FUNÇÕES

Uma função nada mais é do que um bloco de código (ou seja, declarações e outros comandos) que pode ser nomeado e chamado de dentro de um programa.
Em outras palavras, uma função é uma sequência de comandos que recebe um nome e pode ser chamada de qualquer parte do programa, quantas vezes forem necessárias, durante a sua execução.

A linguagem C possui muitas funções já implementadas, e as temos utilizado constantemente. Um exemplo são as funções básicas de entrada e saída: **scanf()** e **printf()**. O programador não precisa saber o código contido dentro das funções de entrada e saída para utilizá-las. Basta saber seu nome e como utilizá-la.

Assim, a finalidade deste material é apresentar os conceitos e detalhes necessários para um programador criar suas próprias funções dentro da linguagem C. Ao final, deve-se ter a capacidade de:

- Declarar uma nova função;
- Definir os parâmetros de uma função;
- Definir o retorno de uma função;
- Fazer a passagem de parâmetros por valor para a função;
- Fazer a passagem de parâmetros por referência para a função.

Modularização

A ideia principal do conceito de modularização é dividir o programa em subprogramas, o que torna o trabalho de desenvolvimento e manutenção menos desgastante. Em C o conceito de modularização é implementado por meio de funções. As funções podem ter variáveis próprias ou utilizar as variáveis declaradas como globais. É importante lembrar que variáveis locais com o mesmo identificador (nome) de variáveis globais ocultam o acesso às variáveis globais.

Por que modularizar?

- Para permitir o reaproveitamento de código já construído (próprio ou de 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, mais difíceis de entender (garantir legibilidade);
- Para facilitar a leitura e depuração do programa fonte de uma forma mais fácil;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

Funções em C

Uma função é uma unidade de código de programa autônoma desenhada para cumprir uma tarefa particular. A função em C, pode receber vários parâmetros, mas retorna apenas um valor. Da mesma forma que declaramos variáveis, devemos declarar a função. A declaração de uma função é chamada de **protótipo** e é uma instrução geralmente colocada no início do programa, que estabelece o tipo da função e os argumentos que ela recebe. Tecnicamente, quando o *tipo_da_funcao* é **void,** esta função pode ser chamada de um PROCEDIMENTO (embora, a maior parte da literatura não use esta nomenclatura para a linguagem C).

Declaração das Funções

tipo nome_função (declaracao dos argumentos);
→ declaração do protótipo da função deve preceder sua definição e chamada.

void main(void)
{
a=nome_função (lista dos argumentos a serem enviados);
}
→ chama a função **nome_função**, a qual retorna um valor em **a;**

tipo nome_função(lista dos parâmetros recebidos)
{
declaração das variáveis locais comandos;
comandos;
return(valor); → retorno de um valor
}

* Expressões em negrito correspondem a comandos opcionais.

Diferentemente do que acontece na declaração de variáveis, na qual muitas podem ser declaradas com o mesmo especificador de tipo, na declaração de parâmetros de uma função é necessário especificar o tipo de cada variável.

```
01  //Declaração CORRETA de parâmetros
02  int soma(int x, int y){
03        return x + y;
04  }
05
06  //Declaração ERRADA de parâmetros
07  int soma(int x, y){
08        return x + y;
09  }
```

Tipos de Funções

O tipo de uma função é determinado pelo tipo de valor que ela retorna pelo comando **return** e não pelo tipo de seus argumentos. Se uma função for do tipo não inteira ela deve ser declarada. O valor default é **int** caso não for declarada.
Alguns tipos:

- **float** - retorna um valor numérico real ;
- **int** - retorna valor inteiro (não é necessária a sua declaração);
- **void** - funções que não retornam valores.

Exemplo:

```
 /* Calcula a área de uma esfera */

#define PI 3.14159

 float area(int r);                          /* protótipo da função */

 void main(void)
{
    int raio;
    float area_esfera;
    printf (“Digite o raio da esfera: ”);
    scanf(“%d”, &raio);
    area_esfera = area(raio);                          /* chamada à função */
    printf(“A área da esfera é %f”, area_esfera);
 }

 float area(int r)                /* definicao da funcao */
{
    return(4*PI*r*r);           /* retorna float */
}
```

Estrutura de um programa modularizado

Bibliotecas

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

Constantes

```
#define MaxValor  100
#define TF  40
```

Struct’s

```
struct TpPessoa
{
  int matricula;
  float peso, altura;
  char Nome[20];
};
```

Declaração dos protótipos das funções

```
void LeVetor(TpPessoa VetPessoa[TF], int &TL);
void ExibeVetor(TpPessoa VetPessoa[TF], int TL);
void Mensagem(void);
char Menu(void);
```

Implementação das funções

```
void Mensagem(void)
{
  printf(“\n Pressione algo para continuar!”);
```

```
  getch();
}

char Menu(void)
{
  printf("## Menu de Opções ##");
  printf("\n[A] Digite dois números");
  printf("\n[B] Somar números");
  printf("\n[C] Multiplicar números");
  printf("\n[D] Exibir resultado");
  printf("\n[ESC] Finalizar");

  return (toupper(getche());
}
```

Função Principal

```
int main(void)
{
  char op;
  do
  {
     op = Menu();
   switch(op)
   {
     case 'A':  ...;
                break;
     case 'B':  ...;
                break;
     case 'C':  ...;
                break;
     case 'D':  ...;
                break;

   }
  } while (op!=27);

  return 0;
}
```

Passagem de Parâmetros

..:: por Valor: também conhecido como **parâmetro de entrada**, deve ser utilizado quando não se deseja a atualização da variável, ou seja, haverá apenas o uso do valor contido na mesma. O compilador faz uma 'cópia' da variável e a utiliza.

