

Programação Procedimental

Introdução, Variáveis e Entrada/Saída

Aula 01

Prof. Felipe A. Louza



Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Primeiro programa em C

Um programa em C possui a seguinte estrutura:

```
1 #include <stdio.h>
2
3 //função principal
4 int main() {
5
6     printf("Hello, world!\n");
7
8     return 0;
9 }
```

- Saída:

```
1 Hello, world!
```

Primeiro programa em C

Algumas características:

- As linhas terminam com **ponto e vírgula** (quase sempre!).
- Em **C**, indentação não é obrigatória, mas é **boa prática** de programação

Primeiro programa em C

Função principal (`main`)

- Sempre devolve um valor inteiro (`int`)
- Se devolver `0` significa que não houve erros: `EXIT_SUCCESS`
 - Valores diferentes indicam o erro que ocorreu.

Comentários

- Os comentários de um programa são ignorados.
 - Uma linha `// Comentários`.
 - Uma ou várias linhas `/* Comentários */`

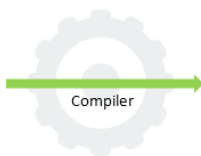
Executando o programa

Como executar um programa escrito em C?

- Precisamos gerar um arquivo executável \Leftarrow **Compilador**

```
#include  
<stdio.h>  
int main()  
{  
    printf("Hello")  
;  
    return 0;  
}
```

Source code



```
100010101010101  
000100101010111  
011111100110000  
001011001101010  
010111011100011  
011111001111000  
000110011110101  
010010010101000
```

Executable code

Executando o programa

Compilando (no terminal):

```
1 $ gcc hello.c -o hello
```

Executando o programa:

```
1 $ ./hello
```

Executando o programa

Mais opções do compilador (no terminal):

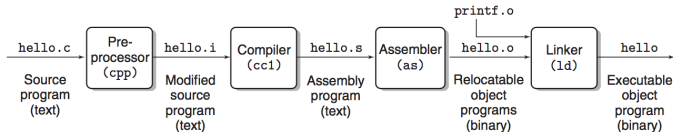
```
gcc -std=c99 -Wall -Werror -lm hello.c -o hello
```

Flags:

- std=c99: usa o padrão C99
- Wall: dá mais warnings de compilação
- Werror: warnings viram erros de compilação
- g: permite usar gdb e valgrind
- o: define o nome do programa

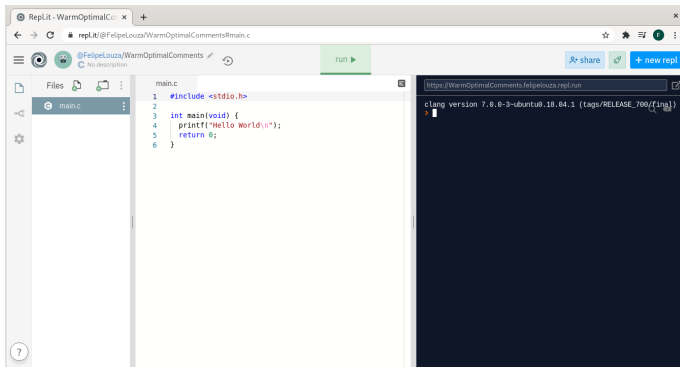
Compiladores

Na verdade o **compilador** realiza a sua tarefa juntamente com outros programas.



Compiladores

Alternativa: <https://repl.it/>



Erros

Erros de compilação?

- Caso o programa não esteja de acordo com as regras da linguagem, erros de compilação ocorrerão.

```
1 #include <stdio.h>
2
3 int main(){
4
5     print ("Hello, world!\n");
6
7     return 0;
8 }
```

- Saída:

```
1 /usr/bin/ld: /tmp/ccYsOLKG.o: in function `main':
2 teste.c:(.text+0xf): undefined reference to `print'
3 collect2: error: ld returned 1 exit status
```

Erros

Erro comum: esquecer o ponto e vírgula

```
1 #include <stdio.h>
2
3 int main(){
4
5     printf("Hello, world!\n")
6
7     return 0;
8 }
```

- Saída:

```
1 teste.c:5:18: error: expected ';' before 'return'
2     printf("Hello")
3
4                 ;
5         6
6     return 0;
7         ~~~~~
```

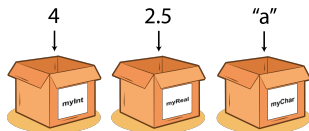
Roteiro

- 1 Primeiro programa em C
- 2 Variáveis**
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Variáveis

Variáveis são utilizadas para armazenar dados na memória do computador.

