FUNÇÕES

Em C existem apenas funções, noutras linguagens existem 2 tipos de rotinas/subprogramas - as funções e os procedimentos (os procedimentos não devolvem qualquer valor).

Razões que levam à escrita de funções:

- Reduzir a complexidade dos programas;
- Evitar a repetição de código ao longo de um programa;
- Facilidade na leitura e na alteração de código.

Caraterísticas duma função:

- Tem que ter um nome único, pelo qual será invocada/chamada algures no programa a que pertence;
- Não pode ser definida dentro de outras funções, mas pode ser invocada a partir de delas;
- Deve realizar uma única tarefa bem definida;
- O seu código deve ser o mais independente possível do resto do programa e deve ser tão genérico quanto possível, para poder ser reutilizado;
- Pode receber parâmetros que alteram o seu comportamento de forma a adaptar-se facilmente a situações distintas;
- Pode devolver, para a entidade que a invocou, um valor resultante do código que a constitui.

Cuidados a ter na escolha do nome duma função:

- Ter em conta as regras para nomear variáveis;
- Ser único;
- Especificar aquilo que a função na realidade faz e deve ser de fácil leitura e interpretação.

Funcionamento de uma função:

- O seu código só é executado quando esta é invocada, algures, no programa a que está ligada;
- Sempre que uma função é invocada:
  - o programa que a invoca é "suspenso" temporariamente;
  - são invocadas as instruções presentes no corpo da função;
  - terminada a função, o controlo da execução do programa volta ao local em que esta foi invocada.

Formato genérico de uma função:

```
tipo nome_função (tipo1 parâmetro1, … tipoN parâmetroN)
{
        Corpo da função
}
```

- O corpo de uma função é constituído por instruções de C, de acordo com a sintaxe da linguagem. Deve estar imediatamente a seguir ao cabeçalho da função e é escrito entre chavetas { };
- Sempre que não é colocado o tipo de retorno é assumido o tipo *int* ;
- O cabeçalho de uma função **nunca** deve ser seguido de (;) ;
- A instrução ***return*** permite terminar a execução de uma função e voltar ao programa que a invocou (a utilização da instrução *return* na função *main* faz com que o programa termine).

A Invocação (chamada) de funções pode ser feita de três formas:

- **Numa atribuição**

  Exemplo: X = dobro(X);

- **Dentro de uma função**, em que o valor de retorno é aproveitado com parâmetro para outra função

  Exemplo1: printf("%d %d", dobro(5), soma(dobro(2), 3));

  Exemplo2: if(soma(x,y)>0) …

- **Tal como se invoca um procedimento** (uma função de tipo void), perdendo-se neste caso, o valor de retorno

  Exemplo: getchar();

ONDE COLOCAR AS FUNÇÕES:

Em C, as funções podem ser colocadas em qualquer local do programa, antes ou depois de serem invocadas, antes ou depois da função *main.* **Mas quando não devolvem qualquer valor, ou quando o tipo é diferente de inteiro, há restrições.** Nestes casos, se as funções não estiverem definidas antes de serem utilizadas é necessário declará-las (a ver mais adiante).

COMO USAR UMA FUNÇÃO

Supõe que queríamos escrever um programa que coloque no ecrã o output do quadro abaixo, escrevendo a linha de 25 asteriscos com um ciclo for:

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

main()
{  int i;
    for (i=1; i<=20; i++)
        putchar('*');
    putchar('\n');
    puts("Números de 1 a 3");
       for (i=1; i<=20; i++)
        putchar('*');
    putchar('\n');
    for (i=1; i<=3; i++)
        printf("%d\n",i);
    for (i=1; i<=20; i++)
        putchar('*');
    putchar('\n');
}
```

```
********************
Números de 1 a 3
********************
1
2
3
********************
```

Se usarmos uma função para escrever os 25 asteriscos teremos o programa seguinte:

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

/*função*/
linha()
{
    int i;        /*variável local*/
    for (i=1; i<=20; i++)
        putchar('*');
    putchar('\n');
}
```

```
/*programa principal*/
 main()
{
   int i;          /*variável global*/
   linha() ;      /*chamada da função*/
    puts("Números de 1 a 3");
    linha();
    for (i=1; i<=3; i++)
        printf("%d\n",i);
    linha();
}
```

VARIÁVEIS LOCAIS: São variáveis que só têm efeito dentro das funções onde são declaradas. Após a execução da função, todas as variáveis locais são destruídas.

