



# **PROGRAMAÇÃO EM C/C++ - FUNDAMENTOS**

# CONTEÚDOS

- Estrutura de um programa em C/C++
- Dados em C
- Estudo e emprego da função printf()
- Cadeia de caracteres e entrada e saída de dados formatados
- Operadores e expressões em C/C++
- Estruturas repetitivas em C/C++
- Estruturas repetitivas complexas
- Estruturas alternativas em C/C++
- Funções
- Matrizes
- Cadeias de caracteres (string)



# Sessão 1



## C ou C++?

- Para aprender C++ preciso de saber C?
- Qual a diferença entre C e C++?
- Qual a melhor?
- Onde se usa C e onde se usa C++?



# PARA APRENDER C++ PRECISO DE SABER C?

- A linguagem C++ é uma extensão do C, sendo cada uma delas independente da outra.

Logo a resposta é, não!



## QUAL A DIFERENÇA ENTRE C E C++?

- O C++ podemos dizer que é o C com mais recursos e funções. E a principal diferença é que C++ é multi-paradigma.

Um paradigma pode ser entendido como um **tipo de estruturação ao qual a linguagem deverá respeitar**. Dependendo do objetivo proposto, a solução que a linguagem oferecerá obedece a um tipo de paradigma.

- E a principal diferença é que C++ é multi-paradigma: podemos programar em C++ com o paradigma Estruturado (o C é somente estruturado) ou com o paradigma de Orientação a Objetos (como a linguagem Java, por exemplo).



## QUAL A MELHOR?

- Não existe linguagem melhor que a outra, existe linguagem mais adequada para determinado tipo de problema.

Sim, o C++ tem mais recursos, mas isso tem também algumas contrapartidas.

A sua implementação é mais complexa, tem mais coisas ocorrendo em segundo plano. Logo, C++ não é tão eficiente quanto C, consome mais memória e processamento.



## ONDE SE USA C E ONDE SE USA C++?

Como dissemos, C é mais eficiente, é mais rápido. Programar em C é programar próximo ao 'metal', é ter acesso direto a memória do computador. Isso se chama programação em baixo nível.

O C++ também consegue isso, mas C++ também oferece a oportunidade de trabalhar em alto nível.





## ONDE SE USA C E ONDE SE USA C++?

Porém, algumas aplicações exigem o máximo de velocidade e eficiência possível, por isso o C é usado para a criação de Sistemas Operacionais (como Linux e Windows) e até mesmo para trabalhar com microcontroladores.

Já C++ não serve para esses propósitos, porém serve para outros.



## ONDE SE USA C E ONDE SE USA C++?

O problema do C ocorre quando as aplicações começam a ficar muito grandes, o que dificulta o controle e manutenção do código. É aí que entra o C++.

Devido à sua Orientação a Objetos, e ao fato de possuir mais recursos e opções (como Templates), facilita muito na hora de programar e aumenta incrivelmente a eficiência do programador C++.

Ou seja, em um mesmo intervalo de tempo, o programador C++ consegue ser mais eficiente que um programador C fazendo a mesma coisa, pois o C++ tem mais opções à disposição, prontas para o uso do programador.



## ONDE SE USA C E ONDE SE USA C++?

Um exemplo de uso da linguagem C++ são os jogos de alto rendimento.

Eles são muito complexos para se fazer, por isso não é recomendado o uso de C ou Assembly.

Porém, eles precisam de uma eficiência muito grande, por isso não pode ser feito numa linguagem de mais alto nível, como Java ou C#.

É aí que entra o C++: ele é, ao mesmo tempo, muito mais eficiente que a maioria das linguagens e é mais pronto para o uso do que as linguagens de baixo nível, como o C.



# COMPILADOR

- O compilador é uma ferramenta/programa que pega no código fonte e converte-o num programa executável.
- No decorrer da formação vamos utilizar o compilador gcc. O **gcc** da GNU é um dos compiladores de C mais avançados e mais versáteis que existe no mercado e, ainda por cima, é software de domínio livre.



# COMPILADOR

- O primeiro passo de um compilador é analisar o código presente no arquivo-fonte, verificando se existem erros de sintaxe. Caso algum erro de sintaxe seja encontrado, a compilação é interrompida para que o programador possa corrigir estes erros. Caso o código não possua erros o próximo passo do compilador é criar um arquivo de código-objeto, que possui as instruções do programa já traduzidas para a linguagem máquina e informações sobre alocação de memória, símbolos do programa (variáveis e funções) e informações de debug. A partir deste arquivo de código-objeto, o compilador finalmente cria um arquivo executável com o programa compilado, que funciona independente do compilador e realiza as instruções criadas pelo programador.



# ESTRUTURA DE UM PROGRAMA EM C/C++

- Um programa escrito na linguagem C tem a seguinte estrutura:

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

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

```
int main()
```

```
{
```

```
    printf("Hello world!\n");
```

```
    return 0;
```

```
}
```



# ESTRUTURA DE UM PROGRAMA EM C/C++

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

```
int main()
{
```

```
    printf("Hello world!\n");
```

```
    return 0;
}
```

## Bibliotecas

**Função principal (main)** - Todos os comandos executados pelo programa estão contidos entre as chavetas “{ }” da função main().

## Comandos

## Retorno da função main()



# ESTRUTURA DE UM PROGRAMA EM C/C++

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

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

```
int main()
```

```
{
```

```
    printf("Hello world!\n");
```

```
    return 0;
```

```
}
```

## Instrução:

cada instrução termina  
com ponto-e-vírgula (;)





# ESTRUTURA DE UM PROGRAMA EM C/C++

```
int main()
```

```
{
```

```
// comentário de uma só linha
```

```
printf("Hello world!\n");
```

```
/* comentário  
de  
várias linhas */  
return 0;  
}
```

- O comentário começa com `//` e termina no final da linha.

- Um *comentário* começa com `/*` e termina com `*/`



## BIBLIOTECA

- Todos os programas em linguagem C usam funções das bibliotecas padrão da linguagem.
- O conjunto de funções de cada biblioteca é descrito em um arquivo-interface (= *header-file*), que tem o mesmo nome da biblioteca e sufixo .h.



# BIBLIOTECA

- Vejamos algumas interfaces das principais bibliotecas padrão C:

*stdlib.h*

*stdio.h*

*math.h*

*string.h*

*limits.h*

*ctype.h*

*time.h*

*stdbool.h*



## BIBLIOTECA

○ Para ter acesso a uma biblioteca (às funções nela contidas), o nosso programa deve incluir uma cópia do correspondente arquivo-interface. Por exemplo, basta escrever:

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

para que o pré-processador da linguagem C acrescente uma cópia de `stdio.h` ao seu programa.

Esta biblioteca (`stdio.h`) *contém várias funções de entrada e saída.*



# RETORNO DA FUNÇÃO MAIN

- O comando “return 0” é a “resposta” da função main para o sistema. Quase todas as funções retornam um valor para o sistema ou programa que a chamou, por exemplo, uma função pode retornar o resultado de uma operação matemática executada por ela. No caso da função main(), ela retorna um valor para o sistema operativo que executou o programa. Esse valor é interpretado pelo sistema como uma mensagem indicando se o programa foi executado corretamente ou não. Um valor de retorno ‘0’ indica que o programa foi executado sem problemas; qualquer outro valor de retorno indica problemas. Quando o programa é executado até ao fim, ele retorna ‘0’ ao sistema operativo, indicando que foi executado e terminado corretamente. Quando o programa encontra algum erro ou é terminado antes da hora, ele retorna um valor qualquer ao sistema, indicando erro durante a execução.



# Sessão 2



# DADOS EM C

Como vamos armazenar dados ?

- Variáveis e atribuições

Os programas em C efectuem computação *modificando valores* em memória

Os locais para guardar esses valores são designados usando *variáveis*



# DADOS EM C

O que é uma **variável**?

- Variável é um local reservado na memória para armazenar um tipo de dado.
- Toda variável deve ter um identificador, ou seja um nome.
- Além de ter um nome, a variável também precisa ter um tipo.
- O tipo de dado de uma variável determina o que ela é capaz de armazenar.





# DADOS EM C

## Tipos de dados

- Existem diversos tipos de dados que podem ser usados nas variáveis.
- Cada tipo de dado é específico para armazenar uma determinado formato de dado.
- Podemos dizer que os principais tipos de dados numéricos dividem-se em:
  - Inteiro: armazena números inteiros
  - Real: armazena números com casas decimais, são as variáveis de ponto flutuante;
- O tipo de dado caracter é capaz de representar um caracter ou um conjunto de caracteres (letras, dígitos ou símbolos) .



# DADOS EM C

## **Tipos de dados** mais comuns em linguagem C

- **int**: armazena valores numéricos inteiros.
- **char**: armazena caracteres.
- **float**: armazena números com ponto flutuante (reais) com precisão simples.
- **double**: armazena números com ponto flutuante, com precisão dupla, ou seja normalmente possui o dobro da capacidade de uma variável do tipo float.



# DADOS EM C

## Sintaxe de declarações de variáveis em C

- Em C devemos listar primeiro o tipo, depois o nome da variável.
- **Sintaxe:**

<tipo> <nome\_da\_variável>;

Exemplo de declaração de variável do tipo inteiro

**int contador;**

onde:

- int é o tipo da variável (inteiro)
- contador é o nome da variável.



# DADOS EM C

- **Declarando uma variável real (ponto flutuante):**

```
float salario;
```

- **Declarando uma variável do tipo character:**

```
char letra;
```



## DADOS

- Exemplo:

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

```
int main(void)
```

```
{
```

```
    // declaração de variáveis
```

```
    int a, b, c;
```

```
    scanf("%d", &a);
```

```
    scanf("%d", &b);
```

```
    c = a + b;
```

```
    printf("%d\n", c);
```

```
    return 0;
```

```
}
```



# DADOS EM C

- Endereço de uma variável

Toda variável possui um endereço de memória associado a ela. Esse endereço refere-se ao local da memória onde os dados ficam armazenados.

Quando um programa está sendo escrito, não conhecemos o endereço onde a variável será armazenada. Sendo assim, para fazer referência à variável usamos o nome da mesma e o computador se encarrega do resto.



# DADOS EM C

## Regras para criação de nomes de variáveis

Devemos observar algumas regras para criar os identificadores das variáveis.

- O nome de uma variável deve sempre iniciar com uma letra ou com `_`, jamais deve iniciar com um número.

Exemplo:

- `float total2; // está correto`
  - `float 2total; //está errado, pois não iniciou com uma letra.`
- Como visto, o nome de uma variável pode conter dígitos em qualquer posição, menos na primeira.



# DADOS EM C

## Regras para criação de nomes de variáveis

- Lembre-se que a linguagem C é “case sensitive”, isto é as letras maiúsculas e minúsculas fazem diferença, pois são tratadas como caracteres diferentes.

Isso significa que uma variável declarada como Total será diferente de total.





# DADOS EM C

## Comando de atribuição

- Atribuir significa armazenar um valor em uma variável.
- Em linguagem C usamos o operador = para fazer uma atribuição.

Exemplo:

```
X = 10;
```

Esta instrução atribui o valor 10 para a variável X.

Para poder atribuir um valor a uma variável, esta tem que ter sido previamente declarada.



# DADOS EM C

**Constantes** são usadas para armazenar valores que NÃO podem ser modificadas durante a execução de um programa.

- Uma constante precisa ser declarada, e para tanto usamos a diretiva de pré-processador `#define`.