- Toda variável possui um nome e tipo.
- O tipo de uma variável determina o que pode ser armazenado nela.



Em C, toda variável precisa ser **declarada** antes de ser usada:

- Exemplo:

```
1 int i;
```

- declara uma variável de nome **i** do tipo **int**

Variáveis

Podemos declarar **mais de uma** variável em uma linha:

- Exemplo:

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     int i, j, k;
7
8     return 0;
9 }
```

Comando de Atribuição

O símbolo '=' é o **comando de atribuição** de valores para variáveis.

- A sintaxe é a seguinte:

variável = **valor**;

- Exemplo:

```
1  int a;  
2  float c;  
3  a = 5;  
4  c = 67.89505456;
```

Comando de Atribuição

Podemos **inicializar** uma variável durante a sua declaração:

- Exemplo:

```
1  int a = 5;  
2  float c = 67.89505456;
```

- **Importante:** variáveis não inicializadas começam com **lixo!**

Regras para nomes de variáveis em C

Algumas regras para nomes:

- Deve começar com uma letra (maíuscula ou minúscula) ou “_”.
- Nunca pode começar com um número.
- Pode conter letras maiúsculas, minúsculas, números e “_”.
- Não pode conter os seguintes símbolos:

{ (+ - * / \ ; . , ?

- Letras maiúsculas e minúsculas são diferentes.

Regras para nomes de variáveis em C

Nomes reservados:

- As seguintes palavras **já tem um significado** na linguagem C e por esse motivo **não podem** ser utilizadas como nome de variáveis:

| | | | | |
|----------|----------|---------|----------|--------|
| auto | double | int | struct | break |
| enum | register | typedef | char | extern |
| return | union | const | float | short |
| unsigned | continue | for | signed | void |
| default | goto | sizeof | volatile | do |
| if | static | while | | |

Tipos de variáveis

Variáveis podem ocupar **tamanhos diferentes** na memória¹, dependendo do seu tipo:

| Tipo | Bytes | Valores |
|--------|-------|--|
| short | 2 | -32.768 a 32.767 |
| int | 4 | -2.147.483.648 a 2.147.483.647 |
| long | 8 | 2^{63} a $2^{63} - 1$ |
| float | 4 | $1,2 \times 10^{-38}$ a $3,4 \times 10^{38}$ |
| double | 8 | $2,2 \times 10^{-308}$ a $1,8 \times 10^{308}$ |
| char | 1 | 0 a 255 |

¹pode depende do compilador...

Memória

Memória do computador:

- O espaço ocupado por uma variável depende do seu tipo.
- O endereço de uma variável é obtido pelo operador `&`.

| Address | Content | Name | Type |
|----------|---------|------|--------|
| 90000000 | 00 | i | int |
| 90000001 | 00 | | |
| 90000002 | 00 | | |
| 90000003 | FF | | |
| 90000004 | FF | s | short |
| 90000005 | FF | | |
| 90000006 | 1F | d | double |
| 90000007 | FF | | |
| 90000008 | FF | | |
| 90000009 | FF | | |
| 9000000A | FF | | |
| 9000000B | FF | | |
| 9000000C | FF | | |
| 9000000D | FF | | |
| 9000000E | 90 | p | int |
| 9000000F | 00 | | |
| 90000010 | 00 | | |
| 90000011 | 00 | | |