VARIÁVEIS GLOBAIS: São variáveis que mantêm o seu valor durante toda a execução do programa.

PARÂMETRO

- Qualquer tipo de dados da linguagem C pode ser enviado como parâmetro para uma função;
- Um parâmetro funciona como uma variável e serve para fazer passar valores para dentro e para fora das funções;
- Os parâmetros são separados por vírgulas (,) e é absolutamente necessário que, para cada um deles, seja indicado o seu tipo;
- Um parâmetro é automaticamente inicializado com o valor enviado pelo programa que invoca a função;
- O nº e tipo dos argumentos enviados devem coincidir com os parâmetros presentes no cabeçalho da função;
- Se a função receber mais do que 1 parâmetro, os argumentos enviados são associados aos parâmetros da função pela ordem em que são escritos, por exemplo:

```
#include<stdio.h>
funcao (int x, float y, char a)   /* parâmetros*/
{
        ...
}
main()
{
funcao (12, 3.1418, 'A') ;     /* argumentos*/
 int a=1; float b=3.0; char c='k';
funcao (a, b, c) ;      /* argumentos*/
...
```

- É comum chamar parâmetro, tanto aos argumentos de invocação de uma função, como aos verdadeiros parâmetros da função.
- Qualquer expressão válida em C pode ser enviada como argumento para uma função (exemplo: sqrt(4*3)).
- O nome das variáveis presentes no cabeçalho de uma função é totalmente independente do nome das variáveis que lhe vão ser enviadas pelo programa que a invoca (significa que **podemos dar o mesmo nome aos parâmetros e às variáveis colocadas como argumentos da função**).

EXERCÍCIOS:

1. Escreve um programa que, recorrendo a 2 funções distintas, escreva no ecrã o seguinte output:

   ***

   *****

   *****

   ***

   NOTA: Para a realização do programa precisamos de 3 funções:

   linha3x – função responsável por escrever 3 asteriscos no ecrã.

   linha5x – função responsável por escrever 5 asteriscos no ecrã.

   main – função que invoca as funções.

2. Modifica o programa anterior, recorrendo a um ciclo, para escrever no ecrã o mesmo output 5 vezes.

3. Altera o programa anterior de forma que a função linha escreva qualquer caráter e não apenas o asterisco. Deve ser lido no programa principal qual o caráter e o n.º de vezes que será impresso.

   NOTA: Utiliza o seguinte código como cabeçalho da função linha:

   linha (int num, char ch) – função responsável por escrever (num) carateres (ch) no ecrã.

Notas: A função **system ("pause")** permite que o programa aguarde até ser premida a tecla enter; a função **system("cls")** permite "limpar" o ecrã; a função **exit(0)** permite terminar o programa, independentemente de estar escrita numa função ou na função principal main.

DESAFIO (FACULTATIVO):

Implementa um programa que apresente um menu com as opções 3-Linha3x, 5-Linha5x e 7-Linha7x, caso seja inserido o valor 0, deverá terminar o programa. Se for introduzido um valor diferente dos anteriores, deverá ser exibida a mensagem- "Valor inválido!"
O programa deverá chamar as respetivas funções para imprimir no ecrã 3, 5 ou 7 asteriscos.

NOTA: Para a realização do programa deves utilizar 4 funções:

menu – função responsável por escrever o resultado a obter no programa.
linha3x – função responsável por escrever 3 asteriscos no ecrã.
linha5x – função responsável por escrever 5 asteriscos no ecrã.
Linha7x – função responsável por escrever 5 asteriscos no ecrã.

main – função que invoca a função menu (deve ser apresentado o menu usando o ciclo do while com condição sempre verdadeira).

