

Programação de Computadores

Aula 02

Problema 1

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

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

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

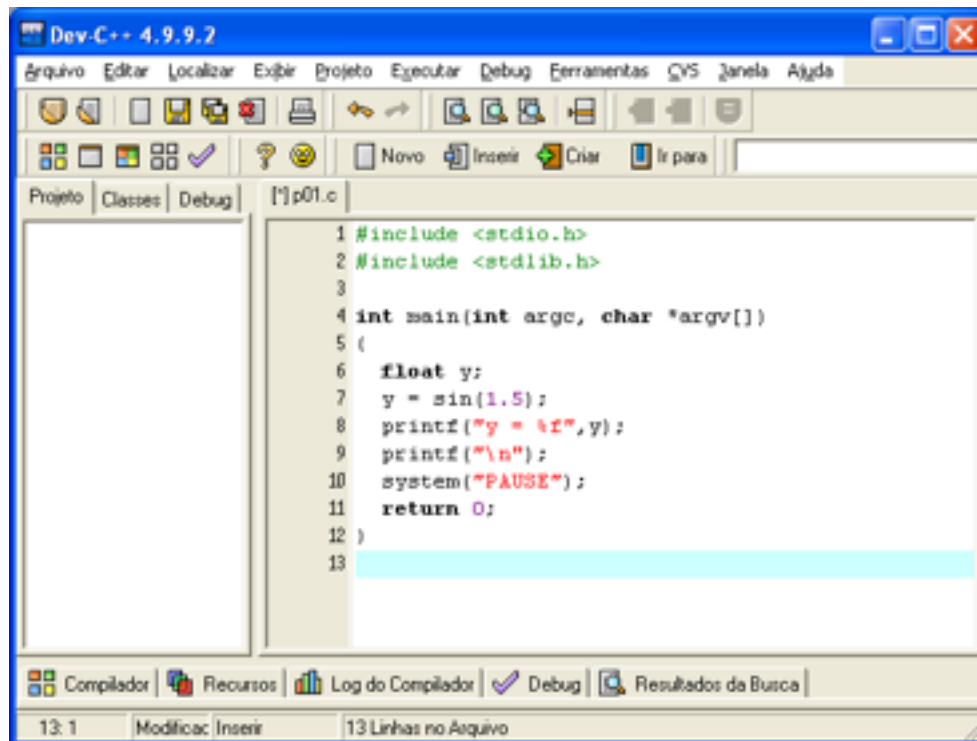
- Neste curso, será utilizada a **linguagem C**.
- A **linguagem C** é subconjunto da **linguagem C++** e, por isso, geralmente, os **ambientes de programação** da linguagem C são denominados ambientes C/C++.
- Um **ambiente de programação** contém:
 - Editor de programas: viabiliza a escrita do programa.
 - 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 e DEV-C++.
- Será utilizado neste curso o DEV-C++.



