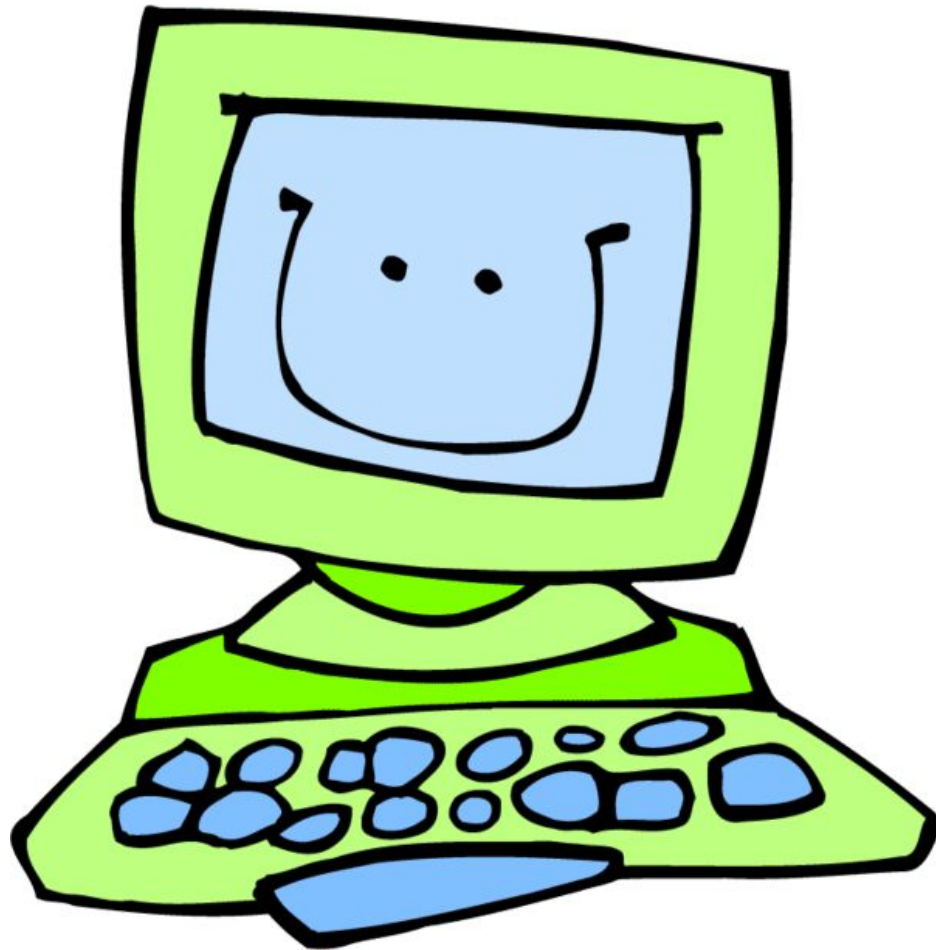


## Introdução à Programação de Computadores

Compilação,  
memória e variáveis

# 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 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**.

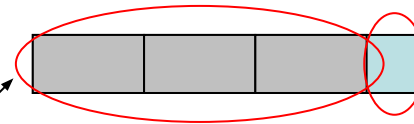
A

B

$A = 7, B = 2$

Quociente de  $A/B$ : nº de vezes que B cabe em A.

O que sobra em A é o resto da divisão.



# 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

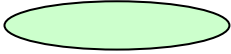

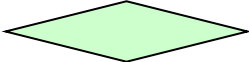



- Pode-se escrever este algoritmo como:

- 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);



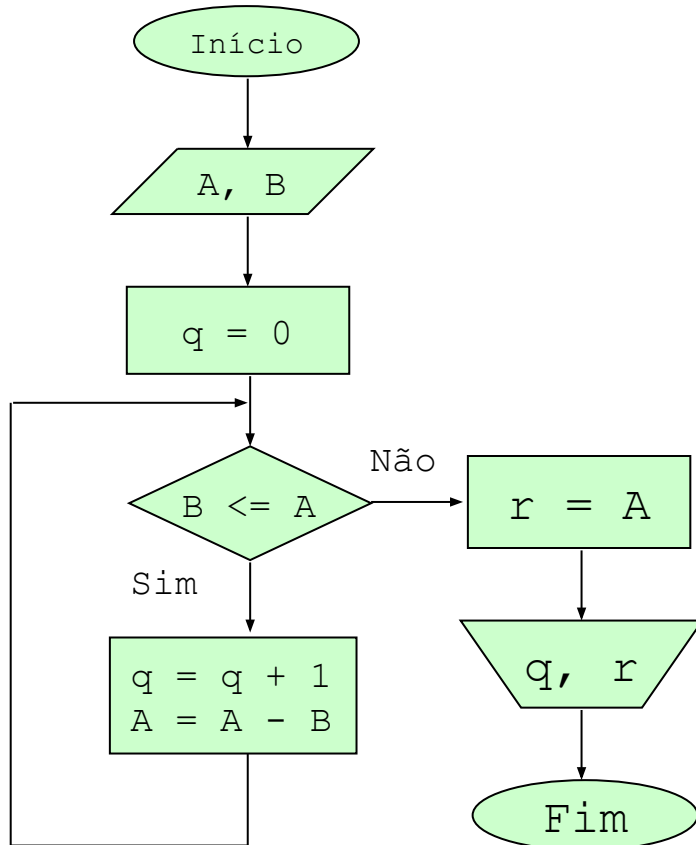
# Fluxograma

- 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.