FUNÇÕES E PROCEDIMENTOS:

A palavra reservada ***void*** permite indicar que uma função não devolve qualquer tipo.

Exemplo:

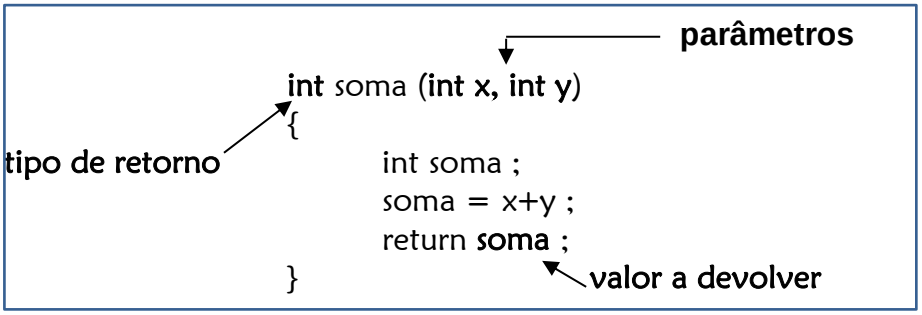
```
void linha (int num, char ch)
{
   int i;
   for (i=1; i<=num; i++)
         putchar(ch);
   putchar('\n');
}
```

- É habitual também encontrar a palavra reservada *void* para indicar que uma função não recebe qualquer parâmetro.
- Uma função que retorna void chama-se, normalmente, de **procedimento.**
- Sempre que se quer sair dum procedimento podemos usar a instrução ***return*** sem qualquer valor à frente.

FUNÇÕES QUE DEVOLVEM UM VALOR:

O resultado de uma função pode ser armazenado ou aproveitado por qualquer instrução.
A devolução de um resultado é feita através da instrução ***return*** seguida do valor a devolver.
A seguir à instrução *return* pode ser colocada qualquer expressão válida em C.

Exemplo 1:



EXERCÍCIOS:

4. Implemente um programa que faça uma calculadora com as 4 operações mais básicas (+, -, *, /) utilizando para tal 4 funções (soma, subtracao, produto, divisao), sendo os operandos fornecidos como parâmetros. A operação a efetuar e os operandos são introduzidos pelo utilizador.

5. Escreve um programa que solicite ao utilizador 2 n.ºs inteiros e apresente no ecrã a soma desses nºs, utilizando para o efeito a função acima.

6. Escreve um programa que solicite ao utilizador 2 nºs inteiros e apresente no ecrã o dobro da soma desses nºs, utilizando para o efeito a função soma dentro duma função que calcule o dobro.

7. Escreve uma função que calcule e devolva qual o maior de dois números inteiros, sendo esses nºs passados como parâmetros. Chama a função de MAX. Escreve um programa onde uses a função.

PROTÓTIPOS DE FUNÇÕES:

Em C, como já foi referido, se as funções não estiverem definidas antes de serem utilizadas é necessário declará-las.

Esta declaração consiste na escrita do seu cabeçalho seguida de um ponto e vírgula (**protótipo**), tradicionalmente logo a seguir aos *#include*.
Desta forma o compilador pode verificar, em cada invocação da função, se esta foi ou não corretamente implementada.

O **protótipo de uma função** corresponde então ao seu cabeçalho seguido de **;**. Este identifica a estrutura da função (nome, parâmetros e tipo de retorno).

Notas: É uma boa prática de programação colocar as funções definidas pelo programador, imediatamente antes do código da função principal, embora baste colocar o protótipo apenas antes da sua invocação.
O compilador apenas necessita saber qual o tipo de retorno da função. Assim, os seguintes protótipos têm a mesma funcionalidade:

```
int max(int n1, int n2);
int max(int , int );
int max();
```

Exemplo:

```
#include<stdio.h>
void fr (int n, char ch);
int max (int n1, int n2);

main()
{
  ....

}
void fr (int n, char ch)
{
  …
}
int max (int n1, int n2)
{
   …
}
```

Notas:

a) Como a linguagem C **não possui**, na sua estrutura, **um formato** de entrada e saída de dados **do tipo bool**, usamos para isso o formato de dados int- **%d**.

b) Sempre que tenhamos de devolver um resultado lógico em funções, também podemos usar o tipo int, pois, em C, falso é representado pelo valor 0 e verdadeiro por qualquer valor inteiro diferente de 0.