Definições

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

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

```
int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```



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**.

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:

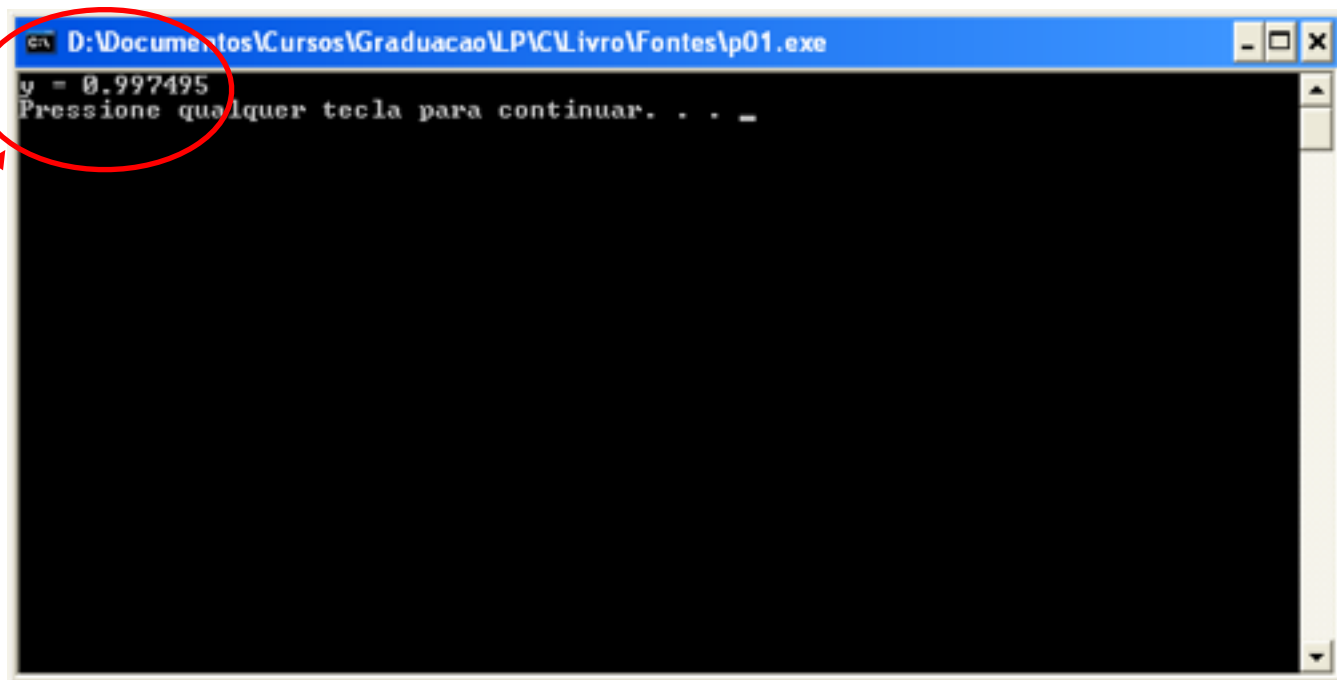
```
printf("y = %f, y);
```

- 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`.
- Executando-se o programa `p1.exe`, o resultado será:

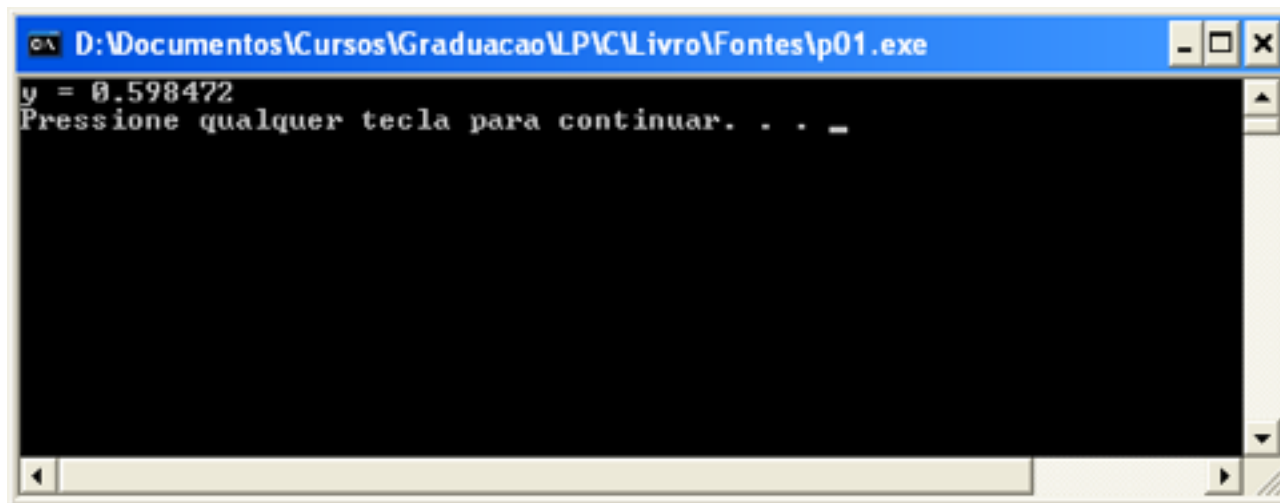


```
D:\Documentos\Cursos\Graduacao\LP1C\Livro\Fontes\p01.exe
y = 0.997495
Pressione qualquer tecla para continuar. . . _
```

**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 is blue and contains the text "D:\Documentos\Cursos\Graduacao\LPIC\Livro\Fontes\p01.exe". The window has standard minimize, maximize, and close buttons. The main area is black with white text. The first line shows "y = 0.598472". The second line shows "Pressione qualquer tecla para continuar. . . _". The window has a scroll bar on the right and a status bar at the bottom.

