

Programação e Desenvolvimento de Software I

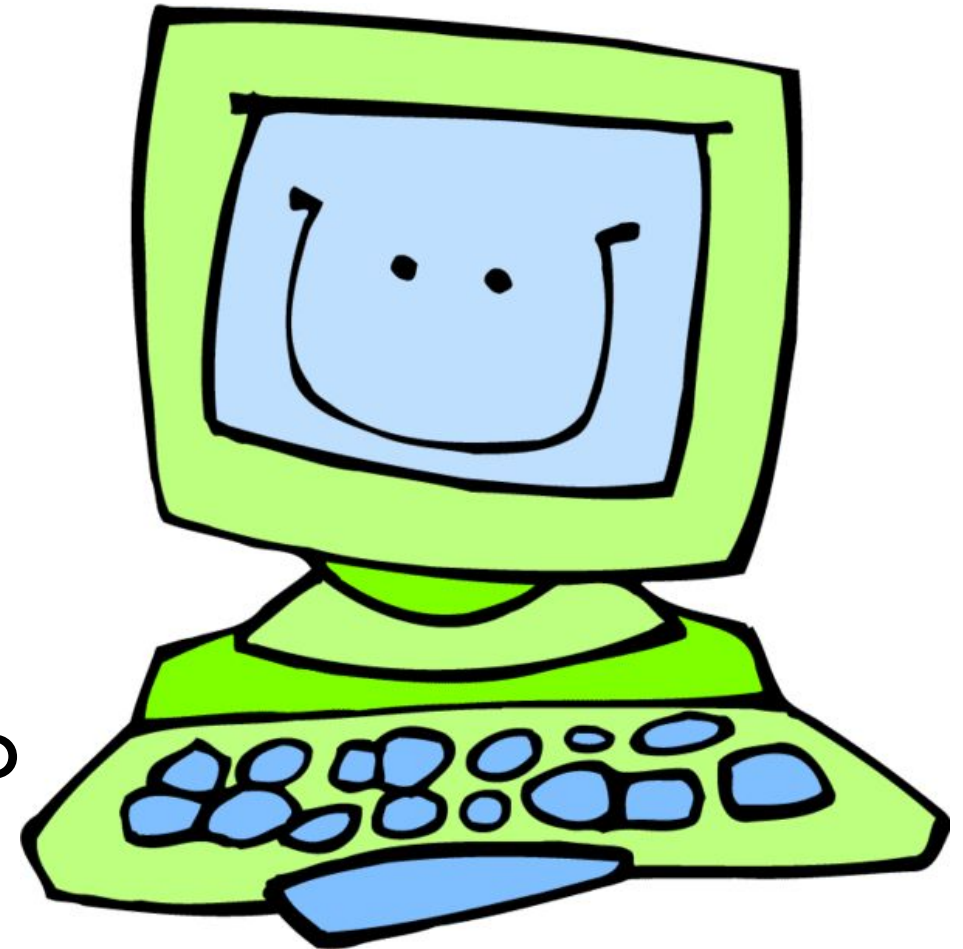
Tipos de dados

Prof. Héctor Azpúrua
(slides adaptados do Prof. Pedro Olmo)

Resumo da aula anterior

Computadores

- Por que usar um computador?
 - Permite que realizemos contas matemáticas muito rapidamente!
 - Capacidade de armazenamento:
 - Videos, musicas, livros, jogos...
 - Comunicação!
 - Uso da internet, redes sociais...
- Usar os computadores para o nosso beneficio!



Resumo da aula anterior

Introdução

- Qual é a dificuldade?
 - Pessoas, conseguem abstrair e entender ideias de alto nível
 - Tradicionalmente falando: Computadores NÃO!
 - Vamos excluir ChatGPT e afins da conversa... 😊
- Computadores são literais:
 - Precisam de uma serie de passos
 - Passos bem especificados
 - Para conseguir realizar uma tarefa qualquer

Resumo da aula anterior

Introdução

■ Problema I

- 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**
- Para resolver o Problema I, **precisamos de um algoritmo:**

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

Resumo da aula anterior

Algoritmos estruturados

- Pode-se escrever este algoritmo como:

1. Sejam A e B os valores dados;
2. Atribuir o valor 0 ao quociente (q);
3. Enquanto $B \leq A$:
4. {
5. Somar 1 ao valor de q ;
6. Subtrair B do valor de A ;
7. }
8. Atribuir o valor final de A ao resto (r);