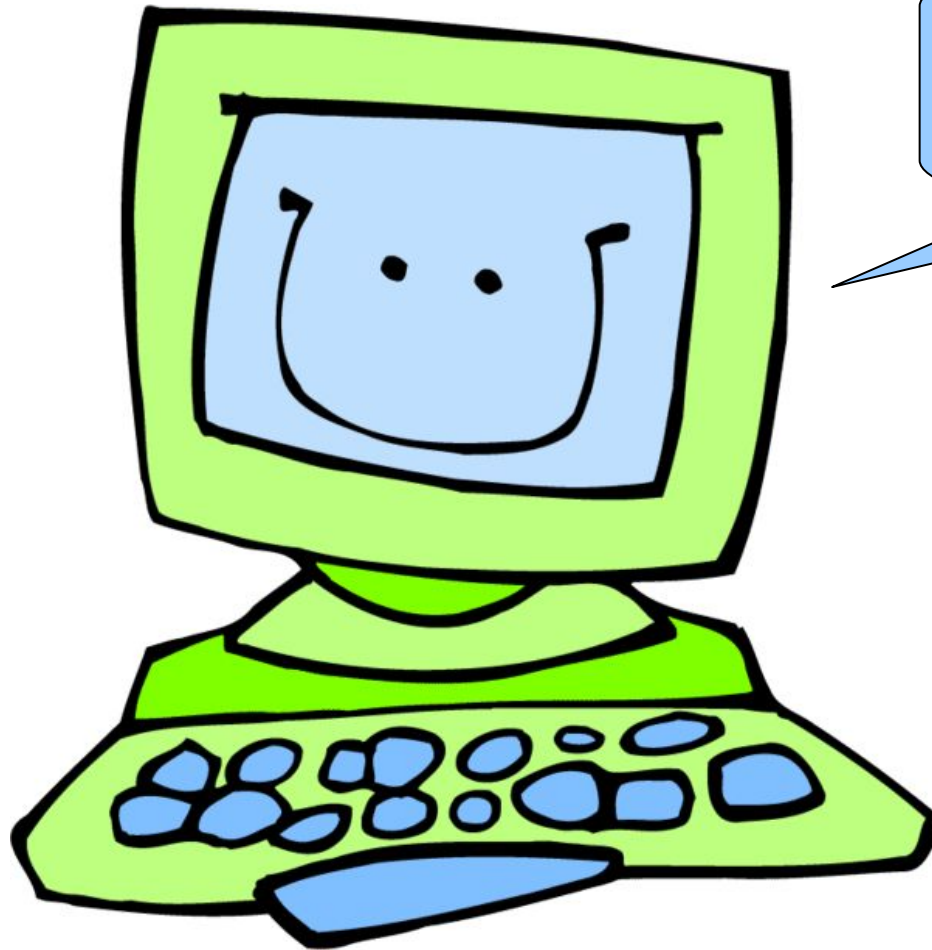
# Fluxograma

- **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?



010101001010101101011110010  
000010101101010110101101011  
101011010110001010101101011  
000

# 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 e exibir o valor de  $y = \text{seno}(1,5)$ .
  - Escrever um programa:

```
000101010111010111
001010111010101111
011101011101011100
```

# Como conversar com um computador?

- Considere o seguinte problema:
  - Determinar e exibir 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 e exibir 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

# Ambiente de Programação

- Que ambiente de programação iremos utilizar?
  - Existem muitos ambientes de programação integrados (IDEs)
    - Microsoft Visual C++
    - Borland C++ Builder
    - Code Blocks
    - DEV-C++
    - etc

# Ambiente de Programação

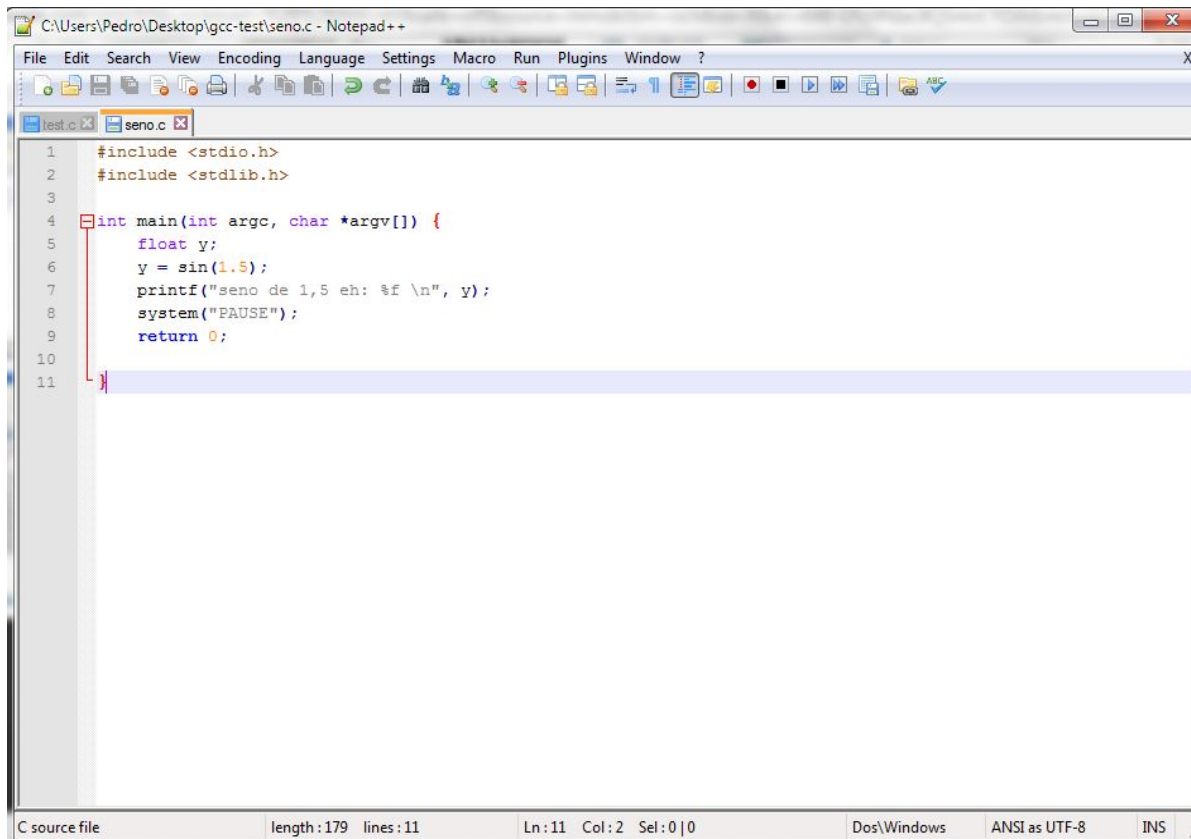
- Que ambiente de programação iremos utilizar?
  - Existem muitos ambientes de programação integrados (IDEs)
- Não recomendo nenhum!