```
D:\Documentos\Cursos\Graduacao\LPIC\Livro\Fontes\p01.exe
y = 0.598472
Pressione qualquer tecla para continuar. . . _
```

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:
Todo programa-fonte em linguagem C começa com linhas deste tipo.

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

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

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

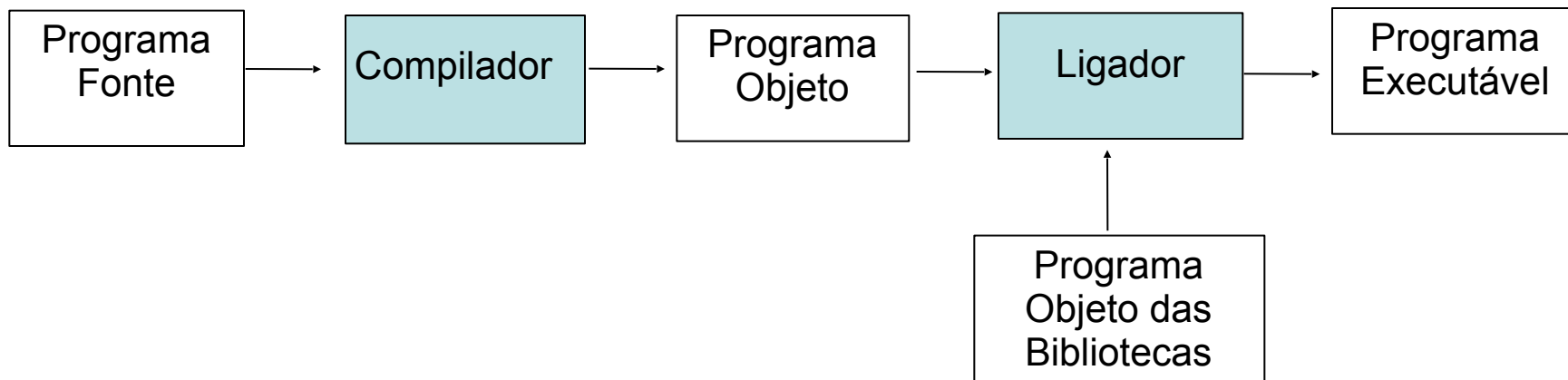
- Ao encontrar uma instrução `#include` em um programa-fonte, o compilador traduz este texto da mesma forma que o faria se o texto tivesse sido digitado no programa-fonte.
- Portanto, as linhas:

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

Conversão de números

- Do sistema decimal para o sistema binário.
- Considere um número real x na base 10 (sistema decimal), representado como a seguir:

$$x = a_m a_{m-1} \dots a_1 a_0, b_1 b_2 \dots b_n$$

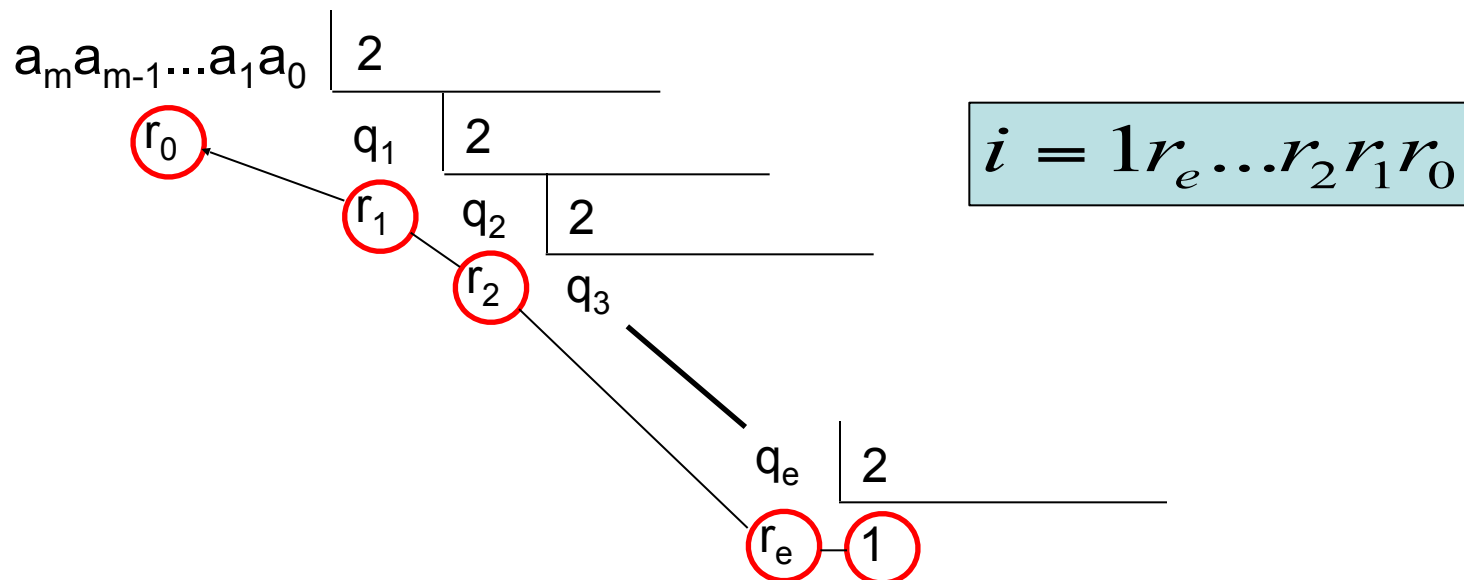
- Onde: a_i ($i = 0, 1, 2, \dots, m$) e b_j ($j = 1, 2, \dots, n$) são elementos do conjunto:

$$A = \{k \in \mathbb{N}; 0 \leq k \leq 9\}$$

- Exemplo: 12385,328 ($m=4$, $n=3$)

Conversão de números

- Considere i e f como sendo as **partes inteira** e **fracionária**, respectivamente, de um número x representado na base 2.
- Passo 1: determinação de i .
 - Divide-se sucessivamente a parte inteira de x na base 10 por 2, até que o quociente seja igual a 1.



Conversão de números

- Passo 2: determinação de f .
 1. Multiplica-se a parte fracionária de x na base 10 por 2.
 2. Desse resultado, toma-se a parte inteira como sendo o primeiro dígito de f na base binária.
 3. A parte fracionária oriunda da primeira multiplicação é novamente multiplicada por 2.
 4. Do resultado obtido, toma-se a parte inteira como segundo dígito de f e assim por diante.
 5. Quando parar?
 - Quando a parte fracionária do último produto seja igual a zero ou até que se observe o aparecimento de uma dízima periódica.

