Linguagem C (parte 2)

- Definição de função
- Assinatura de função
- Ficheiros cabeçalho ou headers
- Escopo de variáveis
- Passagens por valor ou referência
- Vetores como argumento

Definição de função

- Funções são as estruturas que permitem separar os programas em blocos. Se não as tivéssemos, os programas teriam que ser curtos e de pequena complexidade. Para fazermos programas grandes e complexos temos de construí-los bloco a bloco.

```
tipo_de_retorno nome_da_função (declaração_de_parâmetros)
{
    corpo_da_função
}
```

Definição de função

- O comando ***return***

  return valor_de_retorno; ou return;

```
#include <stdio.h>
int EPar (int a)
{
        if (a%2)           /* Verifica se a e divisivel por dois */
          return 0;        /* Retorna 0 se nao for divisivel */
        else
          return 1;        /* Retorna 1 se for divisivel */
}
int main ()
{
        int num;
        printf ("Entre com numero: ");
        scanf ("%d",&num);
        if (EPar(num))
                printf ("\n\nO numero e par.\n");
        else
                printf ("\n\nO numero e impar.\n");
        return 0;
}
```

Exercício 6

- Escreva a função *EDivisivel(int a, int b)* - tome como base *EPar(int a)*. A função deverá retornar 1 se o resto da divisão de a por b for zero. Caso contrário, a função deverá retornar zero.

Assinatura de uma função

- Assinatura de função (protótipo de função) é assim representada:

  tipo_de_retorno nome_da_função (declaração_de_parâmetros);

```
#include <stdio.h>
float Square (float a);
int main ()
{
        float num;
        printf ("Entre com um numero: ");
        scanf ("%f",&num);
        num=Square(num);
        printf ("\n\nO seu quadrado vale: %f\n",num);
        return 0;
}
float Square (float a)
{
        return (a*a);
}
```

O tipo *void*

- O tipo ***void*** é utilizado da seguinte forma:

  void nome_da_função (declaração_de_parâmetros);

```
#include <stdio.h>
void Mensagem (void);
int main ()
{
        Mensagem();
        printf ("\tDiga de novo:\n");
        Mensagem();
        return 0;
}
void Mensagem (void)
{
        printf ("Ola! Eu estou vivo.\n");
}
```

  - ***NOTA:***
    - *A função main() é uma função e como tal devemos tratá-la com tal. O compilador acha que a função main() deve retornar um inteiro. Isto pode ser interessante se quisermos que o sistema operativo receba um valor de retorno da função main().*
    - *Devemos respeitar a convenção: se o programa retornar zero, significa que ele terminou normalmente, e, se o programa retornar um valor diferente de zero, significa que o programa teve um término anormal.*

Ficheiros cabeçalho ou *headers*

- Ficheiros cabeçalho são aqueles que mandamos o compilador incluir no início dos exemplos:
  - stdio.h, conio.h, string.h.
  - Estes arquivos não possuem os códigos completos das funções. Eles só contêm as assinaturas de funções.
  - O compilador lê estes protótipos e, baseado nas informações lá contidas, gera o código correto.
  - O corpo das funções cujas assinaturas estão nos ficheiros cabeçalho, são incluídas no programa quando se faz a "linkagem".
    - Todas as referências a funções cujos códigos não estão nos arquivos fontes são resolvidas

Ficheiros cabeçalho ou *headers*

- Suponha que a função 'int EPar(int a)', é importante em vários programas,
  - No arquivo de cabeçalho chamado por exemplo de 'funcao.h' teremos a seguinte declaração:

```
int EPar(int a);
```

  - O código da função será escrito num arquivo a parte. Vamos chamá-lo de 'funcao.c'. Neste arquivo teremos a definição da função:

```
int EPar (int a)
{
if (a%2)                /* Verifica se a e divisivel por dois */
        return 0;
else
        return 1;
}
```

Ficheiros cabeçalho ou headers

- Finalmente, teremos o programa principal que vamos designar por *princip.c*:

```
#include <stdio.h>
#include "funcao.h"
void main ()
{
        int num;
        printf ("Entre com numero: ");
        scanf ("%d",&num);
        if (EPar(num))
                printf ("\n\nO numero e par.\n");
        else
                printf ("\n\nO numero e impar.\n");
}
```

  - Este programa terá de ser compilado utilizando o comando:

```
gcc princip.c funcao.c -o saida
```

Exercício 7

- Escreva um programa que faça uso da função *EDivisivel(int a, int b)* apresentada anteriormente.
  - Organize o seu programa em três arquivos:
    - o arquivo prog.c , conterá o programa principal;
    - o arquivo func.c conterá a função;
    - o arquivo func.h conterá o protótipo da função.
  - Compile os arquivos e gere o executável a partir deles.