# Ambiente de Programação

- Que ambiente de programação iremos utilizar?
  - Recomendo: editor de texto + gcc

# Editores de texto recomendados (gratuitos)

- Atom: <https://atom.io>
- Sublime: <https://www.sublimetext.com/>
- Notepad++: <https://notepad-plus-plus.org>
- VS code



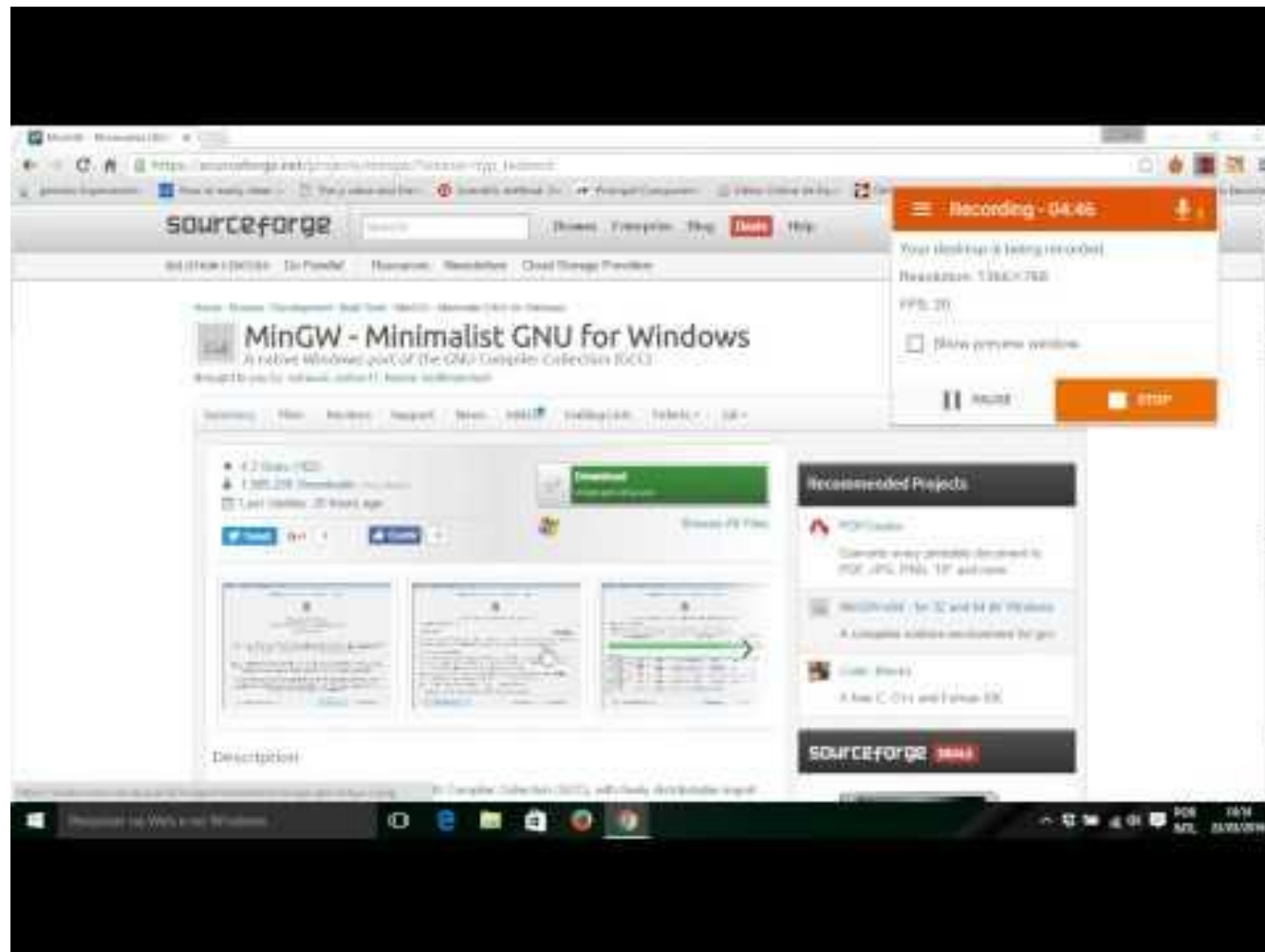
The image shows a screenshot of the Notepad++ text editor. The title bar indicates the file path is 'C:\Users\Pedro\Desktop\gcc-test\seno.c' and the application is 'Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The editor window shows 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 shows 'C source file', 'length: 179 lines: 11', 'Ln: 11 Col: 2 Sel: 0 | 0', 'Dos\Windows', 'ANSI as UTF-8', and 'INS'.