Conversão de números

- Passo 2: determinação de f .
- Esquema do cálculo:

$$\begin{array}{ccc}
 0, b_1 b_2 \dots b_n & 0, c_1 c_2 \dots c_n & 0, z_1 z_2 \dots z_n \\
 \hline
 x & x & x \\
 \hline
 & 2 & 2 \\
 \hline
 f_1, c_1 c_2 \dots c_n & f_2, d_1 d_2 \dots d_n & f_k, 00 \dots 0
 \end{array}$$

- Logo:

$$f = 0, f_1 f_2 \dots f_k$$

- Finalmente, soma-se i com f para obter o número na base 2 equivalente ao dado na base 10.

$$x = (a_m a_{m-1} \dots a_1 a_0, b_1 b_2 \dots b_n)_{10} = (1r_e \dots r_1 r_0, f_1 f_2 \dots f_k)_2$$

Conversão de números

- Exemplo: converter o número $(23,625)_{10}$ para a base 2.

1. Determinação de i :

$$i = 10111$$

23	2			
1	11	2		
	1	5	2	
		1	2	2
			0	1

2. Determinação de f :

0,625	0,250
x 2	x 2
1,250	0,500

$$f = 101$$

0,500
x 2
1,000

Logo:

$$(10111,101)_2$$

Conversão de números

- Atenção!
 - Um número finito na base decimal pode ser igual a um número binário infinito!!!
 - Exemplo: $(0,6)_{10} = (0,1001\ 1001\dots)_2$



Conversão de números

- Do sistema binário para o sistema decimal.
- Considere um número real x na base 2 (sistema binário), representado como a seguir:

$$x = a_m a_{m-1} \dots a_1 a_0, b_1 b_2 \dots b_n$$

- Onde: a_i ($i = 0, 1, 2, \dots, m$) e b_j ($j = 1, 2, \dots, n$) são elementos do conjunto:

$$A = \{0, 1\}$$

- Exemplo: 10111,101 ($m=4, n=3$)

Conversão de números

- Passos:

1. Dado o número binário:

$$x = a_m a_{m-1} \dots a_1 a_0, b_1 b_2 \dots b_n$$

Expressa-se o número como:

$$x = a_m 2^m + a_{m-1} 2^{m-1} + \dots + a_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + \dots + b_n 2^{-n}$$

2. Realiza-se as operações aritméticas na expressão.

- Exemplo:

$$(10111,101)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = (23,625)_{10}$$

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:

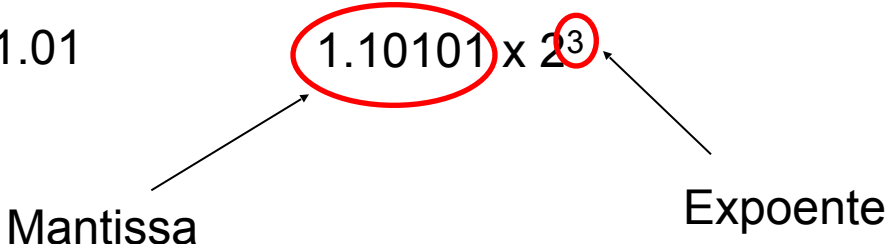
00101	
11101	
00010	← Que corresponde ao número +2!