Resumo da aula anterior

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



Resumo da aula anterior

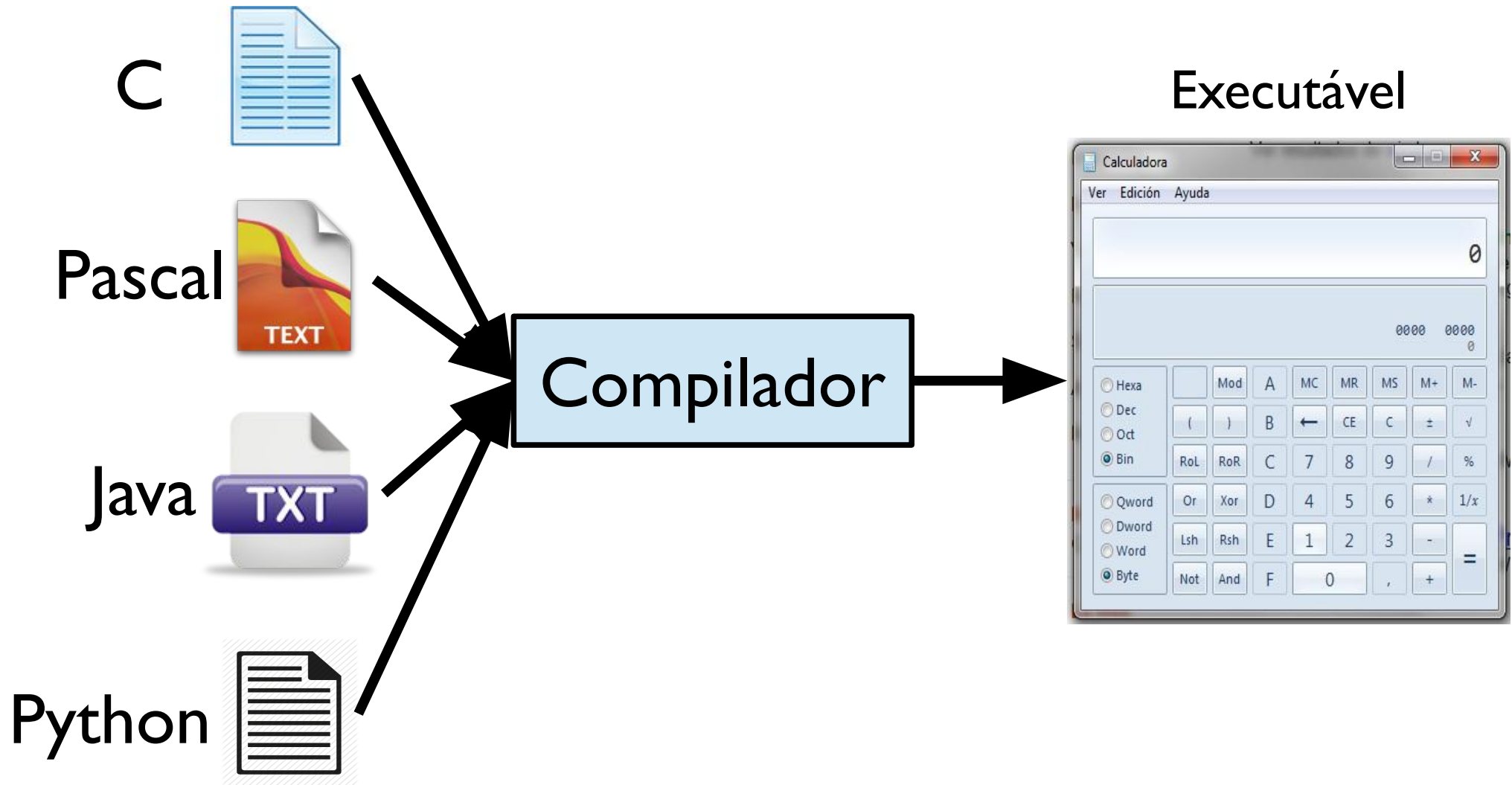
Definições

- 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

Resumo da aula anterior

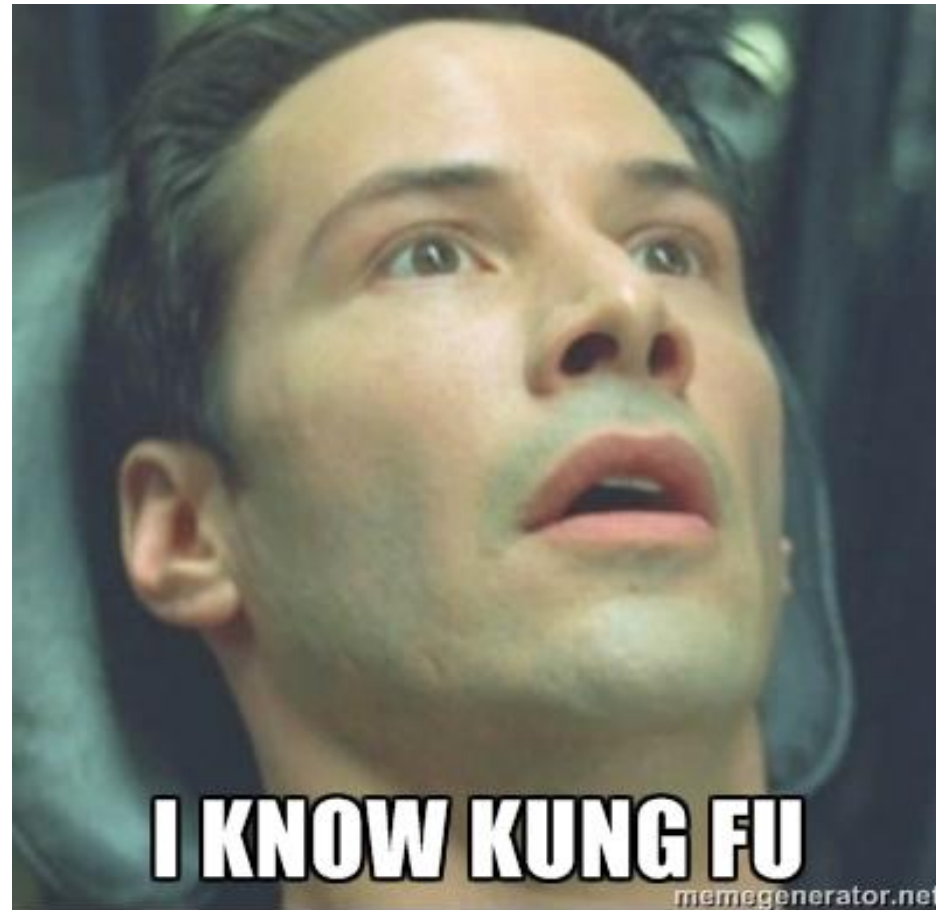
Definições



Resumo da aula anterior

Arquivos de cabeçalho podem conter habilidades novas!