Sintaxe:

**`#define <nome_da_constante> <valor>`**

- A declaração da constante deve ser feita no início do programa.
- É extremamente recomendável utilizar letras maiúsculas ao declarar uma constante.
- Declarando as constantes como maiúsculas podemos facilmente diferenciá-las das variáveis que por convenção devem ser declaradas em letras minúsculas.

Exemplo:        **`#define IVA 6`**



# DADOS EM C

## Constantes

Outra forma de utilizar constantes em C é usar o comando **const**. Ao usar const será declarada uma constante de um determinado tipo de dado que ocupa um espaço na memória RAM cujo valor não pode ser alterado em tempo de execução. Ao declarar o tipo devemos em seguida atribuir o valor para a constante. Por exemplo:

```
const float frete = 10.50;
```



# EXERCÍCIO

- Quais dos seguintes nomes podemos atribuir a variáveis?
  - soma
  - valor1
  - idade do aluno
  - data\_nascimento
  - 3valor
  - media,
- Como poderíamos alterar os outros para que pudessem ser nomes de variáveis?



## EXERCÍCIO - RESOLUÇÃO

- Quais dos seguintes nomes podemos atribuir a variáveis?
  - soma ✓
  - valor1 ✓
  - idade do aluno
  - data\_nascimento ✓
  - 3valor
  - media,
- Como poderíamos alterar os outros para que pudessem ser nomes de variáveis?
  - idade ou idade\_aluno ou idadeAluno ✓
  - valor3 ou terceiroValor ✓
  - media ✓



## EXERCÍCIO

- Qual a instrução adequada para efetuar a atribuição de um valor à variável idade?
  - a) idade:= 10
  - b) idade -> 13;
  - c) idade = idade (22)
  - d) idade= 24;
  - e) idade – 18;



## EXERCÍCIO - RESOLUÇÃO

- Qual a instrução adequada para efetuar a atribuição de um valor à variável idade?
  - idade:= 10
  - idade -> 13;
  - idade = idade (22)
  - **idade= 24;** ✓
  - idade – 18;



# EXERCÍCIO

- Quais as declarações de variáveis que são adequadas?
  - `int a;`
  - `float numero_irmãos;`
  - `char apelido;`
  - `int media_avaliacao;`
  - `int nome;`





## EXERCÍCIO - RESOLUÇÃO

- Quais as declarações de variáveis que são adequadas?
  - `int a;` ✓
  - `float numero_irmãos;`
  - `char apelido;` ✓
  - `int media_avaliacao;`
  - `int nome;`



# EXERCÍCIO

- Declare as seguintes variáveis em um programa:
  - letra = C ;
  - numero1 = 13 ;
  - numero2 = 3.56;
  - pi=3,1415;
  - mol =  $6 \times 10^{23}$ ;



## EXERCÍCIO - RESOLUÇÃO

○ Declare as seguintes variáveis em um programa:

- letra = 'c' ;                      **char** letra;
- numero1 = 13 ;                    **int** numero1;
- numero2 = 3.56;                  **float** numero2;
- pi=3,1415;                        **float** pi;
- mol = 6e23; =  $6 \times 10^{23}$ ;              **double** mol;



# FUNÇÃO PRINTF

- Em C, para apresentar mensagens no monitor pode utilizar-se a função **printf**. Esta função está definida na biblioteca **stdio.h** e permite diversos argumentos.

- No exemplo apresentado, a instrução

```
printf("Olá Mundo!");
```

chama a função printf com o argumento "Olá Mundo!" que é a string que se pretende mostrar no monitor.

De notar que a string pode conter sequências de escape, por exemplo:

```
printf("Olá \nMundo!");
```

mostrará no monitor:

Olá  
Mundo!



continua...

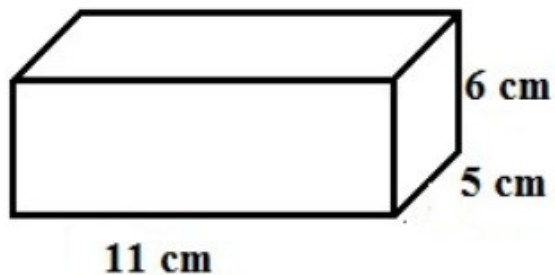
## CODEBLOCK

- Rumo à programação....
- File -> New Project
- Console application
- Language: C
- Definir nome do projeto



## EXERCÍCIO

Crie um programa para calcular o *volume*  $V$  de uma caixa retangular.



$$V = 11\text{cm} \times 5\text{cm} \times 6\text{cm} = 330\text{cm}^3$$



# EXERCÍCIO - RESOLUÇÃO

Crie um programa para calcular o *volume*  $VV$  de uma caixa retangular.

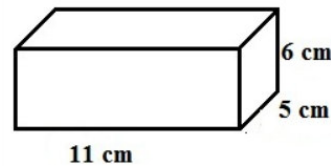
```
#include <stdio.h>
```

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

```
    int a,l,p,v;
    a=6;
    l=11;
    p=5;
    v=l*p*a;
```

```
    printf("Volume: %d",v);
```

```
    return 0;
}
```



$$V = 11\text{cm} \times 5\text{cm} \times 6\text{cm} = 330\text{cm}^3$$



# FUNÇÃO PRINTF – IMPRIMIR VALORES

Podemos usar a função de biblioteca printf para imprimir valores das variáveis. Exemplo:

```
int alt;  
alt = 6;  
printf("Altura: %d cm\n", alt);
```

*Imprime o texto:*

Altura: 6 cm

**%d** é um campo que é substituído pelo *valor duma variável*.





# FUNÇÃO PRINTF – IMPRIMIR VALORES

Para valores float usamos o campo **%f**.

```
float custo;  
custo = 123.45;  
printf("Custo: EUR %f\n", custo);
```

Resultado:

Custo: EUR 123.449997

- Não é possível representar 123.45 de forma exata como float!
- **%f** apresenta o resultado arredondado a 6 casas decimais



# FUNÇÃO PRINTF – IMPRIMIR VALORES

Para forçar formatação com nn casas decimais usamos %.n.nf:

Exemplo:

```
printf("Custo: EUR %.2f\n", custo);
```

Resultado:

Custo: EUR 123.45



# FUNÇÃO PRINTF – IMPRIMIR VALORES

Podemos formatar vários valores num só printf:

```
printf("Altura: %d cm; Custo: EUR %.2f\n", alt, custo);
```

Atenção:

- especificar o mesmo número de *campos* do que *argumentos*
- usar campos corretos para cada *tipo* (%d para int, %f para float)



# SCANF – ENTRADA DE DADOS

Tem por função efetuar a leitura de dados de uma fonte externa.

**A função scanf()** é utilizada para fazer a leitura de dados formatados via teclado.

- **Sintaxe:**

**scanf(“expressão de controle”, lista de argumentos);**

**Exemplo:**

```
scanf(“%f”, &salario);
```



# SCANF – ENTRADA DE DADOS

## Explicação:

**scanf("%f", &salario);**

este comando efetua uma leitura do teclado onde é esperada uma variável float (indicada por "%f"). O valor lido será armazenado no endereço da variável salário.

Na lista de argumentos devemos indicar os endereços das variáveis.

Para fazer isso adicionamos o símbolo "&" como prefixo na frente do nome da variável.

Linguagem C	Formato	Tipo de dados
char	%c	caracter
int	%d	inteiro
float	%f	real
double	%lf	real precisão dupla
Char[ ]	%s	cadeia de caracteres (string)



# EXERCÍCIO - RESOLUÇÃO

Crie um programa para calcular o *volume* VV de uma caixa retangular.

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

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

```
    int a,l,p,v;
```

```
    printf("Largura: ");
```

```
    scanf("%d", &l);
```

```
    printf("Profundidade: ");
```

```
    scanf("%d", &p);
```

```
    printf("Altura: ");
```

```
    scanf("%d", &a);
```

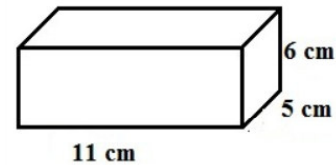
```
    v=l*p*a;
```

```
    printf("Volume: %d",v);
```

```
    system ("pause");
```

```
    return 0;
```

```
}
```



$$V = 11\text{cm} \times 5\text{cm} \times 6\text{cm} = 330\text{cm}^3$$



## EXERCÍCIO

Elabore um programa que sabendo o nome e o apelido do utilizador, devolva o nome completo.

- nome = 'Ana'
- Apelido = 'Santos'

Imprima:

Ana Santos



## EXERCÍCIO - SOLUÇÃO

Ana Santos

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

```
int main()  
{
```

```
    char nome,apelido ;  
    nome='Ana';  
    apelido='Santos';  
    printf("Nome: %c %c",nome,apelido);  
    return 0;  
}
```



Pois...

Nome: a s

Não era o esperado...  
Aparecem as últimas letra  
apenas.. Precisamos de  
uma solução... **STRING**





# STRING?

- O termo string serve para identificar uma sequência de caracteres.
- Na prática, as strings são usadas para representar textos.
- Em linguagem C, ao contrário de outras linguagens, não existe um tipo de dados string nativo.
- Para representar uma **string** em C, devemos criar um **vetor de caracteres**, ou seja um vetor do tipo char.

Vejamos como declarar string em C.

```
char nome_cliente[61];
```



# STRING

- **Inicializando o valor de strings.**

Vejamos alguns exemplos:

```
char nome_cliente[30] = "Fulano";
```

```
char nome_cliente[30] = {'F','u','l','a','n','o'};
```



# STRING

- Inicializando uma string sem definir o tamanho do vetor

```
char nome_cliente[] = "Fulano";
```

Neste caso, a quantidade de caracteres de armazenamento é calculada automaticamente de forma a ter a dimensão exata para conter a string que está sendo atribuída.

Para armazenar “Fulano”, são necessários 6 caracteres + 1 para o finalizador \0.

Então podemos dizer que:

```
char nome_cliente[] = "Fulano";
```

equivale a declarar:

```
char nome_cliente[7] = "Fulano";
```



## EXERCÍCIO - SOLUÇÃO

Ana Santos

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

```
int main()  
{
```

```
    char nome[]="Ana",apelido[]="Santos" ;  
    printf("Nome: %s %s",nome,apelido);
```

```
    return 0;  
}
```

```
// para estar de acordo com o enunciado:  
printf("%s %s",nome,apelido);
```

Agora sim...

Nome: Ana Santos

Ana Santos

100%



## EXERCÍCIO

- Faça as alterações necessárias para que possa solicitar o nome e apelido ao utilizador.



## EXERCÍCIO

## SOLUÇÃO:

- Faça as alterações necessárias para que possa solicitar o nome e apelido ao utilizador.

```
#include <stdio.h>
int main()
{
    char nome[10],apelido[10];

    printf("Introduza o nome: ");
    scanf("%s",&nome);
    printf("Introduza o apelido: ");
    scanf("%s",&apelido);
    printf("Nome completo: %s %s",nome,apelido);

    return 0;
}
```

```
Introduza o nome: Sofia
Introduza o apelido: Neto
Nome completo: Sofia Neto
```



## WEBGRAFIA

- <https://blog.betrybe.com/tecnologia/paradigmas-de-programacao/>
- <https://www.cmmprogressivo.net/2019/09/C-ou-Cmaismais-Cpp-qual-melhor-qual-diferenca-precisa-saber-C.html>
- [Codeblocks:  
https://sourceforge.net/projects/codeblocks/](https://sourceforge.net/projects/codeblocks/)
- <https://www.dcc.fc.up.pt/~pbv/aulas/progimp/teoricas/teorica02.html>



# Sessão 3





## EXERCÍCIO DE REVISÃO

Elabore um programa para calcular a **área de uma circunferência**.