```
1 int main(){  
2  
3     int i,p;  
4     short s;  
5     double d;  
6 }
```


Modificadores para tipos inteiros

Modificador **unsigned**:

- Armazena somente **valores positivos**

| Tipo | Bytes | Valores |
|-----------------------|----------|-------------------|
| unsigned short | 2 | 0 a 65.536 |
| unsigned int | 4 | 0 a 4.294.967.295 |
| unsigned long | 8 | 0 a $2^{64} - 1$ |

Variáveis de tipo ponto flutuante

Variáveis de tipo ponto flutuante

- **float**:
 - Precisão de aproximadamente 6 casas decimais.
- **double**:
 - Maior precisão, mas é mais lento e gasta mais memória
 - Precisão de aproximadamente 15 casas decimais.

Variáveis de tipo caractere

O tipo `char` armazena letras (ou caracteres)

- Na verdade, armazena um valor inteiro em `[0, 255]`
- representa caracteres usando a tabela ASCII
 - cada número representa um caractere
- representamos usando aspas simples
 - `'a'` significa o número do caractere `a` na tabela ASCII

Tabela ASCII

| | | | | | | | | | |
|----|----------|----|---|----|---|-----|---|-----|---|
| 32 | (espaço) | 51 | 3 | 70 | F | 89 | Y | 108 | l |
| 33 | ! | 52 | 4 | 71 | G | 90 | Z | 109 | m |
| 34 | " | 53 | 5 | 72 | H | 91 | [| 110 | n |
| 35 | # | 54 | 6 | 73 | I | 92 | \ | 111 | o |
| 36 | \$ | 55 | 7 | 74 | J | 93 |] | 112 | p |
| 37 | % | 56 | 8 | 75 | K | 94 | ^ | 113 | q |
| 38 | & | 57 | 9 | 76 | L | 95 | _ | 114 | r |
| 39 | ' | 58 | : | 77 | M | 96 | ` | 115 | s |
| 40 | (| 59 | ; | 78 | N | 97 | a | 116 | t |
| 41 |) | 60 | i | 79 | O | 98 | b | 117 | u |
| 42 | * | 61 | = | 80 | P | 99 | c | 118 | v |
| 43 | + | 62 | ¿ | 81 | Q | 100 | d | 119 | w |
| 44 | , | 63 | ? | 82 | R | 101 | e | 120 | x |
| 45 | - | 64 | @ | 83 | S | 102 | f | 121 | y |
| 46 | . | 65 | A | 84 | T | 103 | g | 122 | z |
| 47 | / | 66 | B | 85 | U | 104 | h | 123 | { |
| 48 | 0 | 67 | C | 86 | V | 105 | i | 124 | — |
| 49 | 1 | 68 | D | 87 | W | 106 | j | 125 | } |
| 50 | 2 | 69 | E | 88 | X | 107 | k | 126 | ~ |

Caracteres barrados

- Caracteres com significado especial são barrados, p.ex.:

| | | | | | | |
|----|-----------------|----|--|----|----------------|----|
| \a | alerta sonoro | 7 | | \n | fim de linha | 10 |
| \b | backspace | 8 | | \\ | barra | 92 |
| \r | carriage return | 13 | | \' | aspas simples | 39 |
| \f | form feed | 12 | | \" | aspas dupla | 34 |
| \t | tabulação | 9 | | \0 | caractere nulo | 0 |

Conversão de tipos

Tipos podem ser convertidos (**casting**):

- valor **int** pode ser convertido para **double**
 - por exemplo, escreva **(double) x**
 - **1** é convertido para **1.0**
- valor **double** pode ser convertido para **int**
 - **1.0** é convertido para **1**
 - **1.937** é convertido para **1**

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes**
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Constantes

Podemos definir variáveis com valores constantes:

- Modificador: **const**

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     const int a = 10;
7     const int b;
8
9     return 0;
10 }
```

- Valor padrão (não inicializado) é zero.

Constantes

Valores constantes podem ser inicializadas apenas uma vez.

```
1  #include <stdio.h>
2
3  //função principal
4  int main(){
5
6      const int a;
7      const int b = 12;
8
9      b = 1;
10
11     return 0;
12 }
```

Constantes

Erro de compilação:

```
1 $ gcc teste.c
2 teste.c: In function 'main':
3 teste.c:9:5: error: assignment of read-only variable 'b'
4     9     b = 1;
5         ^
```

Constantes

Outra alternativa:

- Comando: **#define**
- Valores previamente determinados.

```
1  #include <stdio.h>
2
3  #define PI 3.1416
4
5  //função principal
6  int main(){
7
8      double x = PI;
9
10     return 0;
11 }
```

Constantes

Valores definidos também **possuem um tipo**.