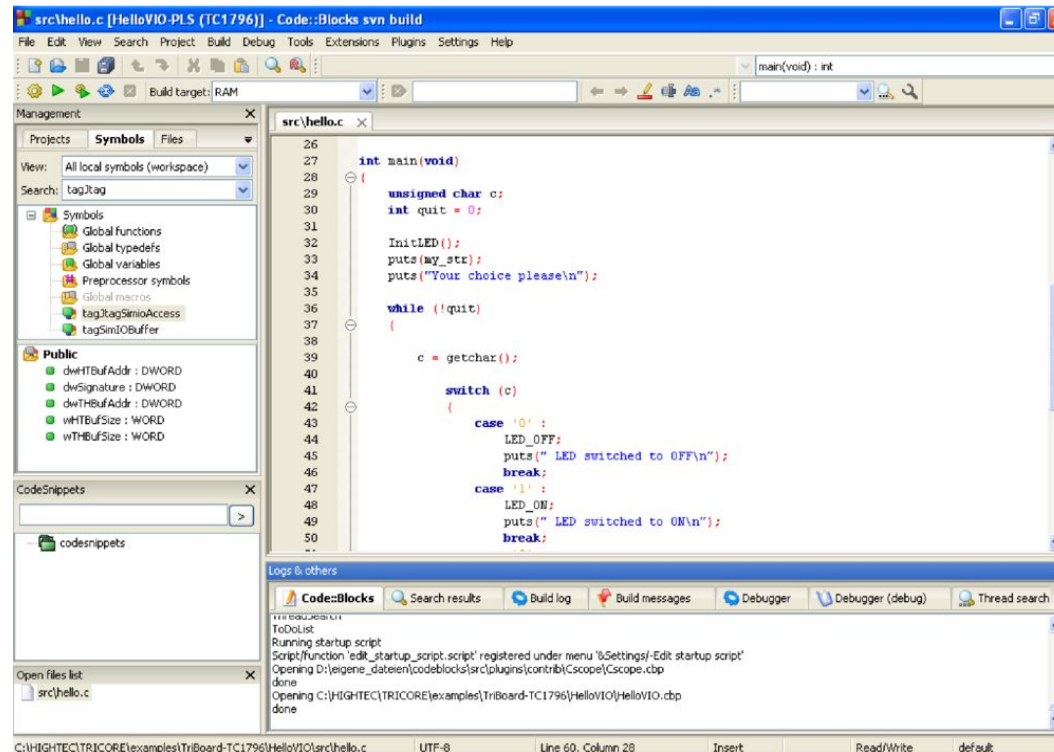
# Editores de texto recomendados (gratuitos)

- Como instalar o GCC no Windows?



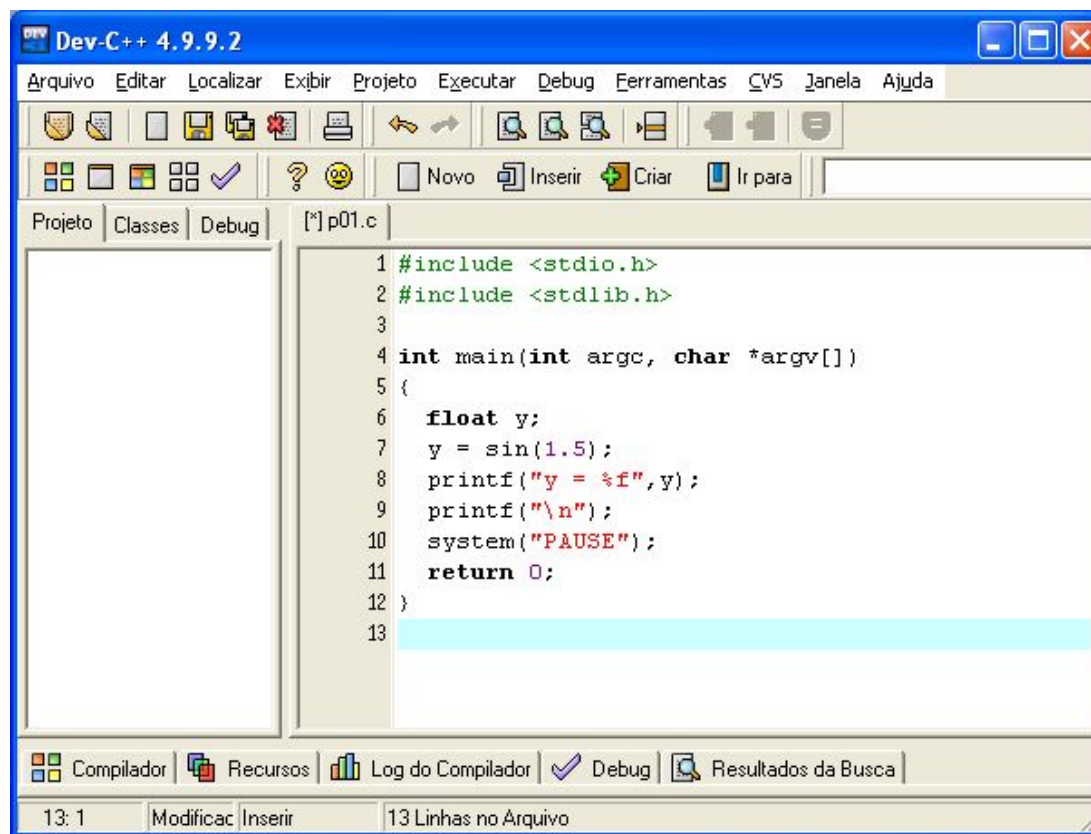
# Definições

- Que ambiente de programação iremos utilizar?
- Pode-se usar o Code Blocks (at your own risk!)