- A área de uma circunferência de raio  $r$  é:

$$A = \pi r^2$$

(onde  $\pi$  é a constante 3.14159...)

- O programa deve solicitar ao utilizador o valor do raio. (pois é a única incógnita que falta descobrir para resolver o problema.)



## EXERCÍCIO DE REVISÃO

# SOLUÇÃO:

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

```
#define PI 3.14159
```

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

```
    float raio, area;
```

```
    printf("Raio da circunferencia?");
```

```
    scanf("%f", &raio);
```

```
    area = PI * raio * raio;
```

```
    printf("Area: %f\n", area);
```

```
    return 0;
```

```
}
```

```
Raio da circunferencia?2
Area: 12.566360
```



# ALTERAÇÕES:

## EXERCÍCIO DE REVISÃO

- Faça os devidos ajustes para que o resultado obtido seja como o da figura:

```
Raio da circunferencia? 2
Circulo:
Raio: 2.00
Area: 12.57
```



## EXERCÍCIO DE REVISÃO

## SOLUÇÃO:

- Faça os devidos ajustes para que o resultado obtido seja como o da figura:

```
#include <stdio.h>

#define PI 3.14159

int main(void) {
    float raio, area;

    printf("Raio da circunferencia? ");
    scanf("%f", &raio);
    area = PI * raio * raio;
    printf("Circulo:\nRaio: %.2f\nArea: %.2f\n",raio, area);
    return 0;
}
```

```
Raio da circunferencia? 2
Circulo:
Raio: 2.00
Area: 12.57
```



## EXERCÍCIO

- Qual o resultado do seguinte programa?

```
#include <stdio.h>

int main(void)
{
    int x;
    scanf( "%d", &x );
    printf("%d\n", 4*x );

    return 0;
}
```



## EXERCÍCIO

## TESTE DE MESA:

Os testes de mesa ajudam-nos a verificar se o algoritmo (ou programa) nos leva a um resultado esperado através da simulação de valores.

```
*main.c X
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int x;
6      scanf( "%d", &x );
7      printf("%d\n", 4*x );
8
9      return 0;
10 }
11
```

Linha	x	4*x
6	2	?
7	2	8

2

8

Confirmar, executando o  
programa

C:\Users\alexa\C

2

8



# STRING

## STRLEN

A biblioteca padrão fornece várias funções úteis para manipular strings. A seguir mostraremos algumas delas. Para usá-las, devemos incluir o cabeçalho **string.h** no início dos seus arquivos.

**strlen**: retorna o tamanho, em caracteres, de uma string dada. Na verdade o `strlen()` procura o terminador de string e calcula a distância dele ao início da string.

Por exemplo:

```
char nome[15] = "Maria da Silva";  
int s = strlen (nome);  
// s conterá o valor 14
```



# STRING

## STRCPY

**strcpy** copia o conteúdo de uma string para outra e coloca um terminador de string.

A sua sintaxe é:

`strcpy (destino, origem)`

Exemplo:

```
char nome[] = "Maria Teresa";
```

```
char nome2[] = "Ana Luisa";
```

```
strcpy (nome, nome2);
```

```
// agora nome conterà "Ana Luisa"
```





## EXERCÍCIO

1. Elabore um programa que devolva o número de caracteres (comprimento da string) que tem um nome introduzido pelo utilizador.
2. Elabore um programa que apresente o resultado ilustrado na imagem seguinte:

```
Introduza a primeira palavra: caixa  
Introduza a segunda palavra: bola  
palavra1: bola    palavra2: bola
```

Nota: O utilizador apenas tem de introduzir as duas palavras.



## EXERCÍCIO

## SOLUÇÃO:

1. Elabore um programa que devolva o número de caracteres (comprimento da string) que tem um nome introduzido pelo utilizador.

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

```
int main(void)
```

```
{
```

```
    char nome[20];
```

```
    int s;
```

```
    printf("Introduza o nome: ");
```

```
    scanf("%s",&nome);
```

```
    s= strlen (nome);
```

```
    printf("Nome: %s tem %d caracteres.",nome,s);
```

```
    return 0;
```

```
}
```

```
Introduza o nome: soraia  
Nome: soraia tem 6 caracteres.
```



## EXERCÍCIO

## SOLUÇÃO:

2. Elabore um programa que apresente o resultado ilustrado na imagem seguinte:

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

```
int main(void)
```

```
{
```

```
    char palavra1[20], palavra2[20];
```

```
    printf("Introduza a primeira palavra: ");
```

```
    scanf("%s",&palavra1);
```

```
    printf("Introduza a segunda palavra: ");
```

```
    scanf("%s",&palavra2);
```

```
    // colocar a palavra2 no lugar da palavra1
```

```
    strcpy(palavra1,palavra2);
```

```
    printf("palavra1: %s  palavra2: %s",palavra1,palavra2);
```

```
    return 0;
```

```
}
```

```
Introduza a primeira palavra: caixa
Introduza a segunda palavra: bola
palavra1: bola      palavra2: bola
```



# STRING

## STRCAT

**strcat** concatena duas strings, adicionando o conteúdo da segunda ao final da primeira, além do terminador (\0). Note-se que a primeira string deve ter espaço suficiente para conter a segunda, para que não ocorra um "estouro de buffer".

Por exemplo:

```
char nome[50] = "Maria";
```

```
char sobrenome[] = " da Silva";
```

```
strcat (nome, sobrenome);
```

```
// agora nome contém "Maria da Silva"
```



# STRING

## STRCMP

Se tentarmos criar duas strings com o mesmo conteúdo e compará-las como faríamos com números, veremos que elas "não são iguais". Isso ocorre porque, na verdade, o que está sendo comparado são os *endereços de memória* onde estão guardadas as strings. Para comparar o *conteúdo* de duas strings, devemos usar a função **strcmp**

Sintaxe:

```
int strcmp (char *s1, char *s2);
```

- O valor de retorno é:
  - menor que zero se *s1* for menor que *s2*;
  - igual a zero se *s1* e *s2* são iguais;
  - maior que zero se *s1* for maior que *s2*.



# STRING

## STRCMP

Costuma parecer estranho dizer que uma string é *menor* ou *maior* que outra; na verdade essa comparação é entre a primeira letra que difere nas duas strings. Assim, se tivermos `s1 = "abc"` e `s2 = "abd"`, diremos que `s2` é **maior** que `s1`, pois na primeira posição em que as duas strings diferem, a letra em `s2` é "maior".

- É importante notar que a comparação feita por `strcmp` distingue maiúsculas de minúsculas. Isto é, as strings `"ABC"` e `"abc"` **não** são iguais para essa função.



## EXERCÍCIO



- Elabore um programa que permita solicitar ao utilizador o seu nome e apelido. Depois deve guardar em uma variável o nome completo e posteriormente mostrar a seguinte frase (exemplo):

*O nome completo do Silva é: Francisco Silva.*

apelido      nome completo



# SOLUÇÃO:

## EXERCÍCIO

*O nome completo do Silva é: Francisco Silva.*

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

```
int main(void)
```

```
{
```

```
    char nome[10], apelido[10];
```

```
    printf("Introduza o nome: ");
```

```
    scanf("%s",&nome);
```

```
    printf("Introduza o apelido: ");
```

```
    scanf("%s",&apelido);
```

```
    // concatenar o nome e apelido
```

```
    strcat(nome," ");
```

```
    strcat(nome,apelido);
```

```
    printf("O nome completo de %s e:
```

```
%s",apelido,nome);
```

```
    return 0;
```

```
Introduza o nome: Ana
Introduza o apelido: Santos
O nome completo de Santos e: Ana Santos
```





## EXERCÍCIO

## SOLUÇÃO:

Também poderíamos usar a função **sprintf**.

A diferença entre printf e sprintf é que printf retorna o resultado para a saída padrão (tela), enquanto sprintf retorna o resultado em uma variável. Isto é muito conveniente, porque podemos simplesmente digitar a frase que queremos ter e sprintf lida com a própria conversão e coloca o resultado na string que desejamos.

- Sintaxe:

```
char nome[10], apelido[10],completo[20];
```

```
sprintf( completo, "%s %s", nome, apelido);
```

```
printf("O nome completo de %s e: %s",nome,completo);
```



## EXERCÍCIO



- Faça as modificações necessárias para que agora o resultado seja:

```
Introduza o nome: Sara  
Introduza o apelido: Norte  
O nome completo de Sara e: Sara Norte
```



# SOLUÇÃO:

## EXERCÍCIO

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

```
int main(void)
```

```
{
```

```
    char nome[10], apelido[10], completo[20];
```

```
    printf("Introduza o nome: ");
```

```
    scanf("%s",&nome);
```

```
    printf("Introduza o apelido: ");
```

```
    scanf("%s",&apelido);
```

```
    // coloca o 'nome' e o 'apelido' numa variável 'completo'
```

```
    sprintf( completo, "%s %s", nome, apelido);
```

```
    printf("O nome completo de %s e: %s",nome,completo);
```

```
    return 0;
```

```
}
```

```
Introduza o nome: Sara  
Introduza o apelido: Norte  
O nome completo de Sara e: Sara Norte
```



## STRING

- Se pretendermos solicitar o nome completo ao utilizador?

```
#include <stdio.h>
void main()
{
    char nome[50]; //define um vetor de caracteres com tamanho máximo 50 posições
    printf("Digite seu nome completo: ");
    scanf("%s",&nome); // captura uma string do teclado
    printf("O nome digitado foi %s ",nome);
}
```

**O QUE ACONTECE?**



## LER STRING...

- O scanf necessita do <enter> como confirmação do dado (int, float, char, string, etc.) digitado para continuar.
- Se o utilizador digitar o nome completo, o scanf trunca a string assim que encontra o primeiro espaço em branco, porque o compilador entende que é o fim da string, imprimindo apenas o primeiro nome na tela.

```
Digite seu nome completo: Ana Lima  
O nome digitado foi Ana
```



## ENTRADA E SAÍDA PADRÃO

- Em C toda entrada e saída é feita com fluxos (*streams*) de caracteres organizados em linhas. Cada linha consiste de zero ou mais caracteres e termina com o caracter de final de linha. Pode haver até 254 caracteres em uma linha (incluindo o caracter de final de linha). Quando um programa inicia, o sistema operacional automaticamente define quem é a **entrada padrão** (geralmente o teclado) e quem é a **saída padrão** (geralmente a tela do monitor).



# ENTRADA E SAÍDA PADRÃO

- As facilidades de entrada e saída não fazem parte da linguagem C. O que existe é uma biblioteca padrão de funções para manipular a transferência de dados entre programa e os dispositivos (*devices*) de saída e entrada padrão. Algumas destas funções são: `scanf()`, `printf()`, `getchar()`, `puts()`, `gets()`. Estas funções são declaradas no arquivo `<stdio.h>`. Existem funções úteis para conversão e teste de caracteres declaradas no arquivo `<ctype.h>`.
- As funções de entrada e saída operam sobre *streams* (fluxos) de caracteres. Toda vez que uma função de entrada é chamada (por exemplo, `getchar()`, `scanf()`) ela verifica pela próxima entrada disponível na entrada padrão (por exemplo, texto digitado no teclado). Cada vez que uma função de saída é chamada, ela entrega o dado para a saída padrão (por exemplo, a tela).