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).

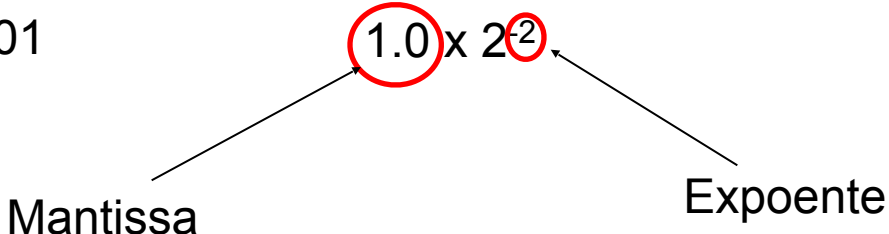
Números de ponto flutuante

- 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	1.10101×2^3
		

Números de ponto flutuante

- Exemplo 2:

<u>Decimal</u>	<u>Binário</u>	<u>Binário normalizado</u>
+0.25	0.01	1.0×2^{-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

0	10000010	101010000000000000000000
---	----------	--------------------------

Ponto flutuante

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

Representação SP

0	01111011	000000000000000000000000
---	----------	--------------------------

- 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 ∞/∞

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).

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

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](#).
 - Por exemplo:

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

Variáveis

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

8 1000	3.25 1001	'a' 1002	'g' 1003
'q' 1004	2 1005	'*' 1006	'1' 1007
1008	1009	1010	1011
1012	1013	1014	1015
1016	1017	1018	1019

← Memória

Variáveis

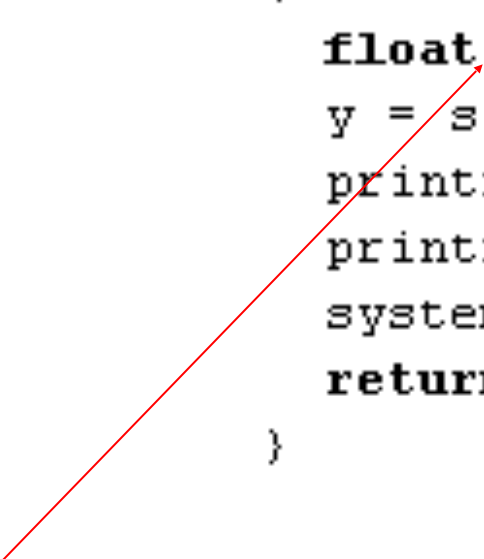
- A partir dos endereços, é possível para o computador saber qual é o valor armazenado em cada uma das posições de memória.
- Como a **memória pode ter bilhões de posições**, é difícil controlar em qual endereço está armazenado um determinado valor!
- Para facilitar o controle sobre onde armazenar informação,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

Variáveis

- Exemplo de variável:

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```



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.

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 abrev-chaves **{** e termina com fecha-chaves **}**.

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

Corpo da
função



Escrevendo um programa em C

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