# Definições

- Que ambiente de programação iremos utilizar?
- 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 }
```



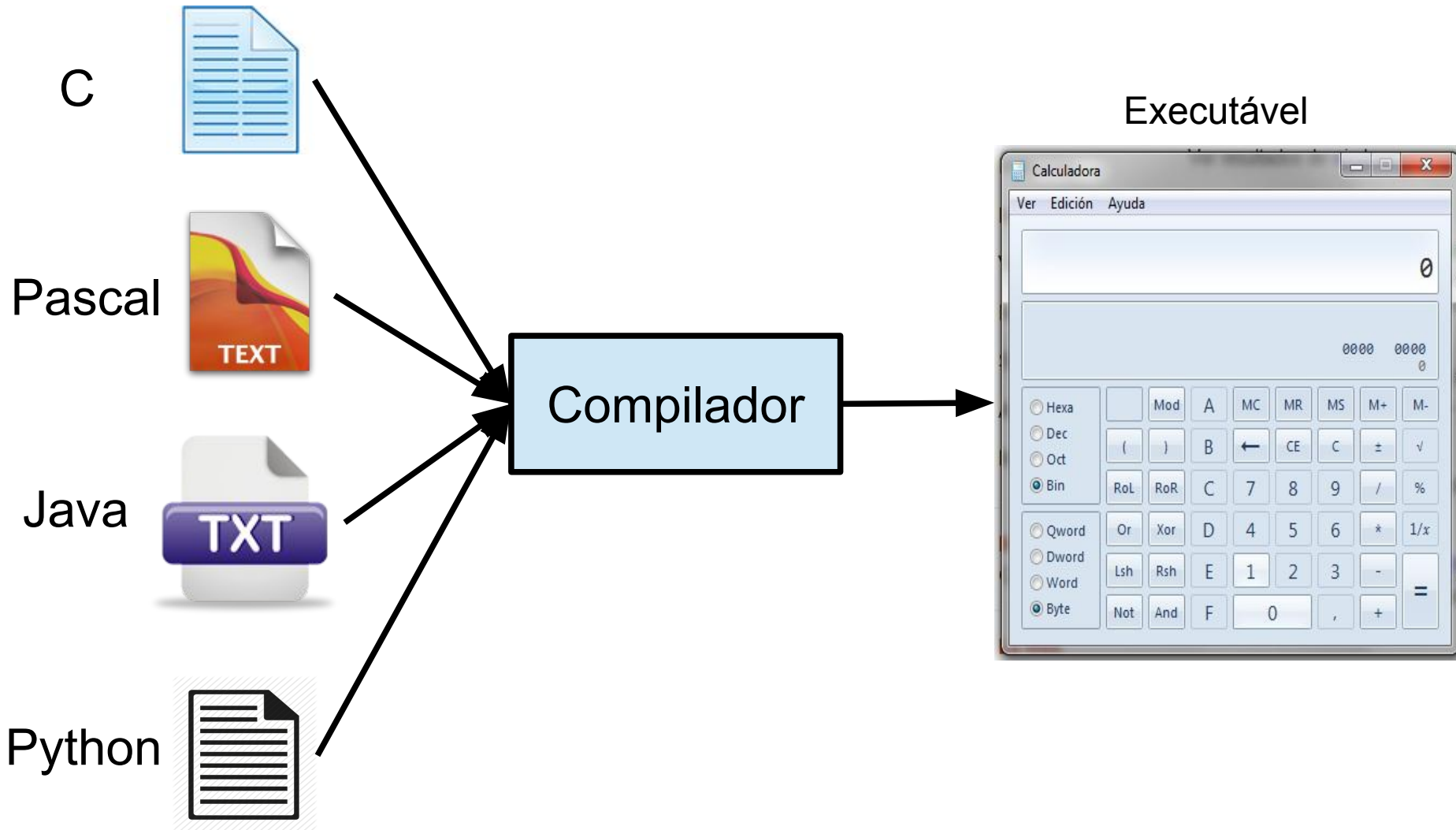
Compilador



```
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:

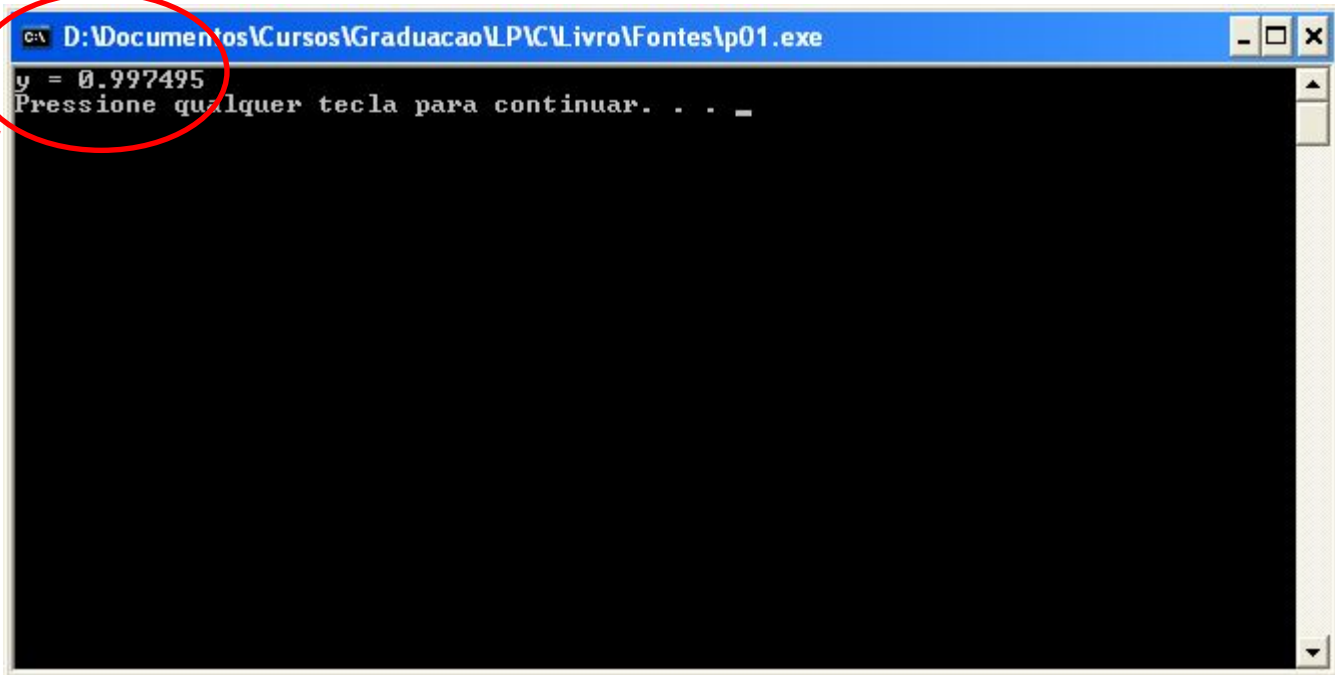
```
printf("y = %f, y);
```

- 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!