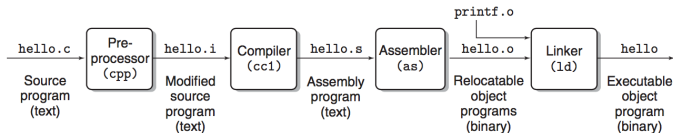
- Exemplos:

```
1 #define a 85  
2 #define b 0.111  
3 #define c 'c'  
4 #define d "Hello!"
```

- Adicionalmente temos um tipo **string** para constantes que corresponde a uma **sequência de caracteres**.

Constantes

Valores definidos são **substituídos** pelo pré-processador durante o **processo de compilação**.



Constantes

Exemplo:

```
1  #include <stdio.h>
2
3  #define msg "Hello World\n"
4
5  //função principal
6  int main(){
7
8      printf(msg);
9
10     return 0;
11 }
```

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos**
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Operadores aritméticos

Operadores aritméticos em C:

| Operação | Operador aritmético | Exemplo | Exemplo em C |
|---------------|---------------------|--------------|------------------|
| Adição | + | $a + 2$ | <code>a+2</code> |
| Subtração | - | $b - c$ | <code>b-c</code> |
| Multiplicação | * | $a \times 5$ | <code>a*5</code> |
| Divisão | / | $d \div e$ | <code>d/e</code> |
| Módulo | % | $5 \bmod 2$ | <code>5%2</code> |

- Módulo é o resto da divisão inteira.

Expressões e aritméticas

Podemos atribuir **expressões aritméticas** com valores em **variáveis**, **constantes** ou resultados de outras expressões:

- A sintaxe é a seguinte:

variável = expressão;

- Exemplo:

```
1  #define pi 3.1416
2
3  int raio = 5+2;
4  double area = 2*pi*(raio*raio);
```

Precedência de operadores

Regras de precedência de operadores:

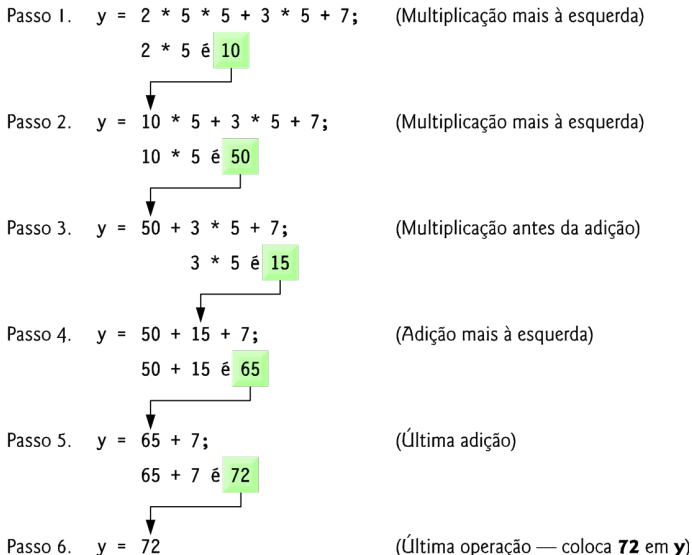
- As mesmas da Álgebra.

| Operação | Operador | Ordem de avaliação |
|---------------|----------|--|
| Parênteses | () | Avaliados primeiro (pares mais internos avaliados antes) |
| Multiplicação | * | Avaliados em segundo lugar. |
| Divisão | / | Se houver vários, avaliação da esquerda para direita. |
| Módulo | % | |
| Adição | + | Avaliados por último. |
| Subtração | - | Se houver vários, avaliação da esquerda para direita. |

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores**
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

Precedência de operadores



Mais sobre o operador / em C

- O resultado depende do tipo dos operandos:

| | Python | C |
|-----------|--------|-----|
| 6 / 4 | 1.5 | 1 |
| 6.0 / 4.0 | 1.5 | 1.5 |
| 6.0 / 4 | 1.5 | 1.5 |
| 6 / 4.0 | 1.5 | 1.5 |