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



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

Notações

Decimal vs Binária

Decimal
Números do 0 ... 9
$0 + 1 = 1$
$1 + 1 = 2$
$2 + 1 = 3$
...
$9 + 1 = 10$
...
$99 + 1 = 100$

Binária
Números 0 e 1
$0 + 1 = 1$
$1 + 1 = 10$
$10 + 1 = 11$
$11 + 1 = 100$

Notações

Decimal vs Binaria

Decimal	Binary	Octal	Hex
0	0	0	0x0
1	1	01	0x1
2	10	02	0x2
3	11	03	0x3
4	100	04	0x4
5	101	05	0x5
6	110	06	0x6
7	111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF
16	1 0000	020	0x10
17	1 0001	021	0x11
18	1 0010	022	0x12
19	1 0011	023	0x13
20	1 0100	024	0x14

Conversor online

<https://www.rapidtables.com/convert/number/decimal-to-binary.html>

Armazenamento no computador



Endereço de memória

Palavra de **16 bits**

	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 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 sequência 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

Notação decimal

Base 10

■ $19.625 =$

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

■ $10 + 9 + 0.6 + 0.02 + 0.005$

Notação binária

Base 2

- $10011.101 =$

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

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

$$19 / 2 = 9.5$$

$$1 \quad 9 / 2 = 4.5$$

$$1 \quad 4 / 2 = 2$$

$$0 \quad 2 / 2 = 1$$

$$0 \quad 1 / 2 = 0$$

$$1 \quad 0$$

Representação binária

é então os restos na direção inversa

na qual eles aparecem: **10011**

Como converter a parte inteira em binária?

Dividimos sucessivamente entre 2,
registramos o resto (0 ou 1) e pegamos
o cociente inteiro para dividir entre 2

Condição de parada!

Quando o cociente é 0

Notação binária

- $10011.101 = 19.625$

$$0.625 \times 2 = 1.25$$

$$1 + 0.25 \times 2 = 0.5$$

$$0 + 0.5 \times 2 = 1.0$$

$$1 + 0.0$$

Representação binária

é então os números inteiros na direção na qual eles aparecem: **101**

- E agora com 0.6 ?
- Preciso de quantos bits depois do “.”?

Como converter a parte fraccionaria em binaria?
Multiplicamos sucessivamente por 2,
armazenamos a parte inteira da multiplicação e
multiplicamos por 2 a parte fraccionaria

Condição de parada quanto a parte
fraccionaria da zero!

Notação binária

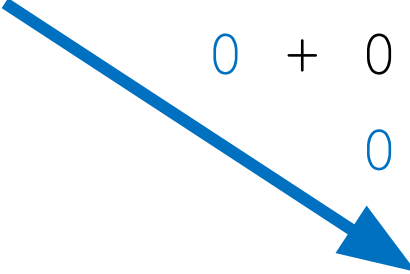
- $0.6 \times 2 = 1.2$

1 + 0.2 $\times 2 = 0.4$

0 + 0.4 $\times 2 = 0.8$

0 + 0.8 $\times 2 = 1.6$

1 + 0.6 $\times 2 = \dots$



0.100110011001...

- Dízima periódica! O que isso significa?
 - Não há bits suficientes no computador para representar 0.6
 - Vamos ver varias esquisitices como essas no curso... [Imprecisões de ponto flutuante](#)
 - O que fazer?

Armazenamento no computador



Endereço de memória

Palavra de **16 bits**

	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

Em cada um desses endereços precisamos guardar letras, números inteiros, números negativos, números fraccionários... Como fazer?

Representação de números inteiros

Sinal-magnitude

- 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**)

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	1 0000
17	1 0001
18	1 0010
19	1 0011
20	1 0100

<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

Complemento-de-2

- 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	

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	1 0000
17	1 0001
18	1 0010
19	1 0011
20	1 0100

Representação de números inteiros

Complemento-de-2

- Note o que ocorre com o zero:

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

- E a soma?

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

Somando:

00101
11101
00010

Corresponde ao número +2!

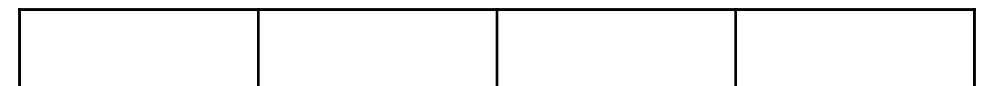
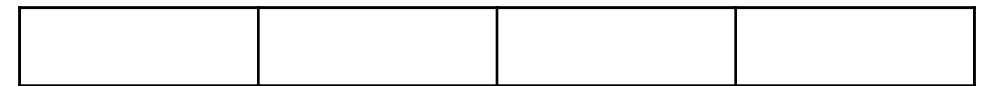
Note que o **vai-um** daqui não é considerado, pois a representação usa apenas 5 bits

<u>Decimal</u>	<u>Binary</u>
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	1 0000
17	1 0001
18	1 0010
19	1 0011
20	1 0100

Números de ponto flutuante

Introdução e erros de precisão

- Algumas simplificações:
 - Não temos números negativos
 - Notação “decimal”
- Problema:
 - Queremos armazenar o numero: 5381
 - Agora o numero: 5381721. Como fazer?
 - Converter para a sua notação ponto flutuante:
 - $538172,1 \times 10^1$
 - $53817,21 \times 10^2$
 - ...
 - $0,5381721 \times 10^7$
 - Com os 4 bits conseguimos então representar algo muito próximo desse grande valor!