```
float y;
```

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

```
#include <stdlib.h>
```

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

```
{
```

```
    float y;
```

```
    y = sin(1.5);
```

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

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

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

```
    return 0;
```

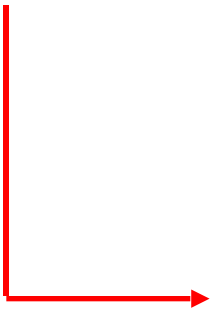
```
}
```

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**:

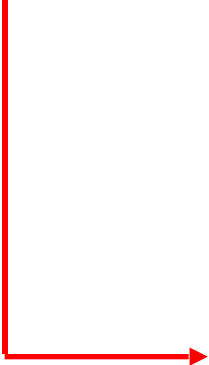


```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

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`.)



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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```


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

- Exemplo:

- Considere as seguintes declarações:

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

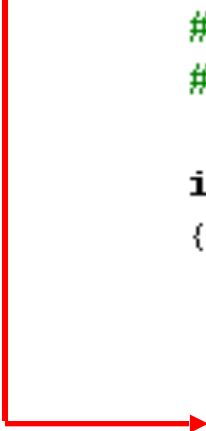
- Neste caso, teremos:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	20
$b = (1 - 4) / (2 - 5)$	1.0
$a = 2.75 + 1.12$	3
$b = a / 2.5$	1.2

Escrevendo um programa em C

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

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



```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(int argc, char *argv[])  
{  
    float y;  
    y = sin(1.5);  
    printf("y = %f", y);  
    printf("\n");  
    system("PAUSE");  
    return 0;  
}
```

- 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**.