# 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 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

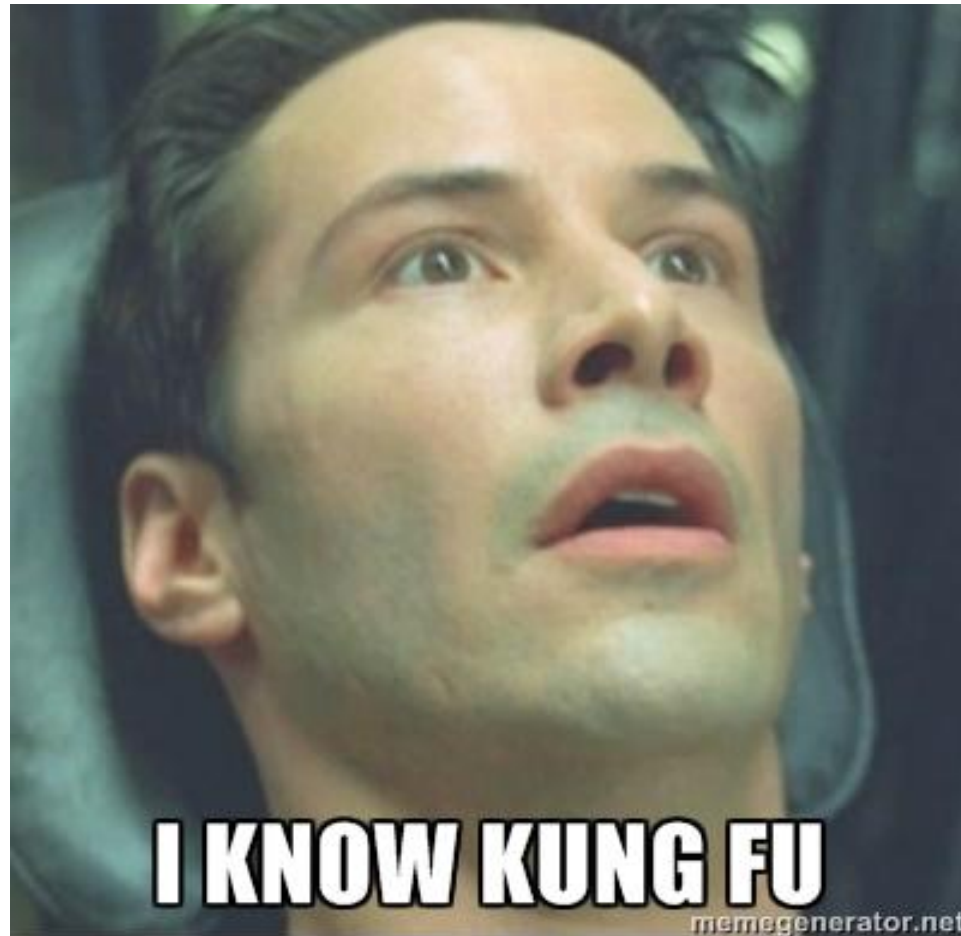
- Note que o programa-fonte p1.c começa com as linhas:

```
#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>
```