Mantissa



Exponente

5380000

Erro de: 1721

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**: 13,25
 - Representação com ponto flutuante: $0,1325 \times 10^2$
 - **Ponto Flutuante** ou **Vírgula Flutuante**? Tanto faz
 - A representação com ponto flutuante segue padrões internacionais (**IEEE-754** e **IEC-559**)

Números de ponto flutuante

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	1 0000
17	1 0001
18	1 0010
19	1 0011
20	1 0100

- 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 $1 . \text{f}$, onde f corresponde aos demais bits
- Ou seja, o primeiro bit sempre é 1
 - Então não precisamos usar esse bit explicitamente

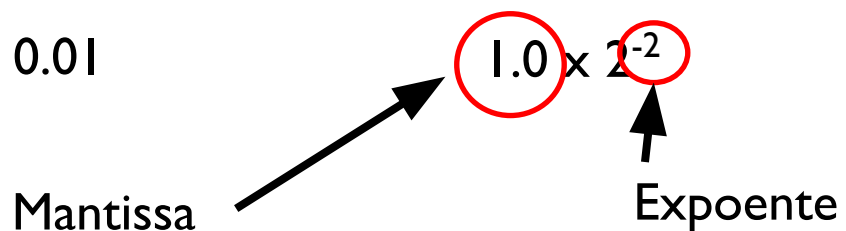
Exemplo 1:

	<u>Decimal</u>	<u>Binário</u>	<u>Binário normalizado</u>
Sinal	$\oplus 13.25$	1101.01	1.10101×2^3
Mantissa			
Expoente			

Números de ponto flutuante

- Exemplo 2:

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



- 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**)
 - 127 é o novo 0
 - 129 é o novo 2

■ Exemplo:

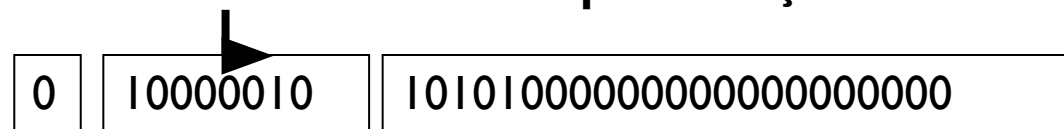
Ponto flutuante

$$1.10101 \times 2^3$$

Ponto flutuante

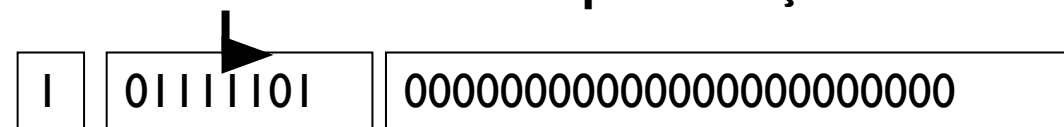
$$1.0 \times 2^{-2}$$

Numero 3 = 130



Representação SP

Numero 2 = 129



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

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

- 126 é o novo -1
- 127 é o novo 0
- 129 é o novo 2
- 130 é o novo 3

Binary	Unsigned	Sign	
		Magnitude	Excess-127
00000000	0	0	-127
00000001	1	1	-126
⋮	⋮	⋮	⋮
01111110	126	126	-1
01111111	127	127	0
10000000	128	-0	1
10000001	129	-1	2
⋮	⋮	⋮	⋮
11111110	254	-126	127
11111111	255	-127	128

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 -127 e 128

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)
- Duplicar o numero de bits **aumenta mais do que o dobro** o quantidade de possíveis números que podem ser representados
- Exemplo:
 - Similar ao abordado para precisão simples...

Escrevendo um programa em C

programa1.c

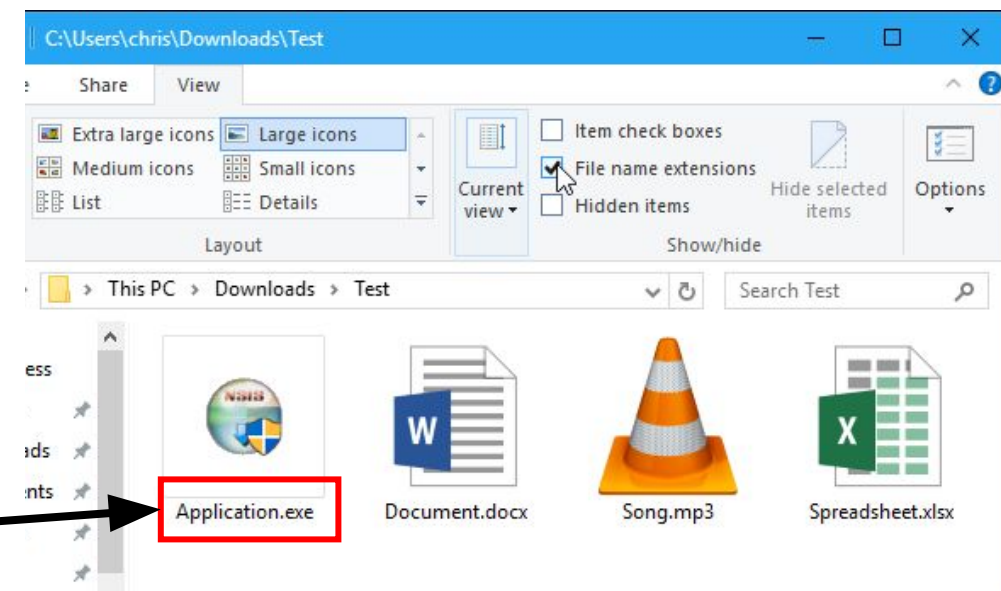
- Escrever um programa em Linguagem C corresponde a escrever o corpo da função principal (`main`)
- Uso de arquivos com terminação `".c"`
- O corpo de uma função sempre começa com abre-chaves `{` e termina com fecha-chaves `}`