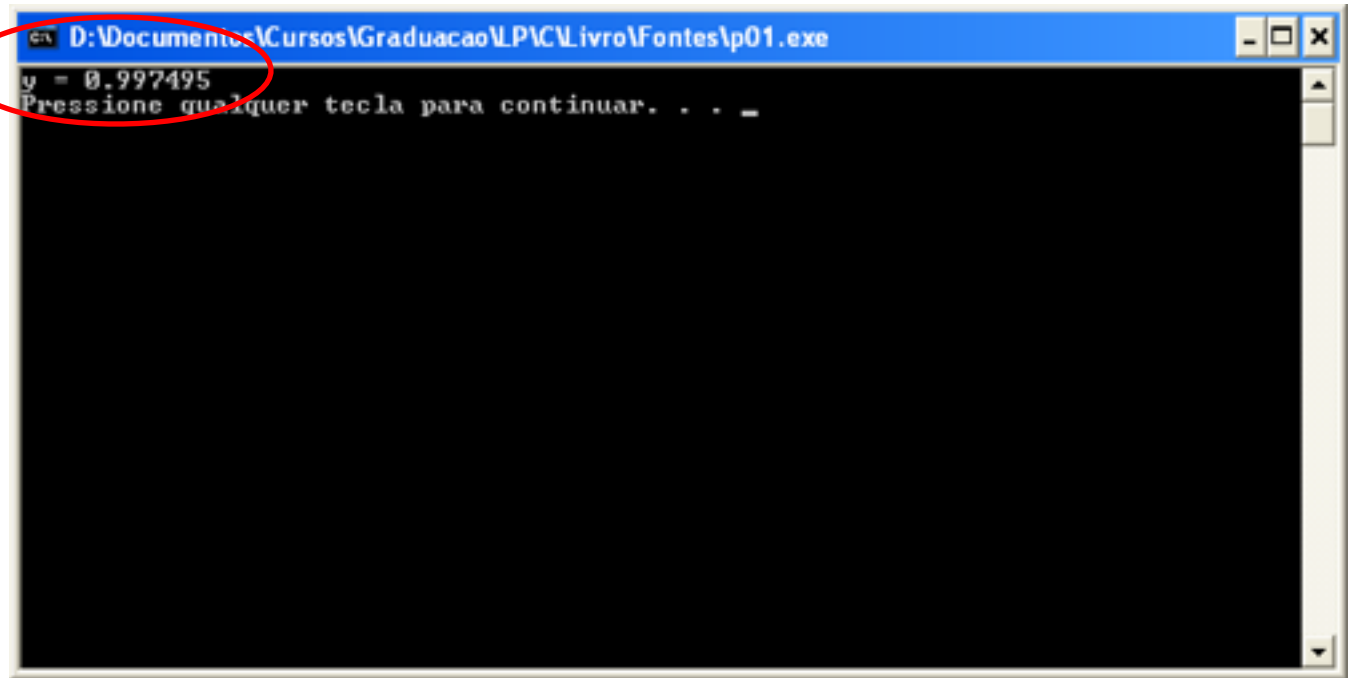
```
printf("y = %f", y);  
printf("\n");
```

**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:



```
D:\Documentos\Cursos\Graduacao\LPVC\Livro\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));
```

- 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

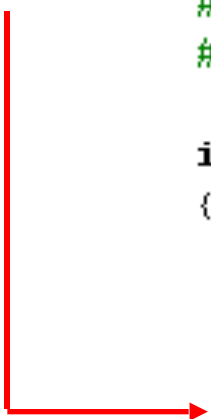
Escrevendo um programa em C

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

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

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```



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

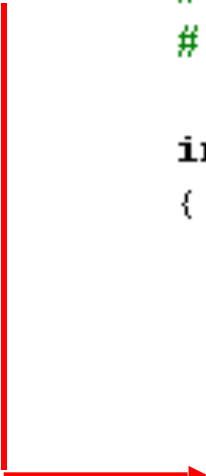
Escrevendo um programa em C

- Observe agora a próxima linha do programa:

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

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

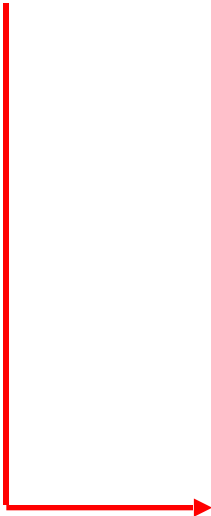


- Ela exibe a mensagem “Pressione qualquer tecla para continuar...” e interrompe a execução do programa.

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;
```



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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

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.

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

```
→ int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

- 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`:

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

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

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

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


Problema 2

- Uma conta poupança foi aberta com um depósito de R \$500,00. Esta conta é remunerada em 1% de juros ao mês. Qual será o valor da conta após três meses?