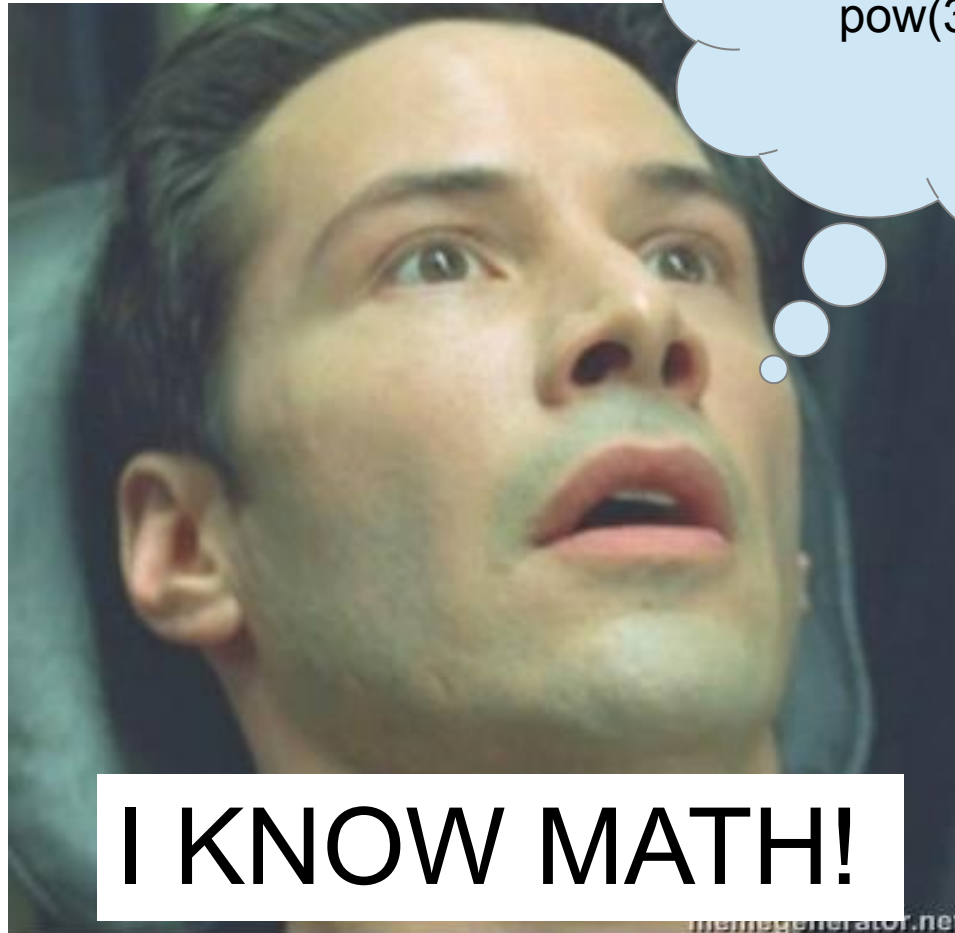




# Arquivos de cabeçalho

```
#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

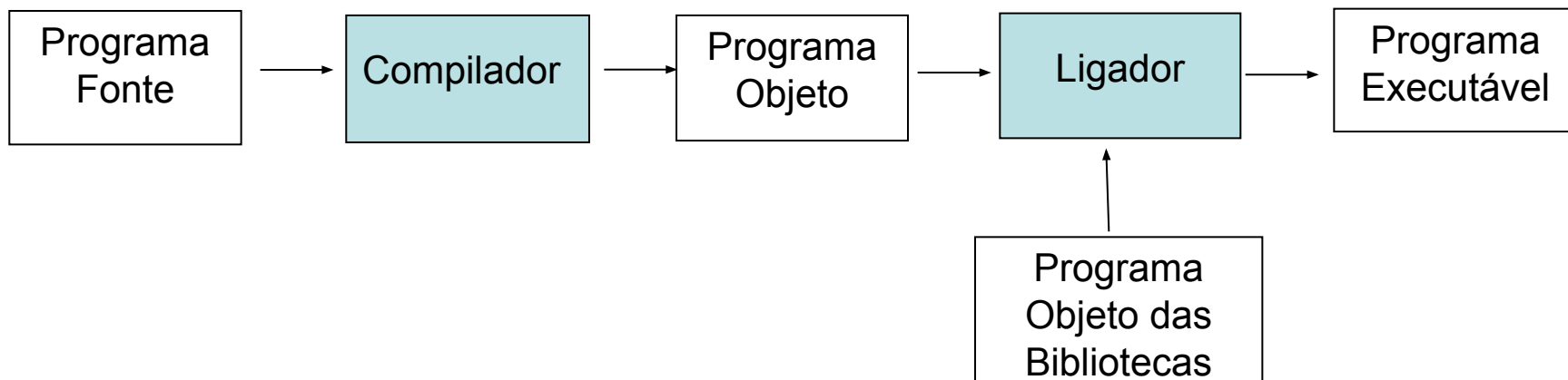
- Ao encontrar uma instrução `#include` em um programa-fonte, o compilador traduz este texto da mesma forma que o faria se o texto tivesse sido digitado no programa-fonte.
- Portanto, as linhas:

```
#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.

# 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



# Notação Decimal

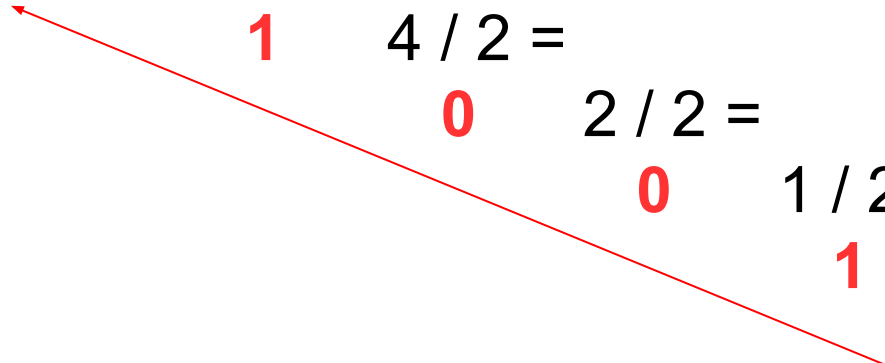
- **19.625** =  
**1** $\times 10^1$  + **9** $\times 10^0$  + **6** $\times 10^{-1}$  + **2** $\times 10^{-2}$  + **5** $\times 10^{-3}$  =  
 $10 + 9 + 0.6 + 0.02 + 0.005 = 19.625$

# Notação Binária

- **10011.101** =  
 $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} =$   
 $16 + 0 + 0 + 2 + 1 + 0.5 + 0 + 0.125 = 19.625$

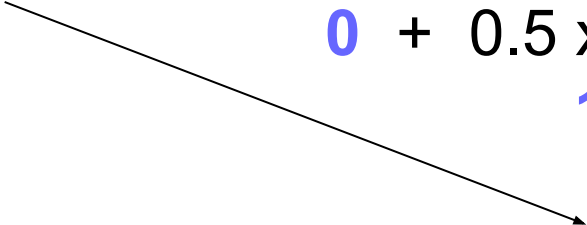
- $19 / 2 =$

**1**      $9 / 2 =$   
      **1**      $4 / 2 =$   
              **0**      $2 / 2 =$   
                  **0**      $1 / 2 =$   
                      **1**     0

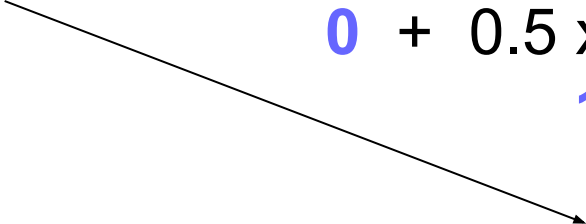


Condição de parada

# Notação Binária

- $0.625 \times 2 =$   
   $\textcolor{blue}{1} + 0.25 \times 2 =$   
     $\textcolor{blue}{0} + 0.5 \times 2 =$   
       $\textcolor{blue}{1} + 0.0$   
                                   $\textcolor{blue}{0.101}$
- 

# 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...

# 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 **sinal-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**).

<u>Decimal</u>	<u>Binário</u>
----------------	----------------

+5	00101
----	-------

-3	10011
----	-------

Desvantagens:

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

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

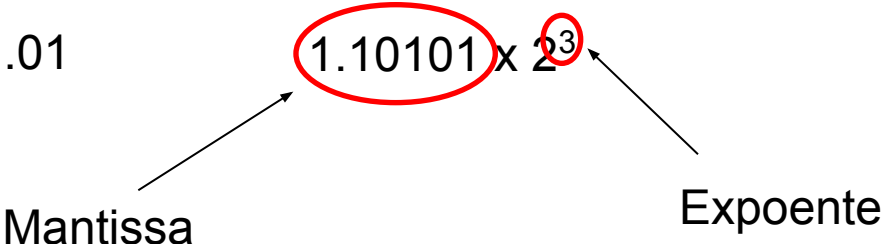
← Que corresponde ao número +2!