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

- O código fonte de um programa em C deve ser do tipo de arquivo “*.c”
- **Formato de arquivo** é a forma usada por determinadas aplicações computacionais para reconhecer os dados gerados e editados por eles:
 - `passaporte.pdf` = Documento PDF
 - `notas.xls` = Documento de Excel
 - `foto123.jpg` = Documento de fotografia
 - `meu_programa.c` = Documento de código fonte de C
 - `meu_programa.exe` = Programa executável!
- Em Linux/Mac os executáveis não precisam de ter terminação “*.exe”



Compilador

gcc



- Qual é o compilador que vamos usar?

gcc

(<https://gcc.gnu.org/>)

- Como usamos o “gcc”?
 - Temos primeiramente que **instalar o programa “gcc”** 😊
 - 1. Depois precisamos criar um arquivo de código fonte em C:
 - Um arquivo como “**meucodigofonte.c**”
 - 2. Compilar o “**meucodigofonte.c**” com “gcc” e criar um novo programa “**programa.exe**”
 - 3. Executar o programa “**programa.exe**”
-
- Você pode fazer tudo isso via o **prompt de comando** ou **terminal!**

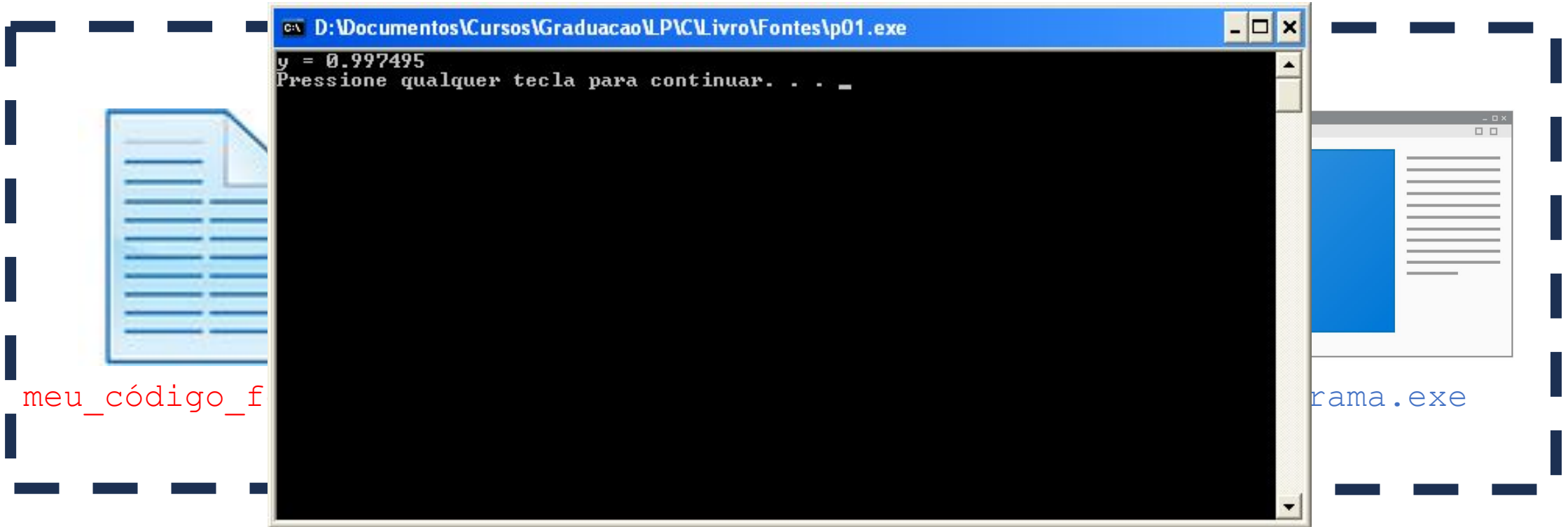
Compilador

gcc

Passo 1: Criamos o nosso código fonte (um arquivo .c) e passamos ele para o compilador

Passo 2: Compilamos o nosso código fonte e criamos um programa usando gcc

`gcc meu_código_fonte.c -o programa.exe`



Passo 3: Executamos o `programa.exe`

Terminal ou prompt de comando

O que é?