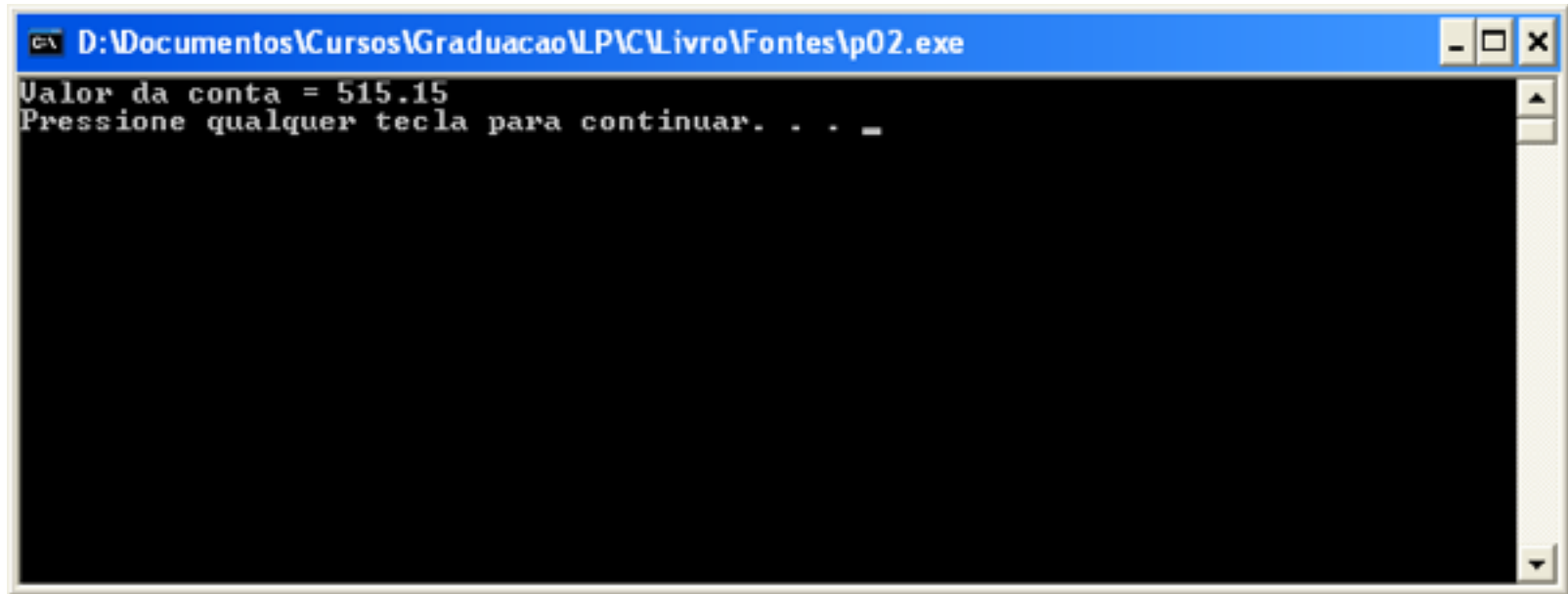
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    float d,p,s,t;

    d = 500;    // depósito inicial
    // após o primeiro mês
    p = d + 0.01*d;
    // após o segundo mês
    s = p + 0.01*p;
    // após o terceiro mês
    t = s + 0.01*s;
    printf("Valor da conta = %.2f\n",t);
    system("pause");
    return 0;
}
```

Problema 2

- Executando-se o programa, obtém-se:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "D:\Documentos\Cursos\Graduacao\LPIC\livro\Fontes\p02.exe" along with standard window control buttons (minimize, maximize, close). The main area of the window is black with white text. The text displayed is "Valor da conta = 515.15" followed by "Pressione qualquer tecla para continuar. . . _" on the next line. A small cursor is visible at the end of the second line.

```
D:\Documentos\Cursos\Graduacao\LPIC\livro\Fontes\p02.exe
Valor da conta = 515.15
Pressione qualquer tecla para continuar. . . _
```

- Após 3 meses: R\$ 515,15

Problema 2

- No programa `p2.c`, note que o tag usado na função `printf` é `%.2f` em vez de `%f`.
- Neste caso, o especificador de formato inclui também o **número de dígitos** desejados após o “ponto decimal”.

Atenção!

- É de extrema importância o uso de **ponto-e-vírgula** após cada instrução.
- Com os pontos-e-vírgulas, o compilador sabe exatamente onde termina cada uma das instruções.