Funções

1. Conceito

Uma função é uma unidade de código de programa (bloco) autônoma e escrita para executar uma tarefa específica. O "C" foi projetado para usar pequenos programas, ou seja, funções.

Vantagens de uso das funções (entre outras):

- permite que se utilize código já existente;
- evita que o programador escreva o mesmo código, repetidas vezes no programa;
- permite escrever os programas em blocos dividindo a complexidade do problema.

2. Forma Geral

tipo de retorno nome_ da_ função (lista de parâmetros)
{
corpo da função
}

tipo de retorno: É o tipo da variável que a função vai retornar. O default é o inteiro (int). Uma função para a qual não declaramos o tipo de retorno é considerada como retornando um inteiro.
nome da função : É um identificador válido dentro da linguagem.
lista de parâmetros: É nos parâmetros, que informamos ao compilador quais serão as entradas da função. A declaração de parâmetros é uma lista com a seguinte forma: *(tipo nome1, tipo nome2, tipo nome3....).*
Exemplos:

```
//exemplo 1
#include <stdio.h>
mensagem( )
{
   printf("Esta é uma mensagem");
}
main( )
{
  int i;
  for (i=1; i<=4; i++)
        {
          mensagem( );
          printf("\n");
        }
 getch();
}
```

3. Chamada de funções

Chamamos a função da mesma maneira que chamamos as funções da biblioteca “C” (como mensagem( )). Os parênteses são necessários para que se possa diferenciar uma chamada de uma função de uma variável não declarada.

Como a chamada de uma função é uma instrução de programa, ela deve ser encerrada por ponto e virgula.

Obs: na definição da função o (;) não é utilizado.

```
//exemplo 2
#include <stdio.h>
int square (int a) /* ou square (int a) */
{
  int quad;
  quad = a * a;
  printf("\n\n O seu quadrado é: %d \n",quad);
}
main( )
{
  int num;
  printf("Entre com numero");
  scanf("%d",&num);
  square(num);
  getch();
}
```

4. Variáveis locais

As variáveis declaradas dentro do bloco de uma função ({ }) são chamadas de locais e conhecidas somente dentro do seu bloco. Elas existem apenas durante a execução do bloco onde foram declaradas. São criadas quando o bloco entra em execução e destruídas ao seu término. A declaração de variáveis locais deve ser a primeira coisa que devemos colocar no bloco. Podemos ter quantos blocos quisermos com variáveis de mesmo nome e não haverá conflito entre elas.

A palavra reservada ***auto*** serve para dizer que uma variável é local, entretanto é desnecessário utilizá-la, pois variáveis declaradas dentro do bloco já são consideradas locais.

5. Parâmetros formais

Estes são declaradas como sendo as entradas de uma função. O parâmetro é como se fosse uma variável local da função.

6. Variáveis globais

São as declaradas fora de todas as funções do programa. Elas são conhecidas e podem ser alteradas por todas as funções. Quando uma função possui uma variável local, com o mesmo nome de uma global, a função sempre irá se referir à variável local.

```
//exemplo 3
#include <stdio.h>
int y,z,k; // globais
func1 (..)
{
  .....
 }
func2 (....)
{
  int x,y,z;
   z=10;
 }
 main ( )
 {
   .................
 }
```

7. Protótipos de funções

Até agora escrevemos as funções antes de escrevermos a função main. Isto foi feito de forma a permitir ao compilador saber com antecedência quais são os tipos de retorno e quais os parâmetros a serem utilizados.

Com o protótipo de funções, que surgiu com a declaração expandida do padrão ANSI, podemos escrever a função após a declaração *main( ).*

```
//exemplo 4
# include <stdio.h>
float square (float a); // protótipo de função
main ( )
{
  float num;
  puts("Entre com um numero");
  scanf ("%f", &num);
  num = square (num);
  printf ("\n\n O seu quadrado e: %3.2f",num);
  getch();
 }
float square (float a)
{
    return( a * a);
}
```

8. Usando mais de uma função

Pode-se usar qualquer número de funções num programa e qualquer função pode chamar outra. Entretanto, não é permitido definir uma função dentro de outra.