# ENTRADA E SAÍDA PADRÃO

## ○ Entrada e saída de caracteres:

- `int getchar( void );`
- `int putchar( int );`

## ○ Entrada e saída de *strings*:

- `char *gets(char *);`
- `int puts( char *);`





# ENTRADA E SAÍDA PADRÃO

Suponhamos que queremos **ler um único caractere**, mas não queremos usar o `scanf()`. Isso pode ser feito usando a função **`getchar()`**.

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

```
int main()  
{
```

```
    char ch;
```

```
    printf("Digite algum caracter: ");
```

```
    ch = getchar();
```

```
    printf("\n A tecla pressionada eh %c.\n", ch);
```

```
}
```

```
Digite algum caracter: c
```

```
A tecla pressionada eh c.
```



# ENTRADA E SAÍDA PADRÃO

Suponhamos que queremos **escrever um único caractere**, mas não queremos usar o `printf()`. Isso pode ser feito usando a função **`putchar()`**.

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

```
int main()
```

```
{
```

```
    char ch;
```

```
    printf("Digite outro caracter: ");
```

```
    ch = getchar();
```

```
    putchar(ch);
```

```
}
```

```
Digite outro caracter: m  
m
```



# LER O NOME COMPLETO

Então podemos ler o nome completo usando o `gets()` em vez do `scanf()`.

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

```
void main()
```

```
{
```

```
char nome[50];
```

```
printf("Digite seu nome completo: ");
```

```
gets(nome); //lê uma string do teclado inclusive espaços em branco
```

```
puts(nome); // coloca a string digitada no teclado
```

```
}
```

```
Digite seu nome completo: Ana Lima  
Ana Lima
```



## ○ Sessão 4



## GETCH E GETCHE

- As funções `getch()` e `getche()` retornam o caracter pressionado.
- A função `getche()` imprime o caracter na tela antes de retorná-lo, enquanto que a função `getch()` apenas retorna o caracter sem imprimi-lo.
- Ambas são definidas no arquivo de cabeçalho `conio.h`, portanto, não pertencem ao padrão ANSI.
- A sintaxe destas funções é similar a da macro `getchar()`.



## EXERCÍCIO



- Elabore um programa que leia a morada do utilizador para uma variável 'temp' e depois deve copiá-la para a variável 'morada'. Deve depois imprimir a morada e indicar quantos caracteres possui.

*gets(string)*  
*strcpy(string destino, string origem)*  
*puts(string)*  
*int strlen (string)*

```
Digite a sua morada: rua da Sorte n.45
A sua morada contem 17 caracteres!rua da Sorte n.45
```



## EXERCÍCIO - RESOLUÇÃO

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

```
void main()
```

```
{
```

```
char temp[50], morada[50];
```

```
int x;
```

```
printf("Digite a sua morada: ");
```

```
gets(temp);
```

```
strcpy(morada,temp);
```

```
x= strlen(morada);
```

```
printf("A sua morada contem %d caracteres!",x);
```

```
puts(morada);
```

```
}
```

```
Digite a sua morada: rua da Sorte n.45
```

```
A sua morada contem 17 caracteres!rua da Sorte n.45
```



# OPERADORES E EXPRESSÕES ARITMÉTICAS

Os operadores aritméticos são símbolos utilizados para realizar as operações aritméticas elementares.

Operador	Operação
+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo (resto da divisão)
--	decremento
++	incremento





# OPERADORES E EXPRESSÕES ARITMÉTICAS

- Sintaxe de uma expressão aritmética:  
 $\text{<operando> } \textbf{<operador>} \text{ <operando>}$
- Operando é qualquer expressão válida
- Exemplos:
  - $a+b$
  - $1+2$
  - $(a+1) * 3$
  - $-6$
  - $\text{num} / 2$
  - $10 \% 5$



# OPERADORES E EXPRESSÕES ARITMÉTICAS

- Os operadores  $+$ ,  $-$ ,  $*$ ,  $/$  podem operar sobre números inteiros ou reais
- O operador  $\%$  aceita apenas operandos inteiros
- O denominador em uma operação de divisão deve ser diferente de 0
- Exemplos:
  - $6 + 2$  (8)
  - $2.0 + 2.0$  (4.0)
  - $6.5 \% 2$  (inválido, 6.5 não é int)
  - $10 / 0$  (inválido, divisão por 0)



## OPERADORES DE INCREMENTO E DECREMENTO

- Operadores aritméticos unários: aplicados a apenas a 1 operando

Operador	Operação
----------	----------

++	soma 1 ao seu operando (incremento)
----	-------------------------------------

--	subtrai 1 do seu operando (decremento)
----	----------------------------------------

- Geram instruções otimizadas (maior velocidade de processamento)
- Exemplos:
  - $x++$  é o mesmo que  $x+1$
  - $x--$  é o mesmo que  $x-1$



## OPERADORES DE INCREMENTO E DECREMENTO

- Podem ser utilizados como prefixo ou sufixo do operando
- Exemplos:
  - `x++` : usa o valor de `x` e posteriormente o incrementa
  - `++x` : incrementa o valor de `x` antes do uso do seu valor em uma expressão
  - `x=10; y = ++x; valor de y = 11;`
  - `x=10; y = x++; valor de y = 10`



## OPERADORES DE ATRIBUIÇÃO ARITMÉTICA

Operador	Equivalência
----------	--------------

$a += b$	$a = a + b$
----------	-------------

$a -= b$	$a = a - b$
----------	-------------

$a /= b$	$a = a / b$
----------	-------------

$a \% = b$	$a = a \% b$
------------	--------------

### ○ Exemplos:

- $\text{num} *= 3 \rightarrow \text{num} = \text{num} * 3$
- $\text{quantia} /= 10 \rightarrow \text{quantia} = \text{quantia} / 10$



## EXERCÍCIO

- Elabore um programa que solicite dois valores, efetue os seguintes cálculos e devolva os resultados.
  - soma:  $\text{valor1} + \text{valor2}$
  - subtração:  $\text{valor1} - \text{valor2}$
  - multiplicação:  $\text{valor1} \times \text{valor2}$
  - divisão:  $\text{valor1} / \text{valor2}$



# EXERCÍCIO



```
Indique o 1ro valor: 9
```

```
Indique o 2do valor: 2
```

```
***      Resultado das operacoes      ***
```

```
9.00 + 2.00 = 11.00
```

```
9.00 - 2.00 = 7.00
```

```
9.00 * 2.00 = 18.00
```

```
9.00 / 2.00 = 4.50
```



# EXERCÍCIO - RESOLUÇÃO

```
#include <stdio.h>
int main()
// recebe dois valores e efetua 4 operações distintas (+,-,*,/)
{
    float valor1, valor2, soma, subtr, multi, divisao;
    // solicita e armazena o 1ro valor
    printf("Indique o 1ro valor: ");
    scanf("%f",&valor1);
    // solicita e armazena o 2do valor
    printf("Indique o 2do valor: ");
    scanf("%f",&valor2);
    // efetua os cálculos
    soma= valor1+valor2;
    subtr = valor1-valor2;
    multi = valor1*valor2;
    divisao= valor1/valor2;
    // imprime os resultados
    printf("\n*** Resultado das operacoes ***\n");
    printf("\n%.2f + %.2f = %.2f",valor1, valor2,soma);
    printf("\n%.2f - %.2f = %.2f",valor1, valor2,subtr);
    printf("\n%.2f * %.2f = %.2f",valor1, valor2,multi);
    printf("\n%.2f / %.2f = %.2f",valor1, valor2,divisao);
    return 0;
}
```

```
Indique o 1ro valor: 9
Indique o 2do valor: 2
```

```
*** Resultado das operacoes ***
9.00 + 2.00 = 11.00
9.00 - 2.00 = 7.00
9.00 * 2.00 = 18.00
9.00 / 2.00 = 4.50
```





## CASTING – CONVERSÃO DE TIPO



- O (int) na frente de uma expressão aritmética é um *molde* (= *cast*).

Se x é uma variável do tipo double, a expressão:

- (int) (x/2) tem por valor a parte inteira de x/2.
- Assim, se x vale 9 então o valor da expressão é 4 (e não 4.5).

Outro exemplo de molde:

Se n é uma variável do tipo int então a expressão

- (double) n / 2 significa o mesmo que ((double) n) / 2

e transforma n em um número real antes de fazer a divisão por 2. Se o valor de n for 9 o valor da expressão será 4.5 (e não 4).



# CASTING

```
#include <stdio.h>

int main()
{
    float real;
    int inteiro;

    real= 9;
    inteiro=7;

    printf("real= %f inteiro= %d\n", real, inteiro);
    printf("\n\nINTEIRO");
    printf("\nopcao1: %d", inteiro/2);
    printf("\nopcao2: %f", inteiro/2);
    printf("\nopcao3: %f", (float) inteiro/2);
    printf("\n\nREAL");
    printf("\nopcao4: %d", real/2);
    printf("\nopcao5: %f", real/2);
    printf("\nopcao6: %d", (int) real/2);

    return 0;
}
```

```
C:\Users\alexa\OneDrive\Documents\Formap0o_Cenc
real= 9.000000 inteiro= 7

INTEIRO
opcao1: 3
opcao2: 0.000000
opcao3: 3.500000

REAL
opcao4: 0
opcao5: 4.500000
opcao6: 4
Process returned 0 (0x0)   e
Press any key to continue.
```

## CASTING

- Sendo:

- `int inteiro=7;`

- `printf("%d",inteiro/2);`

`%d` – indica que vai ser impresso um valor inteiro

`inteiro/2` – sendo ‘inteiro’ uma variável do tipo inteiro, então o resultado será retornado também em valor inteiro

- Resultado:

3



## CASTING

- Sendo:
  - `float real=9;`
  - `printf("%d",(int)real/2);`

`%d` – indica que vai ser impresso um valor inteiro

`real/2` – sendo 'real' uma variável do tipo real (float), então o resultado será retornado também em valor real.

- Será devolvido o valor 4.5, mas então o valor que o printf aguarda para ser impresso é um valor inteiro... Logo não irá aparecer o resultado esperado. Para que seja impresso o valor inteiro devemos moldar o valor real obtido na operação para um inteiro, ou seja a sua parte inteira. Para isso é necessário fazer o cast ao resultado da seguinte forma: `(int)real/2`



## EXERCÍCIO



- Efetue as alterações necessárias (cast) de forma a que os excertos de código fiquem corretos:

```
int a=4;
```

```
float b=7;
```

- 1) `printf("%d",b*2); // 14`
- 2) `printf("%d",a*b); // 28`
- 3) `printf("%f",a*2); // 8.000000`
- 4) `printf("%d",2.5*a); // 8`
- 5) `printf("%d",a*2.5); // 10`



## EXERCÍCIO - RESOLUÇÃO

- Efetue as alterações necessárias (cast) de forma a que os excertos de código fiquem corretos:

int a=4;

float b=7;

- 1) `printf("%d",b*2); // 14`      `printf("%d",(int)b*2);`
- 2) `printf("%d",a*b); // 28`      `printf("%d",(int)(a*b));`  
                                         `printf("%d",(int)b*a);`
- 3) `printf("%f",a*2); // 8.000000`      `printf("%f",(float)a*2);`
- 4) `printf("%d",2.5*a); // 8`      `printf("%d",(int)2.5*a);`
- 5) `printf("%d",a*2.5); // 10`      `printf("%d",(int)(a*2.5));`



# OPERADORES RELACIONAIS E LÓGICOS

Operadores Relacionais	
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual
==	igual
!=	diferente

Operadores Lógicos	
&&	'e' lógico (AND)
	'ou' lógico (OR)
!	'não' lógico (NOT)



# OPERADORES RELACIONAIS E LÓGICOS

O resultado de uma expressão lógica (ou relacional) é um valor numérico

- “verdadeiro”: é representado pelo valor 1 (um).
- “falso”: é representado pelo valor 0 (zero).