Escopo das variáveis

- **Definição**
  - O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.
- **Variáveis locais**

```
func1 (...)
{
        int abc,x;
        ...
}
func (...)
{
        int abc;
        ...
}
void main ()
{
        int a,x,y;
        for (...)
        {
                float a,b,c;
        ...
        }
        ...
}
```

Escopo das variáveis

- P**arâmetro (formal)**
  - São declarados como sendo as entradas de uma função.
    - O parâmetro é uma variável local da função.
    - A alteração do valor de um parâmetro formal, não terá efeito na variável que foi passada à função

Escopo das variáveis

- **Variáveis globais**
  - Variáveis globais são declaradas fora de todas as funções do programa.
  - Podem ser alteradas por todas as funções do programa.
  - Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.

```
int z,k;
func1 (...)
{
        int x,y;
        ...
}
func2 (...)
{
        int x,y,z;
        ...
        z=10;
        ...
}
main ()
{
        int count;
...
}
```

Exercício 7

- Estude o seguinte programa e aponte o valor de cada variável sempre que solicitado:

```
#include <stdio.h>
int num;
int func(int a, int b)
{
        a = (a+b)/2;  /* Qual e o valor de a apos a atribuicao? */
        num -= a;
        return a;
}
main()
{
        int first = 0, sec = 50;
        num = 10;
        num += func(first, sec);  /* Qual e o valor de num, first e sec */
                                  /* antes e depois da atribuicao?      */
        printf("\n\nConfira! num = %d\tfirst = %d\tsec = %d",num, first, sec);
}
```

Passagem por valor ou referência

- **Passagem por valor**
  - Os parâmetros formais da função copiam os valores dos parâmetros que são passados para a função.
  - Não são alterados os valores que os parâmetros têm fora da função.

```
include <stdio.h>
float sqr (float num);
void main ()
{
        float num,sq;
        printf ("Entre com um numero: ");
        scanf ("%f",&num);
        sq=sqr(num);
        printf ("\n\nO numero original e: %f\n",num);
        printf ("O seu quadrado vale: %f\n",sq);
}
float sqr (float num)
{
        num=num*num;
        return num;
}
```

Passagem por valor ou referência

- **Passagem por referência**
  - Quando queremos alterar as variáveis que são passadas para uma função, nós podemos declarar seus parâmetros formais como sendo apontadores
    - Os apontadores são a "referência" que precisamos para poder alterar a variável fora da função.
    - Teremos de lembrar de colocar um **&** na frente das variáveis que estivermos passando para a função.

```
#include <stdio.h>
void Swap (int *a,int *b);
void main (void)
{
        int num1,num2;
        num1=100;
        num2=200;
        Swap (&num1,&num2);
        printf ("\n\nEles agora valem %d  %d\n",num1,num2);
}
void Swap (int *a,int *b)
{
        int temp;
        temp=*a;
        *a=*b;
        *b=temp;
}
```

Exercício 8

- Escreva uma função que receba duas variáveis inteiras e coloque o seu valor a zero.

Vetores como argumentos

- Quando se passa um vetor como argumento de uma função, pode declarar-se a função de três maneiras equivalentes.
  - Dado o vetor:

    int matrx [50];
  - *Podemos declarar a função:*

    void func (int matrx[50]);

    void func (int matrx[]);

    *void func (int *matrx);*

Exercício 9

- Escreva um programa que leia um vetor de inteiros pelo teclado e o apresente no ecrã.

  Crie uma função *void levetor(int *vet, int dimensao)* para fazer a leitura desse vetor.