9. Função do tipo void

Void quer dizer vazio. Void nos permite declarar funções que não retornam nada. O comitê de padronização da linguagem "C" (ANSI) batizou funções que não retornam nada como tendo um novo tipo: *void*

void nome_da_função (parâmetros)

Podemos tambem ter funções sem parâmetros: t*ipo de retorno nome-da-função (void)*

Ou ainda, funções que não retornam nada nem tem parâmetros:

void nome-da-função (void)

```
//exemplo 5
#include <stdio.h>
void mensagem (void);
void main (void);
{
  mensagem( );
  getch();
 }
void mensagem(void)
{
  printf("\n Ola !");
}
```

10. Chamada por valor

Quando chamamos uma função seus parâmetros formais recebem cópias dos valores dos argumentos que lhe são passados. Isto quer dizer, que não são alterados os valores dos argumentos fora da função. Este tipo de chamada de função e conhecido como chamada por valor.

```
//exemplo 6
#include <stdio.h>
void main (void)
 {
  float num,sq;
   puts("Entre com um numero");
   scanf("%f",&num);
   sq=sqt(num);
   printf ("\n\n O numero original e %f",num);
   printf ("\n\n O seu quadrado e %f",sq);
   getch();
  }
 float sqt (float num)
```

```
{
  num=num*num;
  return (num);
}
```

11. Passando vários parâmetros

Podem ser passados para a função quantos argumentos forem necessários, separados por virgulas.

```
//exemplo 7
// imprimir a área de retângulos recebendo dois argumentos
#include <stdio.h>
retang (int largura, int altura);
main ( )
{
  printf ("\n Sala \n");
  retang (22,12);
  printf ("\n Cozinha\n");
  retang (16,16);
  printf("\n Banheiro\n");
  retang (6,8);
  printf("\n Quarto\n");
  retang (12,10);
 getch();
}

 retang (int largura, int altura)
  {
    int area;
    area=largura*altura;
    printf ('%d",area);
}
```

12. Chamada por referência

Outro tipo de passagem de parâmetros para uma função ocorre quando alterações nos parâmetros dentro da função alteram os valores dos argumentos passados. Este tipo de chamada tem o nome de chamada por referência (passam-se não os valores mas sim o endereço de referência).

O “C” não faz chamadas por referência, só por valor. O “C++” possui um recurso que permite chamadas por referência. Entretanto existe um recurso de programação em “C” que podemos usar para simular uma chamada por referência:

- Declarar os parâmetros como sendo ponteiros;
- Colocar **&** na frente das variáveis (argumentos) que estiverem sendo passadas para a função.

```
//exemplo 8
#include <stdio.h>
void main (void)
{
 int num1=100,num2=200;
swap (&num1,&num2);
printf("\n Eles agora valem %d %d \n", num1,num2);
getch()
}
void swap (int *a, int *b)
{
  int temp;
  temp=*a;
  a*=b*;
  *b=temp;
}
```

13. O comando *return*

O comando *return* encerra imediatamente a execução de uma função e se um valor de retorno é informado (entre parênteses), retorna este valor. O comando *return* tem dois usos importantes: o primeiro para devolver um valor e retornar para a próxima instrução do código de chamada, o segundo para causar uma saída imediata da função (sem parênteses).

A função não necessita do comando *return*, neste caso, ela se encerrará após a execução da ultima instrução.

```
//exemplo 9
#include <stdio.h>
epar (int a)
{
  if (a%2 = =0) /* Verifica se é divisivel por 2 */
       return (1);
  else
       return (0);
 }

main ( )
{
  int num;
  puts("Entre com um número");
  scanf ("%d", &num);
  if (epar (num))
       printf( "\n\n O número %d e par ",num);
  else
       printf("\n\n O número %d e impar",num);
  getch();
 }
```