# TABELA DE VERDADE DOS OPERADORES

p	q	$p \& q$	$p \vee q$	$\neg p$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0



# PRECEDÊNCIA DOS OPERADORES

- Da maior precedência para a menor:

!

>, >=, <, <=

==, !=

&&

||



# OPERADORES RELACIONAIS E LÓGICOS

- $10 > 5 \ \&\& \ !(10 < 9) \ || \ 3 \leq 4$

**EXEMPLOS**

Ordem de avaliação:

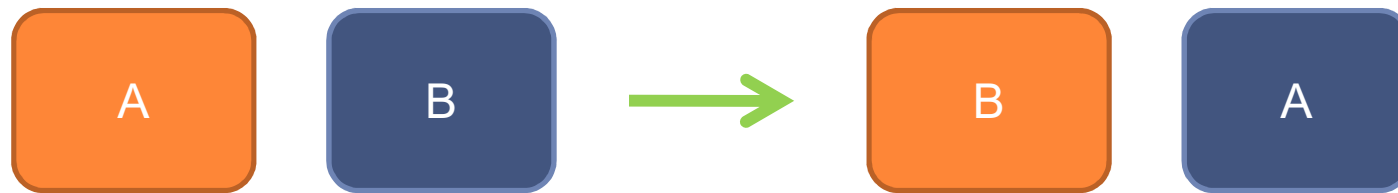
$10 > 5$  (1),  $1 \ \&\& \ 1$ ,  $3 \leq 4$  (1),  $1 \ || \ 1 = 1$  (verdadeira)

- $!0 \ \&\& \ 0 \ || \ 0$  (falsa)
- $!(0 \ \&\& \ 0) \ || \ 0$  (verdadeiro)
- $4 + 1 == 5$  (verdadeiro) ordem: +, ==
- $4 != 2 * 4 > 5$  (verdadeiro) ordem: \*, >, !=



## EXERCÍCIO

- Elabore um algoritmo que permita trocar o valor contido em duas variáveis.



- Elabore um programa que implemente o algoritmo anterior.



## EXERCÍCIO - ALGORITMO

**Algoritmo** troca convencional

**Variáveis** x,y,z : inteiro

**Início**

Ler (x,y)

Copiar o conteúdo da variável x para a variável z

Copiar o conteúdo de y para x

Copiar o conteúdo da variável z para y

Escrever (x,y)

Fim.



## EXERCÍCIO – PROGRAMA EM C

```
#include <stdio.h>
void main(){

    int x,y,z;

    printf("Valor de x: ");
    scanf("%d",&x);
    printf("Valor de y: ");
    scanf("%d",&y);

    z=x;
    x=y;
    y=z;
    printf("X= %d   Y= %d",x,y);

}
```



# WEBGRAFIA

## ○ Printf e scanf

- [https://pt.wikibooks.org/wiki/Programar\\_em\\_C/Entrada\\_e\\_sa%C3%ADda\\_simples](https://pt.wikibooks.org/wiki/Programar_em_C/Entrada_e_sa%C3%ADda_simples)

## ○ Funções strings

- [https://pt.wikibooks.org/wiki/Programar\\_em\\_C/Strings](https://pt.wikibooks.org/wiki/Programar_em_C/Strings)
- <https://www.ime.usp.br/~pf/algoritmos/apend/ctype.h.html>

## ○ Operadores e expressões

- [https://docente.ifrn.edu.br/brunogurgel/disciplinas/2012/fprog/aulas/cpp/aula3-operadores\\_e\\_expressoes.pdf](https://docente.ifrn.edu.br/brunogurgel/disciplinas/2012/fprog/aulas/cpp/aula3-operadores_e_expressoes.pdf)



## ○ Sessão 5





## BLOCOS DE COMANDOS

- Os delimitadores { e } são necessários sempre que existe mais de um comando num bloco

```
{  
  comando1;  
  comando2;  
  comando3;  
}
```



# ESTRUTURAS DE CONDIÇÃO SIMPLES

- Comando **if**

```
if (condição)  
    comando;
```

```
if (condição) {  
    comando1;  
    comando2;  
    comando3;  
}
```

```
if (a<menor)  
    menor=a;
```

```
if (a<menor) {  
    menor=a;  
    printf("%d ",menor);  
}
```



# ESTRUTURA CONDICIONAL COMPOSTA

- Comando **if.. else**

```
if (condição)
    comando1;
else
    comando2;
```

```
if (a<b)
    printf("a<b");
else
    printf("a>=b");
```



# ESTRUTURA CONDICIONAL ENCADEADA

- Comando **if.. else.. if**

```
if (condição1)
    comando1;
else if (condição2)
    comando2;
else
    condição;
```

```
if (a<b)
    printf("a<b");
else if(a>b)
    printf("a>b");
else
    printf("a=b");
```



## EXERCÍCIO 1

- Elabore um programa que solicita dois valores ao utilizador e depois diga qual o maior valor, ou se são iguais.



# EXERCÍCIO 1- SOLUÇÃO

- Elabore um programa que solicita dois valores ao utilizador e depois diga qual o maior valor, ou se são iguais.

```
#include <stdio.h>

void main(){

int x,y;

printf("Valor de x: ");
scanf("%d",&x);
printf("Valor de y: ");
scanf("%d",&y);

if (x>y)
    printf("X eh maior!");
else if(y>x)
    printf("Y eh maior!");
else
    printf("Sao iguais!!");

}
```



## EXERCÍCIO 2

- Faça um programa que leia um valor inteiro e mostre uma mensagem indicando se esse valor é par ou ímpar.



## EXERCÍCIO 2

- Faça um programa que leia um valor inteiro e mostre uma mensagem indicando se esse valor é par ou ímpar.

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

```
void main(){
```

```
int valor;
```

```
printf("Qual o valor? ");  
scanf("%d",&valor);
```

```
if (valor%2==0)  
    printf("O valor eh par");  
else  
    printf("O valor eh impar");
```

```
}
```





## EXERCÍCIO 3

- Elabore um programa que solicite três valores diferentes e devolva qual o maior valor



## EXERCÍCIO 3

- Elabore um programa que solicite três valores diferentes e devolva qual o maior valor



## ○ Sessão 6



# ESTRUTURA DE REPETIÇÃO

- Comando **for**

```
for (var=valor inicial; condição; incremento)  
    comando;
```

```
for (var=valor inicial; condição; incremento)  
{  
    comando1;  
    comando2;  
    comando3;  
  
}
```

```
for (i=0; i<3;i++)  
    printf(“%d”,i);  
//imprime 0,1,2
```



## EXEMPLO

- Vamos criar um programa que imprima todos os valores inteiros de 1 a 10.

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

```
void main()  
{
```

```
    int conta;  
    for (conta=1; conta<11;conta++)  
        printf("\n%d",conta);
```

```
}
```



## EXERCÍCIO 4




- Elabore um programa que calcule  $N!$  (fatorial de  $N$ ), sendo que o valor inteiro  $N$  é fornecido pelo utilizador. Sabendo que:
  - $N! = 1 \times 2 \times 3 \times \dots \times (N-1) \times N$
  - $0! = 1$ , por definição



## EXERCÍCIO 4 - SOLUÇÃO

```
void main (void){  
    // fatorial n!=1*2*3*...*n  
  
    int fat=1,n,i;  
  
    printf("Pretende qual fatorial? ");  
    scanf("%d",&n);  
  
    for(i=2;i<=n;i++)  
  
        fat= fat*i;  
    printf("Resposta: %d!= %d",n,fat);  
}
```

 "C:\Users\dora\Documents\0809 exercicios\aula2\bin\[

```
Pretende qual fatorial? 5  
Resposta: 5!= 120
```

```
Pretende qual fatorial? 12  
Resposta: 12!= 479001600
```

```
Pretende qual fatorial? 3  
Resposta: 3!= 6
```

```
Pretende qual fatorial? 9  
Resposta: 9!= 362880
```



## EXERCÍCIO 5



- Sendo  $h = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{N}$ , elabore um programa para calcular o número  $h$ , sendo o número  $N$  fornecido pelo utilizador.





## EXERCÍCIO 5-RESOLUÇÃO

```
void main (void){  
    // h= 1+1/2+1/3+...+1/n  
  
    int i,n;  
    float h=1;  
  
    printf(" Insira valor? ");  
    scanf("%d",&n);  
  
    for(i=2;i<=n;i++)  
        h= h+1/(float)i;  
  
    printf("Resposta: h= %.2f",h);  
}
```

```
Insira valor? 4  
Resposta: h= 2.08  
  
Insira valor? 7  
Resposta: h= 2.59  
  
Insira valor? 11  
Resposta: h= 3.02  
  
Insira valor? 23  
Resposta: h= 3.73
```



## EXERCÍCIO 6



- Faça um programe que apresente na tela uma tabela de conversão de graus Celsius para Fahrenheit, de -100C a 100C. Use um incremento de 10C.
  - Sabendo que:  $Fahrenheit = (9/5) * (Celsius) + 32$



## EXERCÍCIO 6 - RESOLUÇÃO

```
void main (void){  
    // converter celsius (-100 a 100) em fahrenheit (10 em 10)  
    // Fahrenheit = (9/5) * (Celsius)+32
```

```
    int i;  
    float fahrenheit;
```

```
    for(i=-100;i<=100;i+=10){
```

```
        fahrenheit=((float)9/5)*i+32;//ou i*9/5+32;  
        printf("Celsius: %d  Fahrenheit: %.2f\n",i,fahrenheit);
```

```
    }  
}
```

Celsius: -100	Fahrenheit: -148.00
Celsius: -90	Fahrenheit: -130.00
Celsius: -80	Fahrenheit: -112.00
Celsius: -70	Fahrenheit: -94.00
Celsius: -60	Fahrenheit: -76.00
Celsius: -50	Fahrenheit: -58.00
Celsius: -40	Fahrenheit: -40.00
Celsius: -30	Fahrenheit: -22.00
Celsius: -20	Fahrenheit: -4.00
Celsius: -10	Fahrenheit: 14.00
Celsius: 0	Fahrenheit: 32.00
Celsius: 10	Fahrenheit: 50.00
Celsius: 20	Fahrenheit: 68.00
Celsius: 30	Fahrenheit: 86.00
Celsius: 40	Fahrenheit: 104.00
Celsius: 50	Fahrenheit: 122.00
Celsius: 60	Fahrenheit: 140.00
Celsius: 70	Fahrenheit: 158.00
Celsius: 80	Fahrenheit: 176.00
Celsius: 90	Fahrenheit: 194.00
Celsius: 100	Fahrenheit: 212.00

## ○ Sessão 7



## EXERCÍCIO 7



- Escreva um programa que coloque os números de 1 a 100 na tela na ordem inversa (começando em 100 e terminando em 1).



## EXERCÍCIO 7 - RESOLUÇÃO

```
void main(){  
  
    int i;  
  
    for (i=100;i>=1;i--)  
        printf("%d\n",i);  
  
}
```

```
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74  
73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 4  
6 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19  
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1  
Process returned 2 (0x2)    execution time : 1.998 s  
Press any key to continue.
```



## EXERCÍCIO 8



- Elabore um programa que permita calcular e listar todos os múltiplos positivos de um número dado pelo utilizador, menores ou iguais a 100.



