A Função

- Funções são as estruturas que permitem ao usuário separar seus programas em blocos. Uma função no C tem a seguinte forma geral:

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

- 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.

A Função

- O tipo-de-retorno é o tipo de variável que a função vai retornar. O default é o tipo **int**, ou seja, uma função para qual não declaramos o tipo de retorno é considerada como retornando um inteiro.
- A declaração de parâmetros é uma lista com a seguinte forma geral:

    tipo nome1, tipo nome2, ... , tipo nomeN

- O corpo da função é a sua alma. É nele que as entradas são processadas, saídas são geradas ou outras coisas são feitas.

O Comando return

- O comando **return** tem a seguinte forma geral:

  return *valor_de_retorno*; ou ***return***;

- Digamos que uma função está sendo executada. Quando se chega a uma declaração **return** a função é encerrada imediatamente;

O Comando return

- É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.
- Uma função pode ter mais de uma declaração **return**. A função é terminada quando o programa chega à primeira declaração **return**.

O Comando return

Exemplo 1

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

Exemplo 2

```
#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 */
}
void main ()
{
        int num;
        printf ("Entre com numero: ");
        scanf ("%d", &num);
        if (EPar(num))
              printf ("\n\n O numero e par.\n");
        else
              printf ("\n\n O numero e impar.\n");
}
```

O Comando return

- É importante notar que, como as funções retornam valores, podemos aproveitá-los para fazer atribuições, ou mesmo para que estes valores participem de expressões.
- Mas não podemos fazer:

```
func(a,b) = x;    /* Errado! */
```

Protótipos de Funções

- Até agora, nos exemplos apresentados, escrevemos as funções antes de escrevermos a função main(). Isto é, as funções estão fisicamente antes da função main().
- Isto foi feito por uma razão. Imagine-se na pele do compilador. Se você fosse compilar a função main(), onde são chamadas as funções, você teria que saber com antecedência quais são os tipos de retorno e quais são os parâmetros das funções para que você pudesse gerar o código corretamente.
- Quando o compilador chegasse à função main() ele já teria compilado as funções e já saberia seus formatos.

Protótipos de Funções

- Mas, muitas vezes teremos o nosso programa espalhado por vários arquivos. Ou seja, estaremos chamando funções em um arquivo que serão compiladas em outro arquivo. Como manter a coerência? A solução são os protótipos de funções.
- Um protótipo tem o seguinte formato:

tipo_de_retorno nome_da_função (declaração_de_parâmetros);

Protótipos de Funções

Exemplo

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

Escopo de Variáveis

- O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.
  - Variáveis Locais
  - Variáveis Globais
  - Parâmetros formais

Variáveis locais

- Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas.
- Um bloco começa quando abrimos uma chave e termina quando fechamos a chave.

Variáveis locais

Exemplos de variáveis locais

```
func1 (...)
{
        int abc, x;
        ...
}
func (...)
{
        int abc;
        ...
}
```

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

Parâmetros formais

- Parâmetros formais são declarados como sendo as entradas de uma função.
- O parâmetro formal é uma variável local da função.
- Você pode também alterar o valor de um parâmetro formal, pois esta alteração não terá efeito na variável que foi passada à função. Isto tem sentido, pois quando o C passa parâmetros para uma função, são passadas apenas cópias das variáveis.

Variáveis globais

- Variáveis globais são declaradas fora de todas as funções do programa. Elas são conhecidas e 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.

Variáveis globais

```
int z, k;

func1 (...)
{
        int x, y;
        ...
}

func2 (...)
{
        int x, y, z;
        ...
        z =10;
        ...
}
```

```
main ()
{
        int count;
   ...
}
```

Variáveis globais

- Evite ao máximo o uso de variáveis globais. Elas ocupam memória o tempo todo (as locais só ocupam memória enquanto estão sendo usadas) e tornam o programa mais difícil de ser entendido e menos geral.

Passagem de parâmetros por valor

```
#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\n O numero original e: %f\n", num);
        printf ("O seu quadrado vale: %f\n", sq);
}
float sqr (float num)
{
        num = num*num;
        return num;
}
```

Passagem de parâmetros por referência

- Este tipo de chamada, não se passa para a função os valores das variáveis, mas sim suas referências (endereço das variáveis);
- Neste caso, a alteração dos parâmetros da função, alterará as variáveis que são passadas para a função também.

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

Vetores como Argumentos de Funções

- Quando vamos passar um vetor como argumento de uma função, podemos declarar a função de três maneiras equivalentes. Seja o vetor:

```
int matrx [50];
```

e que queiramos passá-la como argumento de uma função func(). Podemos declarar func() das três maneiras seguintes:

```
void func (int matrx[50]);
void func (int matrx[]);
void func (int *matrx);
```