Sobre o operador % em C

- Aceita apenas valores inteiros:

| | Python | C |
|-----------|--------|--------------------|
| 6 % 4 | 2 | 2 |
| 6.0 % 4.0 | 2.0 | erro de compilação |
| 6.0 % 4 | 2.0 | erro de compilação |
| 6 % 4.0 | 2.0 | erro de compilação |

Se necessário, podemos fazer [casting](#):

```
1 double x = 6.0;  
2  
3 int y = (int)x % 4;
```

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --**
- 7 Entrada/Saída de dados
- 8 Referências

Incremento(++) e Decremento(--)

É muito comum escrevermos expressões para incrementar/decrementar o valor de uma variável por 1.

```
1 a = a + 1;
```

- Podemos abreviar essa expressão:

```
1 a++;
```

- ou:

```
1 ++a;
```

- Qual é a diferença?

Incremento(++) e Decremento(--)

A diferença está na **ordem da operação**.

- **a++**: o valor de **a** é primeiro usado na expressão, e depois é incrementado.

```
1 int a = 1;  
2 x = (a++) * 2;
```

– **x** é igual a 2, e **a** é 2;

- **++a**: o valor de **a** é incrementado, e depois usado na expressão.

```
1 a++;  
2 x = (++a) * 2;
```

– **x** é igual a 4, e **a** é 2;

O tipo int

Outras abreviações

| | |
|--------|-----------|
| a++ | a = a + 1 |
| ++a | a = a + 1 |
| a-- | a = a - 1 |
| --a | a = a - 1 |
| a += b | a = a + b |
| a -= b | a = a - b |
| a *= b | a = a * b |
| a /= b | a = a / b |
| a %= b | a = a % b |

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados**
- 8 Referências

Escrevendo na tela

A função `printf()` é utilizada para imprimir na tela:

- Faz parte da biblioteca `<stdio.h>`.

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     printf("Hello, world!\n");
7
8     return 0;
9 }
```

- Importante:
 - Imprime uma sequência de caracteres (`texto`).

Escrevendo na tela

Como imprimir valores inteiros?

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     int x = 10;
7
8     printf(x);
9
10    return 0};
11 }
```

- Erro comum:
 - A função `printf()` deve receber um texto (entre aspas duplas).

Escrevendo na tela

Podemos **modificar/formatar** o texto passado para a função **printf()**:

- **%d** significa substituir por um inteiro

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     int x = 10;
7
8     printf("%d", x);
9
10    return 0};
11 }
```

- Existem outras substituições: **%f**, **%s**, etc...

Escrevendo na tela

Outros **formatadores**:

| Formato | Tipo |
|--------------------|--|
| <code>"%d"</code> | <code>int</code> (número inteiro) |
| <code>"%ld"</code> | <code>long long</code> (número inteiro) |
| <code>"%f"</code> | <code>float</code> (ponto flutuante) |
| <code>"%lf"</code> | <code>double</code> (ponto flutuante) |
| <code>"%c"</code> | <code>char</code> (caractere) |
| <code>"%s"</code> | <code>string</code> (cadeia de caracteres) |

Escrevendo na tela

A substituição é feita da **esquerda para a direita** no texto:

- Exemplo:

```
1  #include <stdio.h>
2
3  //função principal
4  int main(){
5
6      int x = 10;
7      double y = 2.2;
8
9      printf("%d %s %f = %d", x, "+", y, x+y);
10
11     return 0};
12 }
```

- Importante:

- Não adiciona a quebra de linha '**\n**' automaticamente.

Escrevendo na tela

Podemos especificar **alguns detalhes** para os formatadores, como:

- **número mínimo** de caracteres ocupados: **%4d**

```
1 int x = 10;  
2 printf("%4d", x);
```

- Saída:

```
1  _10
```

Escrevendo na tela

Podemos especificar **alguns detalhes** para os formatadores, como:

- número de casas decimais: **%.3f**

```
1 float y = 3.55556;  
2 printf("%.3f", y);
```

- Saída:

```
1 3.556
```

Escrevendo na tela

Podemos especificar **alguns detalhes** para os formatadores, como:

- **número mínimo** de caracteres e **número de casas decimais**: **%5.3f**

```
1 float y = 3.55556;  
2 printf("%6.3f", y);
```

- Saída:

```
1  3.556
```

- O ponto **'.'** conta como caractere.

Escrevendo na tela

É importante especificar corretamente o formatador:

- Exemplo:

```
1 unsigned int z = 4000000000;  
2 printf("%d", z);
```

- Saída:

```
1 -294967296
```

- O correto seria:

```
1 printf("%u", z);
```

Escrevendo na tela

Outras opções:

- Notação científica:

```
1 printf("%e", 10.02545);
```

- Saída:

```
1 1.002545e+01
```

- Valor ASCII:

```
1 printf("%d", 'A');
```

- Saída:

```
1 65
```

Escrevendo na tela

Caracteres especiais:

| Símbolo | Efeito |
|-----------------|----------------------|
| <code>\n</code> | quebra de linha |
| <code>\t</code> | tabulação horizontal |
| <code>\"</code> | aspas duplas |
| <code>\'</code> | aspas simples |
| <code>\\</code> | barra invertida |
| <code>\a</code> | beep |

Leitura da entrada

A função `scanf()` é utilizada para ler da entrada padrão (teclado):

- Faz parte da biblioteca `<stdio.h>`.

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     int x, y;
7     scanf ("%d %d", &x, &y);
8     printf("Os valores digitados foram: %d %d\n", x, y);
9
10    return 0;
11 }
```

- Recebe uma string com `formatadores` que diz `quantos` valores serão lidos e quais os seus `tipos`.

Formatos de leitura de variável

- Os **formatadores de leitura** são semelhantes aos da função `printf()`.

| Formato | Tipo |
|--------------------|---------------------------------|
| <code>"%d"</code> | Lê um número decimal |
| <code>"%ld"</code> | Lê um inteiro longo |
| <code>"%f"</code> | Lê um número em ponto flutuante |
| <code>"%lf"</code> | Lê um double |
| <code>"%c"</code> | Lê um caractere |
| <code>"%s"</code> | Lê uma sequência de caracteres |

Leitura da entrada


A função `scanf()` altera o conteúdo das variáveis

- Precisamos passar o endereço da variável usando operador `&`.

```
1 int a;  
2 double d;  
3 char c;  
4 scanf("%d %lf %c", &a, &d, &c);
```

- A substituição é feita da esquerda para a direita.

A função scanf

A leitura do teclado **aguarda** que o usuário **digite um valor e** ().
O valor digitado é atribuído à variável.

```
1 #include <stdio.h>
2
3 //função principal
4 int main(){
5
6     int x, y;
7     scanf ("%d %d", &x, &y);
8     printf("Os valores digitados foram: %d %d\n", x, y);
9
10    return 0;
11 }
```

- **ignora** espaços em branco, tabs e quebras de linha (**nem sempre!**)

A função scanf

O programa abaixo é composto de quatro passos:

- 1 Cria uma variável `n`;
- 2 Escreve na tela `Digite um número:`
- 3 Lê o valor do número digitado \leftarrow `10`
- 4 Imprime na tela `0 valor digitado foi 10`

```
1  #include <stdio.h>
2
3  int main(){
4
5      int n;
6      printf("Digite um número: ");
7      scanf("%d", &n);
8      printf("0 valor digitado foi %d\n",n);
9
10     return 0;
11 }
```

A função scanf

Erros comuns:

```
1  #include <stdio.h>
2
3  int main(){
4
5      int a;
6      scanf(a);
7
8      int b;
9      scanf("%d", b);
10
11     return 0;
12 }
```

A função scanf

```
1  #include <stdio.h>
2
3  #define PI 3.1416
4
5  int main(){
6
7      float raio;
8
9      printf("Digite o valor do raio do círculo: ");
10     scanf("%f", &raio);
11
12     //calculando a área
13     float area = PI * (raio*raio);
14     printf("Área do círculo: %d\n", area);
15
16     return 0;
17 }
```

Dúvidas?

Roteiro

- 1 Primeiro programa em C
- 2 Variáveis
- 3 Constantes
- 4 Operadores Aritméticos
- 5 Precedência de operadores
- 6 Operadores ++ e --
- 7 Entrada/Saída de dados
- 8 Referências

- ① Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.