Exemplo:

```
#include <stdio.h>

int verifica(int num)
{
  int res;
  if (num >= 0)
     res = 1;
  else
     res = 0;
  return res;
}

int main(void)
{
  int num, x;
  printf("\nDigite um número: ");
  scanf("%d ",&num);
  x = verifica(num);
  if (x==1)
     printf("\nNúmero positivo");
  else
     printf("\nNúmero negativo");
  getchar();
  return 0;
}
```

..:: por Referência: também conhecido como **parâmetro de entrada e saída**, deve ser utilizado quando se deseja a atualização da variável recebida , isso quer dizer que o conteúdo da variável sofrerá alteração, pois o acesso é realizado diretamente no endereço em que se encontra a variável enviada.

Exemplo:

```
void LeNumero(int &num)
{
   printf ("Digite o número: ");
   scanf("%d", &num);
}

void Soma(int num1, int num2, int &resultado)
{
  resultado = num1 + num2;
}
```

```
void Exibe(int N)
{
   printf ("\nResultado: %d", N);
   Mensagem();
}

int main(void)
{
  int A, B, RES;
  LeNumero(A);
  LeNumero(B);
  Soma(A,B,RES);
  Exibe(RES);
}
```

Exemplos de Funções

Exemplo: cálculo do fatorial dentro da função main()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int main(){
05    printf("Digite um numero inteiro positivo:");
06    int x;
07    scanf("%d",&x);
08    int i,f = 1;
09    for (i=1; i<=x; i++)
10        f = f * i;
11
12    printf("O fatorial de %d eh: %d\",x,f);
13    system("pause");
14    return 0;
15  }
```

Exemplo: cálculo do fatorial em uma função própria

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int fatorial (int n){
05    int i,f = 1;
06    for (i=1; i<=n; i++)
07        f = f * i;
08
09    return f;
10  }
11
12  int main(){
13    printf("Digite um numero inteiro positivo:");
14    int x;
15    scanf("%d",&x);
16    int fat = fatorial(x);
17    printf("O fatorial de %d eh: %d\n",x,fat);
18
19    system("pause");
20    return 0;
21  }
```

Exemplo: função contendo operações de leitura e escrita

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int menu(){
04    int i;
05    do {
06        printf("Escolha uma opção:\n");
07        printf("(1) Opcao 1\n");
08        printf("(2) Opcao 2\n");
09        printf("(3) Opcao 3\n");
10        scanf("%d", &i);
11    } while ((i < 1) || (i > 3));
12
13    return i;
14  }
15
16  int main(){
17    int op = menu();
18    printf("Vc escolheu a Opcao %d.\n",op);
19    system("pause");
20    return 0;
21  }
```

Exemplo: função com retorno

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int soma(int x, int y){
04    return x + y;
05  }
06
07  int main(){
08    int a,b,c;
09    printf("Digite a: ");
10    scanf("%d", &a);
11    printf("Digite b: ");
12    scanf("%d", &b);
13    printf("Soma = %d\n",soma(a,b));
14    system("pause");
15    return 0;
16  }
```

Função que não recebe e não retorna valor

```
void Mensagem(void)
{
  printf("\n Pressione algo para continuar!");
  getch();
}
```

Função que não recebe, mas retorna valor

```
char Menu(void)
{
  printf("## Menu de Opções ##");
```

```
  printf("\n[A] Digite dois números");
  printf("\n[B] Somar números");
  printf("\n[C] Multiplicar números");
  printf("\n[D] Exibir resultado");
  printf("\n[ESC] Finalizar");

  return (toupper(getche());
}
```

Função que recebe e retorna valor

```
#include <stdio.h>

int Verifica(int num)
{
 int res;

 if (num >= 0)
   res = 1;
 else
   res = 0;

return res;
}

int ConheceNum(void)
{
  int num;
  printf ("Digite o número: ");
  scanf("%d", &num);
  return num;
}


int main(void)
{
 int x;

 x = Verifica(ConheceNum());

 if (x==1)
   printf("Número positivo!");
 else
   printf("Número negativo!");

 getchar();
 return 0;
}
```

Função que recebe e retorna valor por parâmetro

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

struct ponto
{
  int x, y;
};

void atribui(ponto &p)
{
  p.x = 10;
  p.y = 20;
}

int main()
{
  ponto p1;
  atribui(p1);
  printf("x = %d\n",p1.x);
  printf("y = %d\n",p1.y);
  system("pause");
  return 0;
}
```