## EXERCÍCIO 8 - RESOLUÇÃO

```
void main(){  
// multiplos menores/igual a 100  
int i,n;
```

```
printf("Qual o valor? ");  
scanf("%d",&n);
```

```
for (i=0;i<=100;i+=n)
```

```
    printf("%d\n",i);
```

```
}
```

```
Qual o valor? 12  
0  
12  
24  
36  
48  
60  
72  
84  
96
```





## EXERCÍCIO 9



- Crie um programa que imprima todos os números pares compreendidos entre dois valores dados pelo utilizador.



## EXERCÍCIO 9 - RESOLUÇÃO

```
void main(){

int min,max,soma,i;
printf("Quais os numeros pares compreendidos entre dois
valores?\n");
printf("Introduza o minimo: ");
scanf("%d",&min);
printf("Introduza o maximo: ");
scanf("%d",&max);

if(min%2!=0)
    min=min+1;
for(i=min;i<=max;i+=2)
    printf("\n%d",i);

}
```

```
Quais os numeros pares compreendidos entre dois valores?
Introduza o minimo: 3
Introduza o maximo: 16

4
6
8
10
12
14
16
```



# ESTRUTURA WHILE

Qualquer ciclo *for* pode ser sempre escrito como um ciclo *while*.

```
for( expressão1; expressão2; expressão3 )  
    instrução;
```

*é equivalente a:*

```
expressão1;  
while( expressão2 ) {  
    instrução;  
    expressão3;  
}
```



# ESTRUTURA WHILE

O funcionamento é o seguinte:

1. Testa a condição;
2. Se a **condição** for **falsa** então pula todos os comandos do bloco subordinado ao **while** e passa a executar os comandos após o bloco do **while**.
3. Se **condição** for **verdadeira** então executa cada um dos comandos do bloco subordinado ao **while**.
4. Após executar o último comando do bloco do **while** volta ao passo 1.

```
while (condição) {  
  
    instruções a serem repetidas  
}  
instruções após o while
```



# ESTRUTURA WHILE

O comando **while** deve ser usado sempre que:

- **não soubermos exatamente quantas vezes o laço deve ser repetido;**
- o teste deva ser feito **antes de iniciar** a execução de um bloco de comandos;
- houver casos em que o laço **não deva ser repetido nenhuma vez.**



# ESTRUTURA WHILE - EXEMPLO

Programa para calcular a soma dos N primeiros números naturais ( $1+2+3+\dots+N$ ) em que N é um número introduzido pelo utilizador.

```
#include <stdio.h>
void main()
{
    int i, s, n;
    printf("Introduz N:\n");
    scanf("%d", &n);
    s = 0;
    for( i = 1; i <= n; i++ )
        s = s + i;
    printf("A soma é %d\n", s);
}
```



Reescreva o programa substituindo a estrutura de repetição for por while



## ESTRUTURA WHILE - EXEMPLO

```
void main()
{
    int i=1, s, n;
    printf("Introduz N:\n");
    scanf("%d", &n);
    s = 0;

    while (i <= n){
        s = s + i;
        i++;
    }

    printf("A soma eh %d\n", s);
}
```

```
Introduz N:
4
A soma eh 10
```



## EXERCÍCIO



- Elabore um programa que permita calcular a média de idades de uma série de indivíduos. A finalização de entrada de dados é dada pela introdução do valor 0.





## RESOLUÇÃO

```
void main(){
int n=0, idade=1,soma=0;
float media;

printf("Calcula media de idades!\n");
printf("Prima 0 para terminar.\n\n");
while(idade>0){
    printf("Introduza idade: ");
    scanf("%d",&idade);
    soma+= idade;
    n++;
}
media= soma/(n-1);
printf("A media das idades eh: %.2f", media);

}
```

```
Calcula media de idades!
Prima 0 para terminar.

Introduza idade: 20
Introduza idade: 30
Introduza idade: 50
Introduza idade: 0
A media das idades eh: 33.00
```



## ○ Sessão 8



## ESTRUTURA DO WHILE

```
do
{
    //comandos a serem repetidos
    //comandos a serem repetidos
} while (condição);
// comandos após o 'do-while'
```

O funcionamento é o seguinte:

1. Executa os comando dentro do bloco **do-while**;
2. Testa a condição;
3. Se a **condição** for **falsa** então executa o comando que está logo após o bloco subordinado ao **do-while** .
4. Se **condição** for **verdadeira** então volta ao passo 1.



## ESTRUTURA DO WHILE

O comando **do-while** deve ser usado sempre que:

- não soubermos exatamente quantas vezes o laço deve ser repetido;
- o teste deva ser feito **depois** da execução de um bloco de comandos;
- o bloco de comandos deve **ser executado pelo menos 1 vez**;



## EXEMPLO – DO WHILE

```
void main(){

    int continua, contador;
    contador = 0;

    // nao precisamos inicializar a variável 'continua'
    // pois o teste é feito depois

    do
    {
        // comandos a serem repetidos

        printf("Repentindo....\n");

        contador = contador + 1;

        printf("Tecle 's' se deseja continuar\n");
        continua = getch();
    } while (continua == 's');

    printf("O bloco foi repetido %d vezes", contador);

}
```



## EXERCÍCIOS



1. Elabore um programa que permita a leitura de um número inteiro positivo. No final devolve esse mesmo valor.
2. Elabore um programa que receba um valor compreendido entre 10 e 60 (inclusive).

**Observação :** A validação dos dados consiste em verificar se o utilizador introduziu valores dentro dos limites impostos ...



# RESOLUÇÃO Nº1

```
void main(){  
  
int num;  
  
do{  
    printf("\nIntroduza um valor positivo: ");  
    scanf("%d",&num);  
  
}while(num<=0);  
  
printf("valor introduzido foi: %d",num);  
  
}
```

```
Introduza um valor positivo: -9  
Introduza um valor positivo: -2  
Introduza um valor positivo: 3  
valor introduzido foi: 3
```



## RESOLUÇÃO Nº2

```
#include <stdio.h>
#define MAX 60
#define MIN 10
```

```
void main(){
```

```
float num;
```

```
do{
```

```
    printf("\nIntroduza valor entre %d e %d!",MIN,MAX);
    scanf("%f",&num);
```

```
}while(num <MIN || num >MAX);
printf("Valor aceite: %.2f",num);
}
```

```
Introduza valor entre 10 e 60!4
Introduza valor entre 10 e 60!78
Introduza valor entre 10 e 60!60.1
Introduza valor entre 10 e 60!59.9
Valor aceite: 59.90
```







## EXERCÍCIO I

- Escreva um programa que solicite 10 números inteiros positivos ao utilizador, e no final mostre o menor e o maior valor introduzidos.

Obs. O utilizador irá digitar um valor de cada vez.

### **Exemplo:**

Entrada: 12 33 4 78 11 29 3 9 44 15

Saída: Min: 3      Max: 78



# SOLUÇÃO I

```
void main(){  
  
    int i,num,min=32800,max=0;  
  
    for(i=0;i<10;i++){  
  
        printf("\nDigite um valor: ");  
        scanf("%d",&num);  
        if(num<min)  
            min=num;  
        if(num>max)  
            max=num;  
    }  
    printf("Min: %d Max: %d", min,max);  
  
}
```

```
Digite um valor: 12  
Digite um valor: 33  
Digite um valor: 4  
Digite um valor: 78  
Digite um valor: 11  
Digite um valor: 29  
Digite um valor: 3  
Digite um valor: 9  
Digite um valor: 44  
Digite um valor: 15  
Min: 3 Max: 78
```



## EXERCÍCIO II



- Escreva um programa que solicite 10 números ao utilizador, e no final mostre os dois maiores valores introduzidos.

Obs. O utilizador irá digitar um valor de cada vez.

### **Exemplo:**

Entrada: 12 33 4 78 11 29 3 9 44 15

Saída: 78 44



## SOLUÇÃO II

```
void main(){
    int i,num,maxx=0,max=0;

    for(i=0;i<10;i++){
        printf("\nDigite um valor: ");
        scanf("%d",&num);
        if(num>maxx){
            max=maxx;
            maxx=num;
        }else
            if(num>max)
                max=num;
    }
    printf("Os dois maiores valores foram: %d e %d", maxx,max);
}
```

```
11 Digite um valor: 5
12
13 Digite um valor: 12
14
15 Digite um valor: 41
16
17 Digite um valor: 23
18
19 Digite um valor: 64
20
21 Digite um valor: 15
22
23 Digite um valor: 22
24
25 Digite um valor: 33
26
27 Digite um valor: 19
28
29 Digite um valor: 27
30 Os dois maiores valores foram: 64 e 41
```



## EXERCÍCIO III



- Escreva um programa que simule uma calculadora que realiza apenas operações de multiplicação e divisão. No entanto, para realizar tais operações, só podemos usar apenas adições e subtrações (não podemos fazer  $x*y$  ou  $x/y$ ). Dados dois valores inteiros, devolve o resultado da operação desejada entre eles.

```
Introduza um valor:
45
Introduza outro valor: 7
Escolha a operacao 1(*) 2(/): 2
45 / 7 = 6.00 e sobram: 3
```



# SOLUÇÃO III

```
void main()
{

    int i,num1,num2,op,aux;
    float resultado=0;

    printf("Introduza um valor: ");
    scanf("%d",&num1);
    printf("Introduza outro valor: ");
    scanf("%d",&num2);

    do
    {
        printf("Escolha a operacao 1(*) 2(/): ");
        scanf("%d",&op);
    }
    while(op!=1 && op!=2);

    if(op==1)
    {
        // calcular a multiplicação dos dois valores

        for(i=0; i<num1; i++)
            resultado = resultado + num2;
        printf("%d * %d = %.2f",num1,num2,resultado);
    }
```

```
else
{
    // calcular a divisão dos dois valores
    if(num2>num1)
    {
        aux=num1;
        num1=num2;
        num2=aux;
    }
    aux=num1;
    do
    {
        aux=aux-num2;
        resultado+=1;
    }
    while(aux>=num2);
    printf("%d / %d = %.2f e sobram:
        %d",num1,num2,resultado,aux);
}
}
```



## ESTRUTURA ALTERNATIVA - SWITCH

Já estudámos a instrução if-else. Essa instrução permite que o computador prossiga por um caminho ou por outro mediante o valor de uma condição ser verdadeiro ou falso. Por essa razão, costuma designar-se por instrução de selecção ou de decisão simples.

Contudo, existem situações em que queremos que o computador escolha um caminho de entre N **alternativas** possíveis. Nesses casos temos uma **escolha múltipla**. Para conseguir tal efeito, podemos programar o computador utilizando uma sequência encadeada de instruções if-else. No entanto, a linguagem C tem uma instrução própria para este tipo de situações: a instrução **switch**.