# 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**).

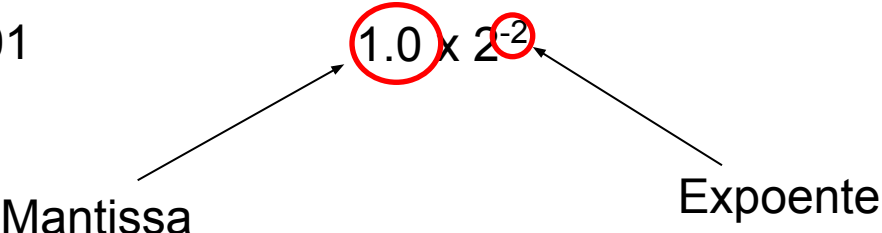
# Números de ponto flutuante

- 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:

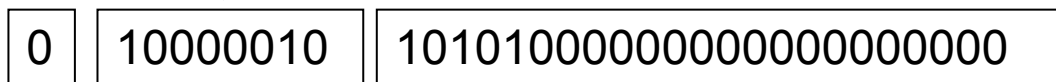
127 é o novo 0

130 é o novo 3

Ponto flutuante

$1.10101 \times 2^3$

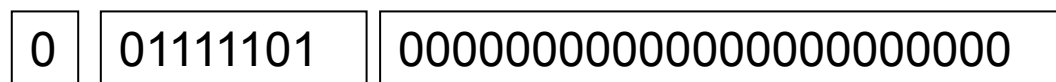
Representação SP



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 }
```

# 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)**.

# Escrevendo um programa em C

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

```
void main() {  
    //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:

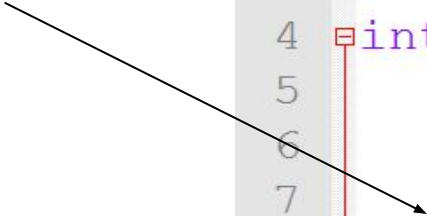
```
#include <stdio.h>
```

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

# 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 algumas **tags** que representam valores, conhecidas como **especificadores de formato**.

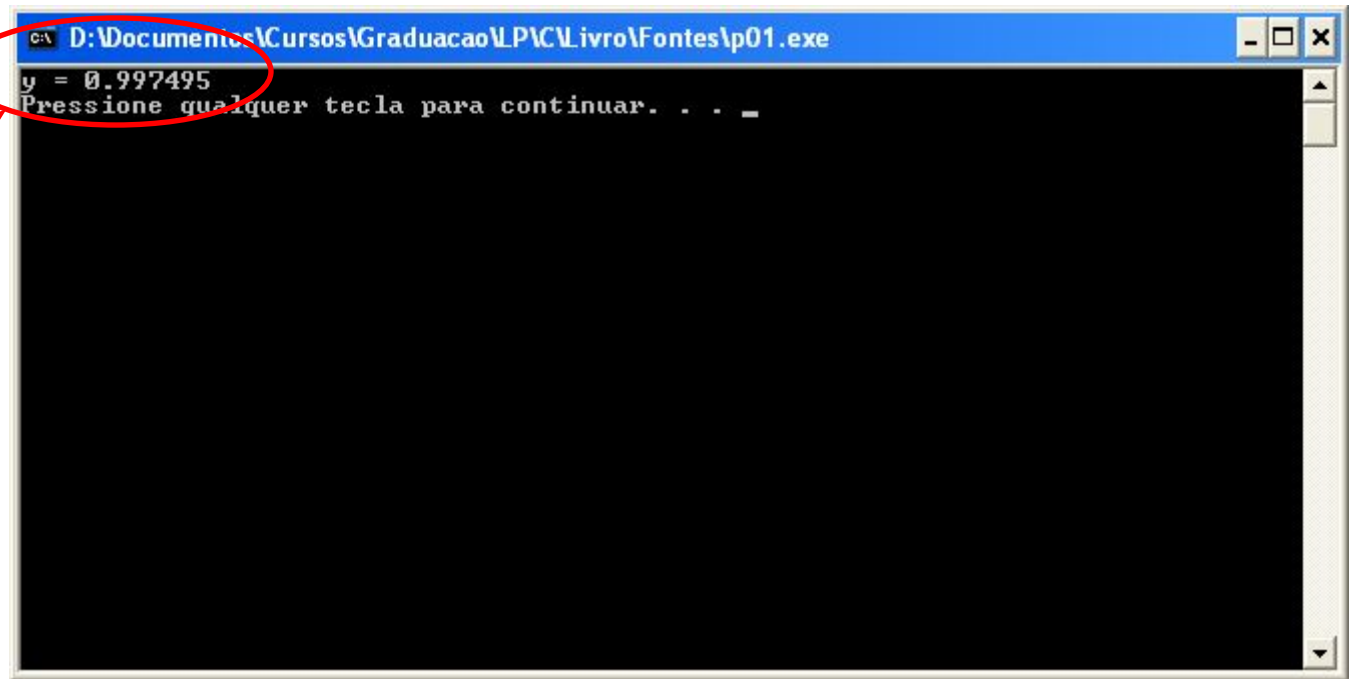
```
printf("y = %f", y);  
printf("\n");
```

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:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\ D:\Documentos\Cursos\Graduacao\LP\CLivro\Fontes\lp01.exe". The command prompt area is black with white text. The first line of output is "y = 0.997495". The second line is "Pressione qualquer tecla para continuar. . . \_". A red circle is drawn around the first line of output, and a red arrow points from the text "Valor armazenado em y." to the circle.

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

Valor  
armazenado  
em *y*.

# 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

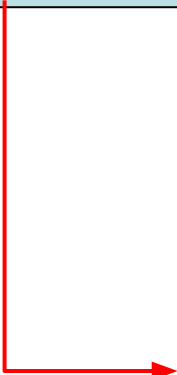
- A linguagem C utiliza o símbolo `\` (barra invertida) para especificar alguns caracteres especiais:

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 }
```

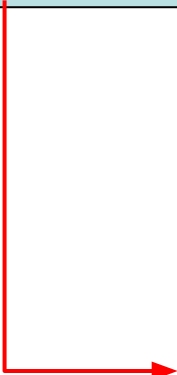
- Ela exibe “o caractere (invisível) de nova linha”. Qual o efeito disso? Provoca uma mudança de linha! Próxima mensagem será na próxima linha.



# 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?