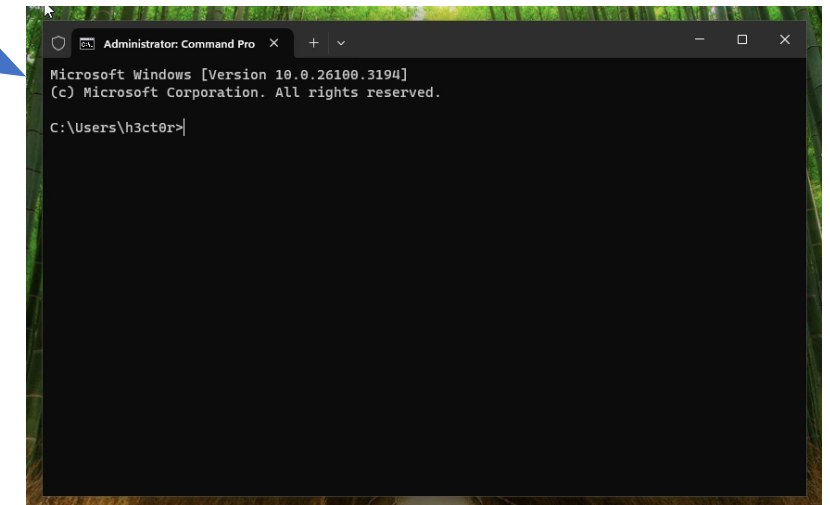
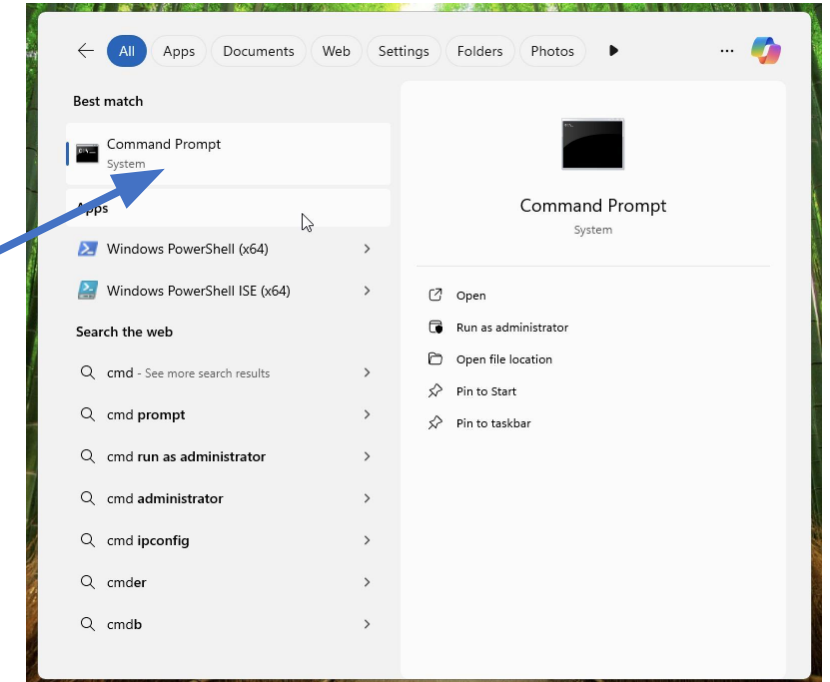
- **Prompt de comando** ou **terminal** é um programa que emula o campo de entrada em uma tela de interface de usuário baseada em texto:
 - Existe no Windows, no Linux e no Mac
- É usado para **executar comandos digitados** e realizar funções administrativas **avançadas**
- Permite que você use programas usando a interface de comandos ou command-line interface (CLI)
- É uma forma antiga de interatuar com o computador!
 - Mas funciona muito bem, e é mais rápido que a interface gráfica



Terminal ou prompt de comando

Como usar o terminal?

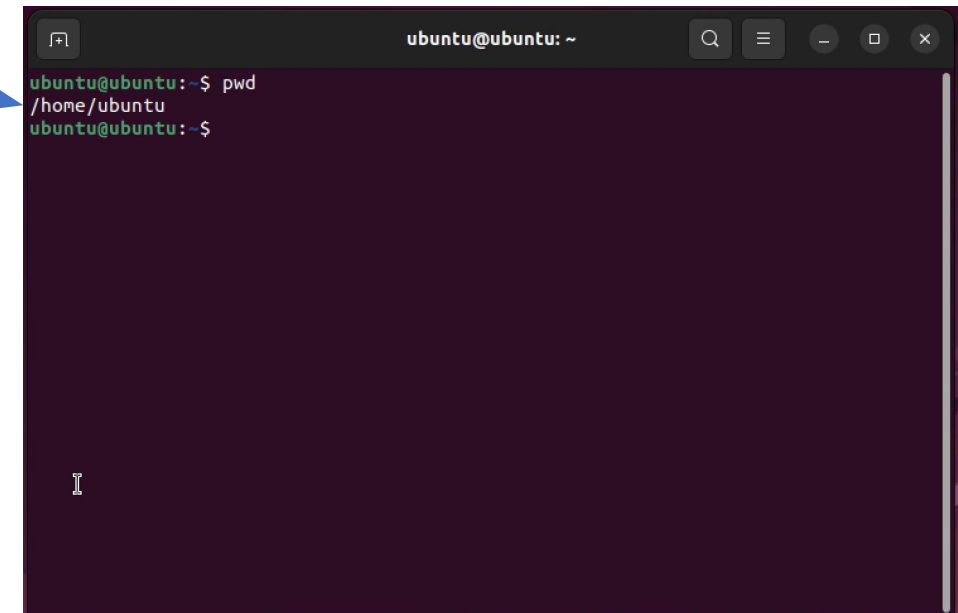
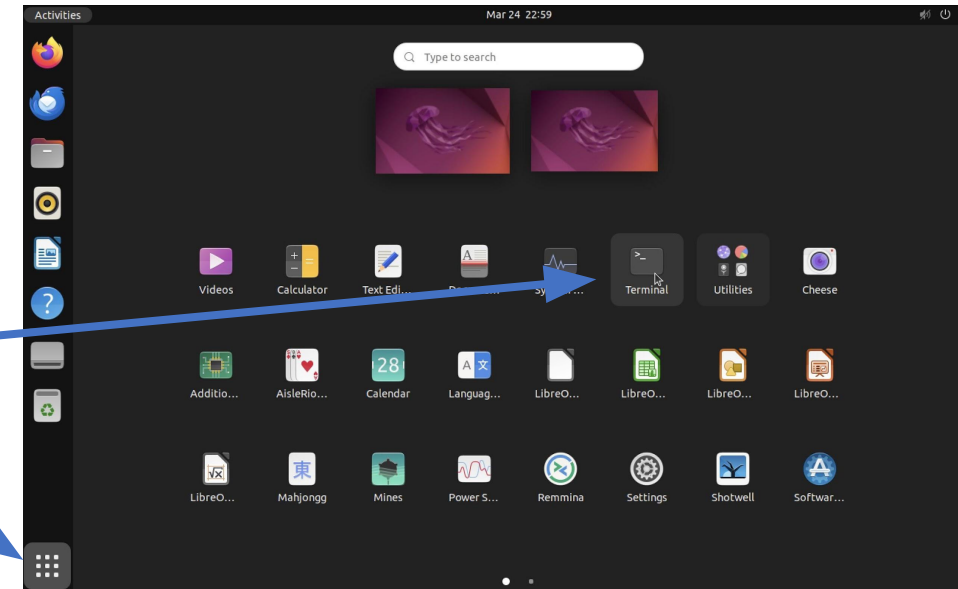
- No Windows:
 1. Pressione as teclas **Windows + r**
 2. Na caixa Executar, digite `cmd` e clique em **OK**
 3. Isso abre a janela do prompt de comando