# SWITCH - SINTAXE

**switch** (variável)

```
{  
  case constante1:  
    Instruções;  
    break;  
  case constante2:  
    Instruções;  
    break;  
  default Instruções;  
}
```

*O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.*

- **Observações:**

*O parâmetro do switch deve ser **int** ou **char***

*O valor após o case deve ser uma **CONSTANTE***

**Muito Cuidado:** A falta do **break** faz o processador continuar para o próximo **case**





# SWITCH - EXEMPLO

```
#include <stdio.h>
#include <conio.h>
int main (void )
{
    int valor;
    printf ("Digite um valor de 1 a 7: ");
    scanf("%d", &valor);

    if(valor==1) printf ("Domingo\n");
    else if(valor==2) printf ("Segunda\n");
    else if(valor==3) printf ("Terca\n");
    else if(valor==4) printf ("Quarta\n");
    else if(valor==5) printf ("Quinta\n");
    else if(valor==6) printf ("Sexta\n");
    else if(valor==7) printf ("Sabado\n");
    else printf ("Valor invalido!\n");

    getch();
    return 0;
}
```



# SWITCH - EXEMPLO

```
#include <stdio.h>
#include <conio.h>
int main (void )
{
    int valor;

    printf ("Digite um valor de 1 a 7: ");
    scanf ("%d", &valor);

    switch ( valor )
    {
        case 1 :
            printf ("Domingo\n");
            break;

        case 2 :
            printf ("Segunda\n");
            break;

        case 3 :
```

```
        case 4 :
```

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

```
        case 5 :
```

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

```
        case 6 :
```

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

```
        case 7 :
```

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

```
        default :
```

```
            printf ("Valor invalido!\n");
```

```
    }
```

```
    getch(); // lê caracter – biblioteca conio
    return 0;
```

```
}
```



## COMANDOS DE DESVIOS

- Os laços em C possuem dois comandos de desvios: **break** e **continue**.
- Estes comandos funcionam com todos os comandos de repetição.



## COMANDOS DE DESVIOS

- O comando **break** encerra o laço no ponto em que for executado.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int contador, numero;
    for (contador = 1; contador <= 3; contador = contador+1)
    {
        scanf("%d",&numero);
        if (numero < 0)
            break; // encerra o laço neste ponto !!!
    }
    printf ("Foram digitados %d numeros validos", contador-1);
    getch();
}
```

- Este programa lê no máximo 3 números positivos
- Caso um número negativo seja digitado no meio da sequência o laço é encerrado de imediato



# COMANDOS DE DESVIOS

- O comando **continue** desvia o fluxo para o início do laço.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int contador, numero;
    clrscr();
    for (contador = 1; contador <= 5; contador = contador+1)
    {
        printf("Inicio do laco. Passo %d\n", contador);
        if ((contador % 2) == 0)
        {
            printf("Terminado antes do tempo....\n");
            continue;
        }
        printf ("Final do Laco. Passo %d\n", contador);
    }
    getch();
}
```

## RESULTADO:

```
Inicio do laco. Passo 1
Final do Laco. Passo 1
Inicio do laco. Passo 2
Terminado antes do tempo....
Inicio do laco. Passo 3
Final do Laco. Passo 3
Inicio do laco. Passo 4
Terminado antes do tempo....
Inicio do laco. Passo 5
Final do Laco. Passo 5
```



## EXERCÍCIO - CALCULADORA



- Escreva um programa que permita efetuar as quatro operações básicas (soma, subtração, multiplicação e divisão) entre dois valores. O utilizador introduz dois valores e também qual a operação que pretende efetuar. O programa devolve o resultado.



Observção : o programa deve estar preparado para quando o utilizador se enganar na operação... Deve voltar a solicitar uma operação válida.



# CURIOSIDADES

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main(void)
{
    int i;
    printf("Gerando 10 valores aleatorios:\n\n");
    for (i = 0; i < 10; i++)
    {
        /* gerando valores aleatórios entre zero e 100 */
        printf("%d ", rand() % 100);
    }
    getch();
    return 0;
```



# CURIOSIDADES

## **Escolhendo uma faixa de números aleatórios em C**

Ok, agora você é mestre nas artes de geração aleatória de números em linguagem C. Porém, seu código gera números entre 0 e RAND\_MAX.

E se você quiser gerar entre 0 e 10? Ou entre 1 e mil?  
Ou entre 0.00 e 1.00?

Para escolher a faixa de valores vamos usar operações matemáticas, principalmente o operador de módulo, também conhecido como resto da divisão: %

Para fazer com que um número 'x' receba um valor entre 0 e 9, fazemos:

```
x = rand() % 10
```

Para fazer com que um número 'x' receba um valor entre 1 e 10, fazemos:

```
x = 1 + ( rand() % 10 )
```

Para fazer com que um número 'x' receba um valor entre 0.00 e 1.00, primeiro geramos números inteiros, entre 0 e 100:

```
x = rand() % 101
```

Para ter os valores decimais, dividimos por 100:

```
x = x/100;
```





# CURIOSIDADES

## **A função** `void srand (unsigned int seed)`

- Cada elemento de uma sequência pseudo-aleatória é gerado a partir do elemento anterior.
- Como pode ser desejável repetir uma sequência pseudo-aleatórios em C sempre usa o mesmo primeiro elemento.
- A função `void srand (unsigned int seed)` permite variar esse primeiro elemento, que serve como semente da sequência.

## ***Usando o relógio como semente***

- A biblioteca `time` possui uma função `time` cujo resultado é um número de segundos transcorridos desde um momento fixado (00:00:00 UTC de 1 de janeiro de 1970).
- Esse número pode ser usado como `seed` na chamada de `srand` para gerar uma semente variável e imprevisível.
  - O resultado de `time` tem um tipo `time_t` e deve ser convertido em `unsigned int` para que possa ser usado como argumento para `srand`.
  - A função `time` também requer um argumento que ao efeito dessa aplicação pode ser passado como `null`
- `srand((unsigned) time(NULL))`



## ○ Sessao 9



## O que são?

### FUNÇÕES

Uma função nada mais é do que uma subrotina usada em um programa.

Na linguagem C, denominamos função a um conjunto de comandos que realiza uma tarefa específica em um módulo dependente de código.

A função é referenciada pelo programa principal através do nome atribuído a ela.

A utilização de funções visa modularizar um programa, o que é muito comum em programação estruturada.

Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa bem definida.



## Porquê usá-las?

### FUNÇÕES

- Para permitir o reaproveitamento de código já construído (por nós ou por outros programadores);
- Para evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, difíceis de entender;
- Para facilitar a leitura do programa-fonte;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.



## FUNÇÕES

```
tipo_retorno_da_funcao NomeDaFuncao (Lista_de_Parametros)
{
    // corpo da função
    // retorno da função
}
```

- A Lista\_de\_Parametros, também é chamada de Lista\_de\_Argumentos, é opcional.



## Exemplo

### FUNÇÕES

Vamos fazer uma função `abs` para calcular o valor absoluto de um número. Depois de definida, podemos usar essa função onde entendermos.

```
#include <stdio.h>

void main()
{
    float a;

    printf("Introduz um número: ");
    scanf("%f", &a );
    printf("O valor absoluto de %f é %f\n", a, abs(a) );
}
```



## Exemplo

### FUNÇÕES

O programa não funciona tal como está porque a linguagem C não conhece a **função abs**. No entanto, podemos definir a função abs do seguinte modo:

```
float abs( float x )  
{  
    if( x < 0 )  
        return -x;  
    else  
        return x;  
}
```



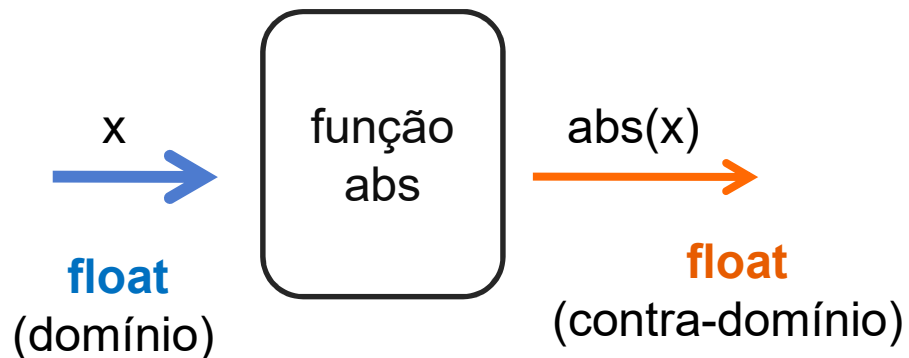
## Exemplo

### FUNÇÕES

Vamos analisar a primeira linha da definição da função abs:

**float** abs( **float** x )

O float que aparece antes da palavra abs e o float que aparece antes de x correspondem na Matemática ao contra-domínio e ao domínio da função.





## Exemplo

### FUNÇÕES

A função em si é uma espécie de caixa preta que recebe um valor real  $x$  e manda cá para fora um outro valor real que é o valor absoluto de  $x$ . Dentro da função propriamente dita, podemos utilizar quaisquer instruções da linguagem C. A instrução `return` serve para "mandar cá para fora" o valor da função e terminar a sua execução.



## Exemplo

# FUNÇÕES

```
#include <stdio.h>

/* função para calcular o valor absoluto de um número */
float abs( float x )
{
    if( x < 0 )
        return -x;
    else
        return x;
}

main()
{
    float a;
    printf("Introduz um número: ");
    scanf("%f", &a );
    printf("O valor absoluto de %f é %f\n", a, abs(a) );
}
```

O x que aparece na definição da função abs chama-se **parâmetro formal**. A variável **a** que aparece quando a função é chamada chama-se **parâmetro actual**. Na altura em que a função é chamada o parâmetro formal recebe uma cópia do valor do parâmetro actual (no exemplo, x recebe uma cópia do valor de a).



## PARÂMETROS DE UMA FUNÇÃO

- Os parâmetros são as variáveis declaradas diretamente no cabeçalho da função.
- A finalidade dos parâmetros é fazer a comunicação entre as funções e a função principal.
- Chamamos de passagem de parâmetros a passagem de valores entre as funções.



## Exercício

### FUNÇÕES

- Desenvolva um programa que permita a multiplicação de dois valores. A operação aritmética deve ser feita em uma função:
  - Nome da função: multiplica
  - recebe como parâmetros dois valores inteiros (N1,N2)
  - objetivo: multiplicar os valores recebidos nos parâmetros.
  - retorno: um parâmetro inteiro (res) contendo o resultado



## ○ Sessão 9



# RESOLUÇÃO DO EXERCÍCIO

```
#include <stdio.h>

int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int
{
    int resultado;
    resultado = N1 * N2;
    return(resultado); //retornando o valor para main
}
/***** função principal (main) *****/
int main(void)
{
    int V1, V2, resultado;
    printf("Digite o primeiro valor:");
    scanf("%d", &V1);
    printf("Digite o segundo valor:");
    scanf("%d", &V2);
    //chama a função e recebe o retorno
    resultado = multiplica(V1,V2);
    printf("\n\nResultado = %d\n", resultado);

    getch();
    return 0;
}
```



# FUNÇÕES E ESCOPO DAS VARIÁVEIS

Chamamos de escopo de variável ao conjunto de regras que determinam a utilização de uma variável em um programa

Podemos dividir as variáveis quanto ao escopo em três tipos:

- variáveis locais
- parâmetros formais
- variáveis globais



# FUNÇÕES E ESCOPO DAS VARIÁVEIS

## ○ **Variáveis locais**

- São aquelas declaradas dentro do bloco de uma função.
- Não podem ser usadas ou modificadas por outras funções.
- Somente existem enquanto a função onde foi declarada estiver sendo executada.

## ○ **Parâmetros formais**

- Os parâmetros formais de uma função também são variáveis locais da função.





# FUNÇÕES E ESCOPO DAS VARIÁVEIS

## ○ **Variáveis Globais**

- São declaradas fora de todos os blocos de funções.
- São acessíveis em qualquer parte do programa, ou seja, podem ser usadas e modificadas por todas as outras funções.
- Existem durante toda a execução do programa.



# FUNÇÕES E ESCOPO DAS VARIÁVEIS

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

```
//declaração de variáveis globais
```

```
// ----- Função main()-----  
int main(void)  
{  
    //declaração das variáveis locais da main()  
    return(0);  
}  
// -----
```