Exemplo:

```
bool isequal( int x, int y)
{
        if (x==y)
                return 1;
        else
                return 0;
}
```

O que equivale a …

…porque devolver 1, -1 ou 400 representa sempre verdade

```
bool isequal( int x, int y)
{
        return (x==y);
}
```

Podíamos, no cabeçalho da função ter escrito: int bool isequal(int x, int y) que seria igual (nota b).

EXERCÍCIOS:

8. Implementa as seguintes funções:

   a) **int abs(int x)** que devolve o valor absoluto de x, i.é., abs(-5) é 5 e abs(5) é 5.

   b) **bool impar(int x)** que devolve verdade se x for ímpar. Falso caso contrário.

   c) **int entre (int x, int lim_inf, int lim_sup)** que verifica se x se encontra no intervalo [lim_inf, lim_sup].

   d) **int is_vogal (char ch)** que verifica se ch é uma vogal do alfabeto. Escreve a função de maneiras diferentes (usando os carateres da tabela ascii ou o seu valor inteiro correspondente).

   Nota: Recorda o significado de casting e usa-lo neste exercício

DESAFIO (FACULTATIVO):

E se fosse pedido, na alínea anterior, que a função verificasse se ch era uma letra do alfabeto?

9. Dadas as funções **tiro** e **liro**

```
void tiro (int x)
 {
   switch (x)
   {
       case 1: printf("\nEste exercício é mesmo fácil");
      case 2: printf("\nAi de quem diga o contrário…");
                  return;
      case 3: printf("\nEstou no Tiro e x= %d", x);
                  liro (x++);
                  break;
      default: printf("\nEntrei pelo default");
                  liro(x);
    }
}
```

```
void liro (int x)
 {
   switch (x)
   {
      case 2: return;
                  return;
      case 3: printf("\nEu percebo bué de C");
                  break;
                  printf("\nSou mesmo Bom!");
      case 4: printf("\nNão percebo nada disto");
                  liro(2);
                  return;
       default: printf("\nCá estou eu mais uma vez");
                  tiro(x--);

    }
}
```

10. Qual o output das seguintes chamadas.

a) tiro (1);

b) tiro (3);

c) liro (2);

d) liro (4);

e) liro (5).

11. Indica se são verdadeiras ou falsas as seguintes afirmações:

a) Uma função em C pode devolver simultaneamente mais do que um valor;

b) Uma função em C pode ter parâmetros;

c) Uma função em C tem que devolver sempre um inteiro;

d) Os parâmetros em C podem ser do tipo *void* ;

e) A instrução *return* termina a execução duma função;

f) Uma variável local pode ter o mesmo nome que um parâmetro;

g) O nome de uma função não pode ser uma palavra reservada do C;

h) Sempre que for necessário devem-se utilizar variáveis locais;

i) A instrução return termina a execução de uma função apenas se for a última instrução da função em que se encontra;

j) Uma função deve fazer o maior nº de tarefas possível sem ocupar muito código;

k) Um protótipo não é mais que a repetição do cabeçalho de uma função;

l) Em C, um procedimento não é mais do que uma função que retorna *void.*

12. Verifica os erros de compilação.

f)
```
void f (int x, int y)
{
        return -1 ;
}
```

g)
```
void f (void) ;
void f (int x, int y)
{
        x=4 ; y=5 ;
}
```

h)
```
f (int x, int y) ;
{
        x=4 ; y=5 ;
}
```

13. Implementa a função **float pot (float x, int n)**

Que devolve o valor de x^n .

Sendo $x^0 = 1.0$; $x^n = x*x*\ldots*x$ (n vezes)