Terminal ou prompt de comando

Como usar o terminal?

- No Linux (Ubuntu):
 1. Entre no menu de todos os programas
 2. Procure pelo “terminal”
 3. OU Pressione as teclas **control + alt + t**
 - Isso abre a janela do prompt de comando



Terminal ou prompt de comando

Comandos

- Programas **muito** usados:
 - Saber em qual diretório me encontro?
 - Linux: `pwd`
 - Windows: `echo %cd%`
 - Trocar de diretório? (Ir de uma pasta a outra)
 - Linux: `cd`
 - Windows: `cd`
 - Listar o conteúdo de uma pasta
 - Linux: `ls`
 - Windows: `dir`
 - Executar um programa?
 - Linux: `./programa`
 - Windows: `.\programa.exe`
- Dicas:
 - Interromper a execução de um programa:
 - `control + c`
 - Repetir o ultimo comando?
 - Usar setinha para arriba no teclado

Compilando um programa em C

- Quais são os passos para compilar um programa?
 1. Abrir um terminal
 2. Instalar o GCC
 3. Escrever um código fonte em C
 4. Compilar o código fonte para criar um programa com GCC
 5. Executar o programa

Escrevendo um programa em C

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

```
1.  void main() {  
2.      //corpo da função ("//" indica um comentário)  
3.      // não retorna nada pela tipo void  
4.  }  
5.  
6.  int main(int argc, char* argv[]) {  
7.      return 0; //retorna qualquer número inteiro  
8.  }
```

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!

```
1.  #include <stdio.h>
2.  void main() {
3.      printf("Alo mundo!");
4.  }
```

Escrevendo um programa em C

programa1.c

■ programa1.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 }
```

O que fazem
essas outras
linhas de
Código?

Escrevendo um programa em C

Imprimindo na tela

- As próximas linhas do programa `programa1.c` são:

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

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

```
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

Imprimindo na tela

- 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

Imprimindo na tela

- Essa sequência de caracteres pode conter algumas tags que representam valores, conhecidas como especificadores de formato

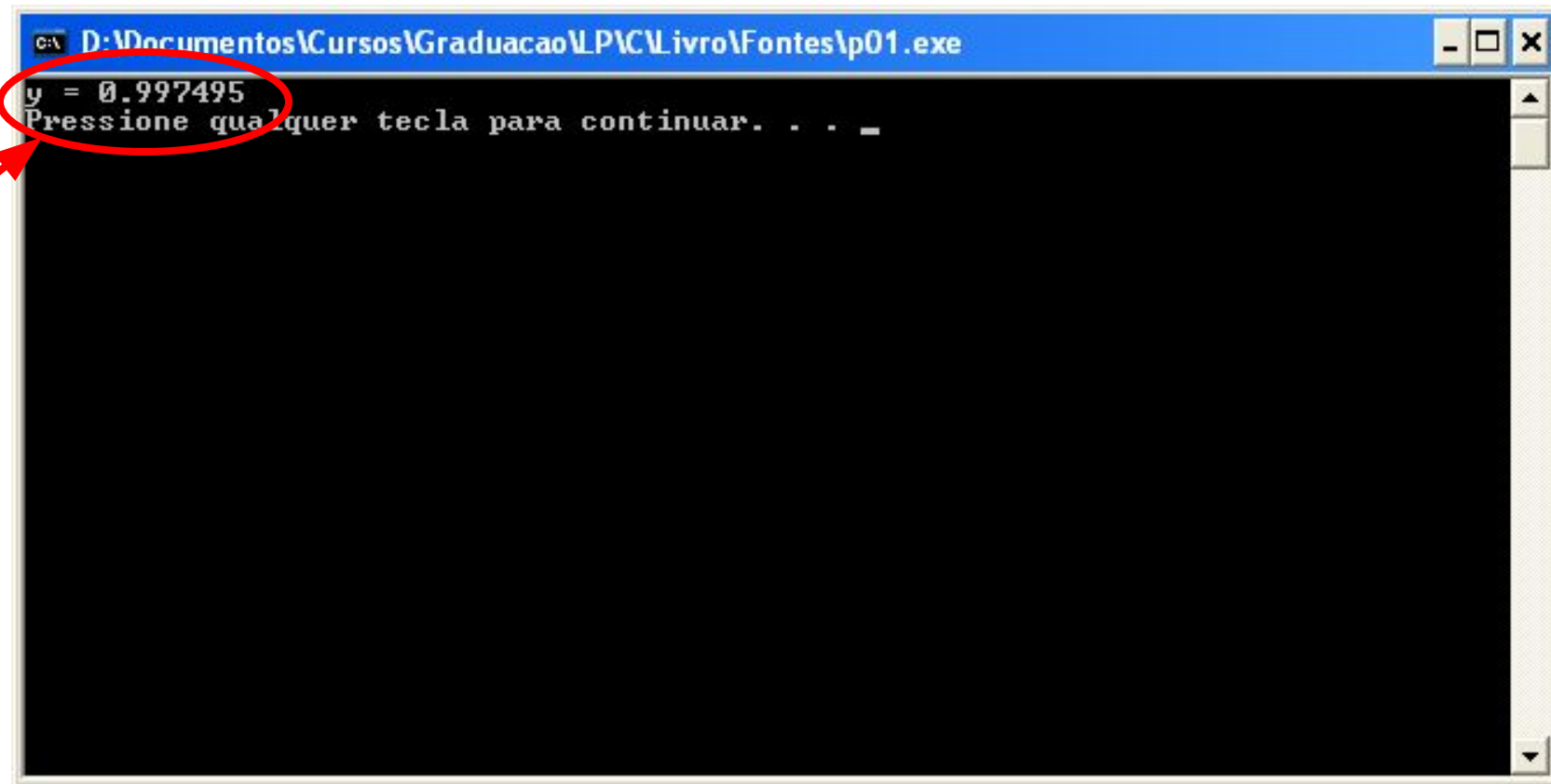
```
printf("y = %f", y);  
printf("\n");
```

