudança de linha! Próxima mensagem será na próxima linha.

# Escrevendo um programa em C

- Observe agora a próxima linha do programa:

```
system("PAUSE");
```




```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

- Ela exibe a mensagem “**Pressione qualquer tecla para continuar...**” e interrompe a execução do programa.

# Escrevendo um programa em C

- A execução será retomada quando o usuário pressionar alguma tecla.
- A última linha do programa `p1.c` é:

```
return 0;
```




```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- É usada apenas para satisfazer a sintaxe da linguagem C.
- O comando `return` indica o valor que uma função produz.
- Cada função, assim como na matemática, deve produzir um único valor.
- Este valor deve ter o mesmo tipo que o declarado para a função.

# Escrevendo um programa em C

- No caso do programa `p1.c`, a função principal foi declarada como sendo do tipo `int`. Ou seja, ela deve produzir um valor inteiro.



```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

- A linha `return 0;` indica que a função principal irá produzir o valor inteiro 0.

# Escrevendo um programa em C

- Mas e daí?!! O valor produzido pela função principal não é usado em lugar algum!
- Logo, não faz diferença se a última linha do programa for:

```
return 0;
```

```
return 1;
```

ou

```
return 1234;
```



# Escrevendo um programa em C

- Neste caso, o fato de a função produzir um valor não é relevante.
- Neste cenário, é possível declarar a função na forma de um **procedimento**.
- Um **procedimento** é uma função do tipo **void**, ou seja, uma função que produz o valor **void** (**vazio**, **inútil**, **à-toa**). Neste caso, ela não precisa do comando **return**.

# Escrevendo um programa em C

- Note que os parâmetros da função **main** também não foram usados neste caso.
- Portanto, podemos também indicar com **void** que a lista de parâmetros da função principal é vazia.
- Assim, podemos ter outras formas para **p1.c**:

```
void main(void)
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return;
}
```

```
void main(void)
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
}
```

# Exercício

- Uma conta poupança foi aberta com um depósito de R\$500,00, com rendimentos 1% de juros ao mês. No segundo mês, R\$200,00 reais foram depositados nessa conta poupança. No terceiro mês, R\$50,00 reais foram retirados da conta. Quanto haverá nessa conta no quarto mês?