```
void funcao1(variáveis locais de parâmetros)  
{  
    // declaração das variáveis locais da função1  
  
    return;  
}
```



//Exemplo de programa com variáveis globais e locais.

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

```
#include<conio.h>
```

**//declaração de variáveis globais**

```
float media, nota1, nota2;
```

**//protótipo da função entrada**

```
void entrada(void);
```

**// ----- função main()-----**

```
int main(void)
```

```
{
```

**//variável local**

```
char resposta;
```

```
do
```

```
{
```

```
    //chamada da função p/ entrada das notas  
    entrada();
```

**//usando variáveis globais: media,nota1,nota2**

```
media = (nota1 + nota2) / 2;
```

```
printf("\nMedia do aluno: %.2f\n", media);
```

```
printf("\nDeseja calcular outra media? (s/n)");
```

```
fflush(stdin); // limpar buffer
```

```
scanf("%c",&resposta);
```

```
}
```

```
while(resposta == 's');
```

```
return(o);
```

```
}
```

**// ----- fim da função main() -----**

### **Notas:**

**Podem existir variáveis locais e globais com o mesmo nome?**

Sim. Caso isto ocorra, as variáveis irão se comportar como variáveis diferentes, embora possuam o mesmo nome.

Nota: por uma questão de clareza na escrita do código, a prática de nomear variáveis globais e locais com o mesmo nome não é recomendada.

**Supondo que exista uma variável local e uma global com o mesmo nome, qual prevalece?**

Prevalece sempre a variável local.

**//função entrada de dados**

**//usa as variáveis globais nota1 e nota2**

```
void entrada(void)
```

```
{
```

```
    printf("\nDigite a primeira nota: ");
```

```
    scanf("%f", &nota1);
```

```
    printf("Digite a segunda nota: ");
```

```
    scanf("%f", &nota2);
```

```
    return;
```

```
}
```



# FUNÇÕES RECURSIVAS

Sim é possível, uma função pode invocar a si mesma!

Em uma função recursiva, a cada chamada é criada na memória uma nova ocorrência da função com comandos e variáveis “isolados” das ocorrências anteriores.

A função é executada até que todas as ocorrências tenham sido resolvidas.

- **Vantagens da recursividade**

- Torna a escrita do código mais simples e elegante, tornando-o fácil de entender e de manter.

- **Desvantagens da recursividade**

- Quando o loop recursivo é muito grande é consumida muita memória nas chamadas a diversos níveis de recursão, pois cada chamada recursiva aloca memória para os parâmetros, variáveis locais e de controle.
- Em muitos casos uma solução iterativa gasta menos memória, e torna-se mais eficiente em termos de performance do que usar recursão.



```
//Cálculo de fatorial com função recursiva
#include <stdio.h>
#include <conio.h>
//protótipo da função fatorial
double fatorial(int n);
int main(void)
{
    int numero;
    double f;

    printf("Digite o numero que deseja calcular o fatorial: ");
    scanf("%d",&numero);
    //chamada da função fatorial
    f = fatorial(numero);

    printf("Fatorial de %d = %.0lf",numero,f);
    getch();
    return 0;
}
//Função recursiva que calcula o fatorial de um numero inteiro n
double fatorial(int n)
{
    double vfat;
    if ( n <= 1 )
        //Caso base: fatorial de n <= 1 retorna 1
        return (1);
    else
    {
        //Chamada recursiva
        vfat = n * fatorial(n - 1);
        return (vfat);
    }
}
```

**Note** que a definição implica em particular que  $0!=1$

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

equivale a:

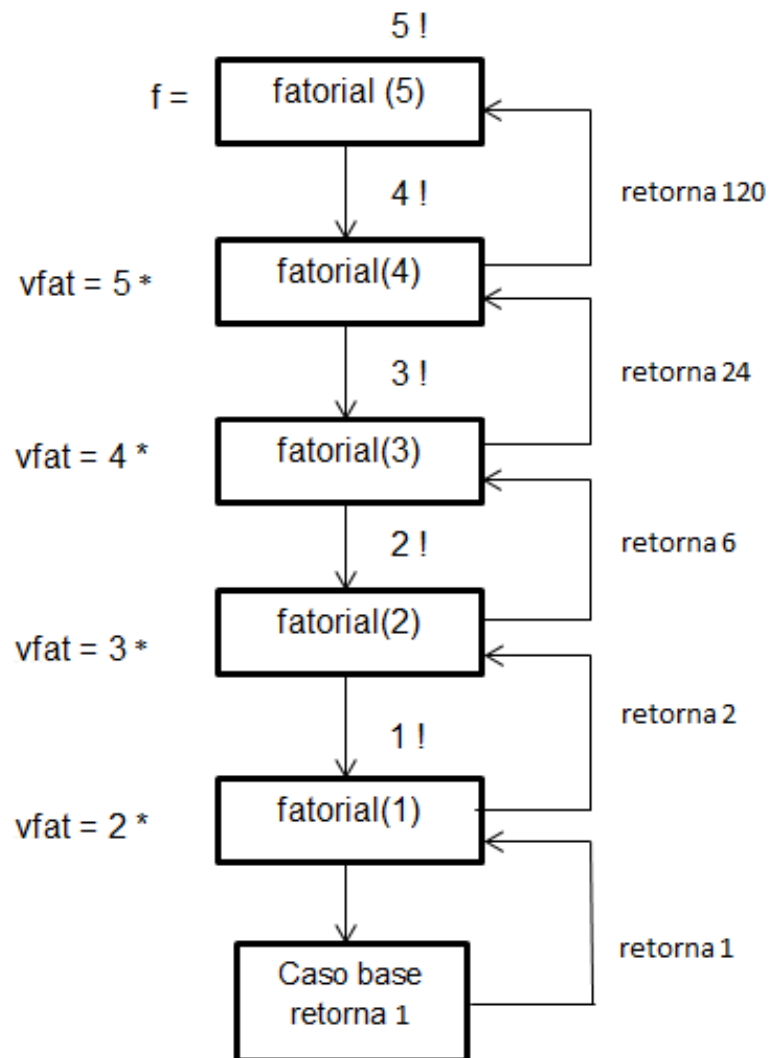
$$5! = 5 \times 4!$$

$$\text{e } 4! = 4 \times 3!$$

E assim sucessivamente...



# FATORIAL – FUNÇÃO RECURSIVA



```
double fatorial(int n)
{
    double vfat;

    if ( n <= 1 )
        //Caso base: fatorial de n <= 1 retorna 1
        return (1);
    else
    {
        //Chamada recursiva
        vfat = n * fatorial(n - 1);
        return (vfat);
    }
}
```



## EXERCÍCIOS



Desenvolva um programa com as funções que considerar apropriadas que permita:

1. Receber dois números e devolva o menor
2. Calcular o resultado de uma potência. Recebendo o valor da base e do expoente. Por exemplo, se receber os valores  $4^3$  devolve 64 ( $4 \times 4 \times 4$ )
3. uma função que recebe um valor inteiro e verifica se o valor é par. A função deve retornar 1 se o número for par e 0 se for ímpar
4. uma função que recebe um valor inteiro e verifica se o valor é positivo, negativo ou zero. A função deve retornar 1 para valores positivos, -1 para negativos e 0 para o valor 0.



## EXERCÍCIOS



5. ler 3 notas de um aluno e a média das notas dos exercícios realizados por ele. Calcular a média de aproveitamento, usando a fórmula:  $MA = (N1 + N2*2 + N3*3 + ME)/7$ . A partir da média, informar o conceito de acordo com a tabela:
- maior ou igual a 9 A
  - maior ou igual a 7.5 e menor que 9 B
  - maior ou igual a 6 e menor que 7.5 C
  - maior ou igual a 4 e menor que 6 D
  - menor que 4 E
6. apresentar a seguinte saída, perguntando ao usuário o número máximo (no exemplo, 9). Este número deve ser sempre **ímpar**.

```
1 2 3 4 5 6 7 8 9
  2 3 4 5 6 7 8
    3 4 5 6 7
      4 5 6
        5
```





## ○ Sessão 11



# MATRIZES

- Matriz é a uma estrutura de dados do tipo vetor com duas ou mais dimensões.
- Os itens de uma matriz tem que ser todos do mesmo tipo de dado.
- Na prática, as matrizes formam tabelas na memória.



# MATRIZES

- As matrizes são compostas por linhas e colunas, ou seja vetores dentro de vetores.

	0	1	2	3
0	2	4	6	8
1	3	6	9	12
2	4	8	12	16
3	5	10	15	20



# MATRIZES

- Identificar posição numa matriz

Matriz	coluna[0]	coluna[1]	coluna[2]	coluna[3]
linha[0]	2	4	6	8
linha[1]	3	6	9	12
linha[2]	4	8	12	16

- linha[0] coluna[0]  $\Rightarrow$  2
- linha [2] coluna [3]  $\Rightarrow$  16
- Também podemos representar a matriz da seguinte forma:

$\{\{2,4,6,8\},\{3,6,9,12\},\{4,8,12,16\}\}$



# MATRIZES

- Declarar e inicializar uma matriz

Nº de linhas

Nº de colunas

Elementos da  
matriz

```
int matriz [3][4];
```

```
int matriz {{2,4,6,8},{3,6,9,12},{4,8,12,16}};
```

Preencher a matriz com '1'

```
for (i = 0; i <3; i++)
```

```
    for(j = 0; j <4; j++)
```

```
        matriz[i][j] =1;
```

	0	1	2	3
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1



# MATRIZES

- Imprimir conteúdo de uma matriz

```
#include <stdio.h>
#define MAXLIN 3
#define MAXCOL 4

void main(){

int m_idades [MAXLIN][MAXCOL]={{12,48,2,41},{32,14,19,64},{85,14,26,24}};
int i,j;

/* Imprimindo os atribuídos para a matriz */
for (i = 0; i <MAXLIN; i++)

    for(j = 0; j <MAXCOL; j++)

        printf ("idade[%d][%d] = %d\n", i,j, m_idades[i][j]);

}
```



# MATRIZES

- Atribuição de valor da matriz através de SCANF
- **Exemplo 1:**

```
printf("Qual a idade do 1º aluno da turma B? ");  
scanf("%d",&m_idades[1][0]);
```

- **Exemplo 2:**

```
for(i=0 ; i<3 ; i++){  
    printf("\n%dº turma",i+1);  
    for(j=0 ; j<4 ; j++){  
        printf("\nIdade do %dº aluno: ",j+1);  
        scanf("%d",&m_idades[i][j]);  
    }  
}
```



## EXERCÍCIOS



1. Desenvolva um programa que leia 12 valores e guarde-os em uma matriz composta por 4 linhas e 3 colunas. Deve também mostrar na tela o primeiro valor da primeira linha e o segundo valor da segunda linha.
2. Crie um programa que leia 6 números e os armazene em uma matriz composta por 2 linhas e 3 colunas. Deve também calcular e devolver a soma de cada linha da matriz. E ainda a calcular e mostrar a soma de todos os valores da matriz.
3. Crie um programa que leia 16 valores e os armazene numa matriz quadrada (4x4). Deve no final mostrar os valores da diagonal, ou seja, o primeiro valor da primeira linha, o segundo valor da segunda linha... Até o quarto valor da quarta linha.





## WEBGRAFIA

- [https://www.slideshare.net/Rafael\\_Lima87/introduo-linguagem-c](https://www.slideshare.net/Rafael_Lima87/introduo-linguagem-c)