- 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

Mostrando o resultado no terminal

- Veja:



```
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

Tags do printf

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

```
int a = 9;  
float b = 0.1;  
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
\a	Caractere (invisível) de aviso sonoro.
\n	Caractere (invisível) de nova linha.
\t	Caractere (invisível) de tabulação horizontal.
\'	Caractere de apóstrofo

Escrevendo um programa em C

- Observe a próxima linha do programa `programa1.c`:
- Ela exibe “o caractere (invisível) de nova linha”
 - Qual o efeito disso? Provoca uma mudança de linha! Próxima mensagem será na próxima linha

`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 }
```

Escrevendo um programa em C

Aplicando um “PAUSE”

- Agora a próxima linha:
- Ela exibe a mensagem:
 - “Pressione qualquer tecla para continuar...” e interrompe a execução 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 }
```

Escrevendo um programa em C

- A execução será retomada quando o usuário pressionar alguma tecla
- A última linha do programa `programa1.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


Retornando 0 no Main

- 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 `programa1.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;  
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 `programa1.c`:

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

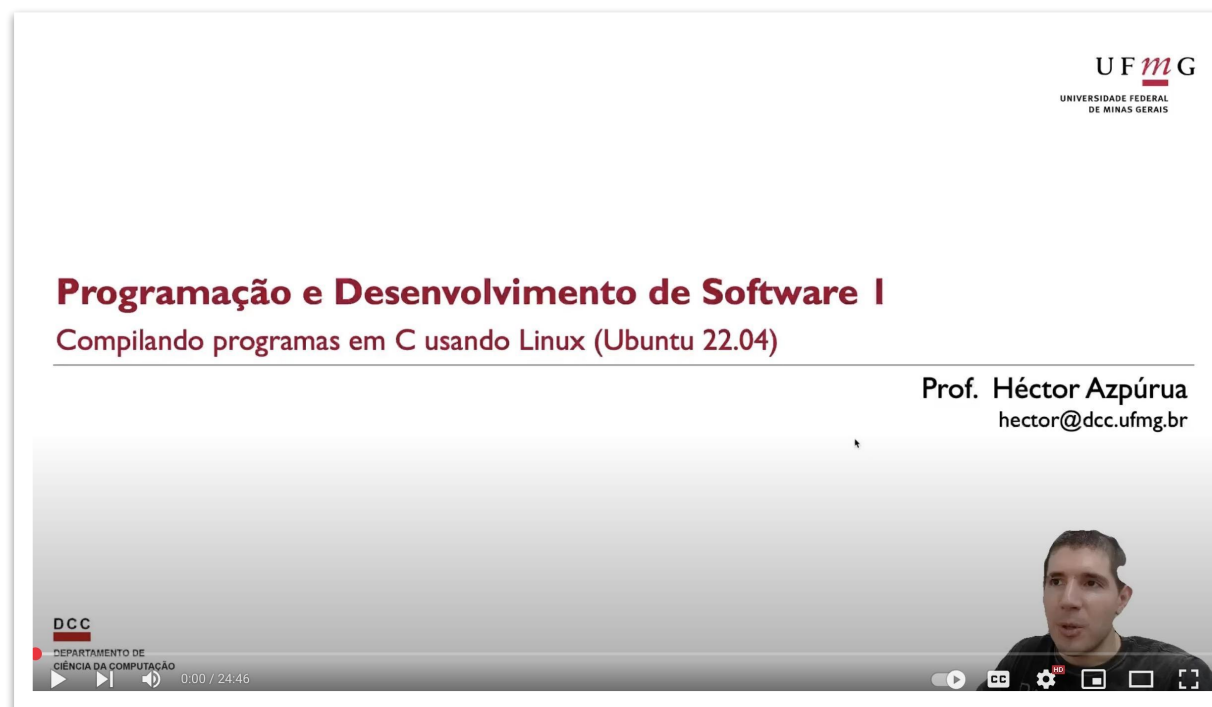
```
void main(void)
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
}
```

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?

Vídeo de passo a passo de compilação

- PDS I - Compilação de arquivos C usando gcc no terminal - Linux Ubuntu
 - <https://www.youtube.com/watch?v=ttQXYJXDZfw>



Perguntas?

- E-mail:
 - hector@dcc.ufmg.br
- Material da disciplina:
 - <https://pedroolmo.github.io/teaching/pdsI.html>
- Github:
 - <https://github.com/h3ct0r>



Héctor Azpúrua
h3ct0r