Funções em C

Funções

- O C é baseado em funções, sendo a função `main` hierarquicamente a mais importante das funções. Dentre as vantagens das funções destacamos:
  - Permitir ao programador modularizar o programa;
  - Reutilizar o código
  - Evitar a repetição de código em um programa;
- Pode-se invocar (chamar) funções contidas nas diversas bibliotecas (ou arquivos de cabeçalho) ou o programador pode criar suas próprias funções e, até mesmo, criar seu próprio arquivo de biblioteca de funções.
- Para utilizar uma função de uma biblioteca, é necessário incluir o nome da biblioteca na seção `#include`

Funções

- Abaixo algumas funções úteis nas bibliotecas mais utilizadas

math.h		string.h	
pow (x, y)	x^y	strlen(x)	Retorna tamanho da string
sqrt (x)	Raiz Quadrada	strcat (x,y)	Concatena strings
ceil (x)	Arredondamento p/cima	strcpy (x,y)	Copia string y para x.
floor (x)	Arredondamento p/ baixo	strcmp (x,y)	Compara strings
log (x)	Logaritmo na base 2	strncat(x,y,n)	Concatena no máximo n caracteres
log10 (x)	Logaritmo na base 10	strncpy(x,y,n)	Copia no máximo n caracteres
sin(x)	seno	strncmp(x,y,n)	Compara no máximo n caracteres
		strupr(x)	Caixa alta

Funções

Utilizando função(s) de uma biblioteca(s)

```c
#include <math.h> //incluindo a biblioteca no programa
main(){
        printf("%5.0f\n",pow(2,3));/*exibindo o resultado
        de 2 elevado a 3 utilizando a função pow*/
        system("pause");
}
```

```c
#include <string.h> //incluindo a biblioteca no programa
main(){
        char x[8]={"Uenf"},y[8]={"Uerj"};
        if (strcmp(x,y)==0)  /*compara as duas strings e
        retorna 0 (iguais) -1 (diferentes)*/
           printf("Iguais");
        else
           printf("Diferentes");
        system("pause");
}
```

Funções

- A forma geral de uma função é:

```
Especificador_de_tipo nome_da_função(lista de parâmetros)
{
  Declarações e comandos
}
```

- Onde:
  - *Especificador_de_tipo*: especifica o tipo de valor (int, char, float, double) que o comando **return** devolve;
  - *Nome_da_função*: é o nome que será utilizado para invocar a função;
  - *Lista de parâmetros*: lista de variáveis e respectivos tipos, separadas por uma vírgula, que recebem os valores dos **argumentos** quando a função é chamada.

Funções

Escrevendo suas próprias funções

- Pode-se declarar novas variáveis dentro da função, sendo que nestes casos será uma variável que só poderá ser manipulada dentro da função, ou seja, será uma **variável local**.
- Mesmo que os parâmetros da lista sejam do mesmo tipo, não se pode informar o tipo uma única vez e declarar as variáveis. Ex.:

```
float soma(float x,y,z)
```
//errado

```
float soma(float x, float y, float z)
```
//certo

- Os parâmetros da função na sua declaração são chamados **parâmetros formais**. Na chamada da função os parâmetros são chamados **parâmetros atuais**.

Funções

Escrevendo suas próprias funções

- Sobre as variáveis definidas dentro da função podemos afirmar que:
  - As variáveis valem no bloco que são definidas;
  - as variáveis definidas dentro de uma função recebem o nome de **variáveis locais**;
  - os parâmetros formais de uma função valem também somente dentro da função;
  - uma variável definida dentro de uma função não é acessível em outras funções, MESMO ESTAS VARIÁVEIS TENHAM NOME IDÊNTICOS.

- Caso a função não retorne valor (**void**) não se deve retornar valor (não usar **return**).
- Usar **return** nos casos em que a função deva retornar algum tipo.

Funções

`Escrevendo suas próprias funções`

- O tipo de uma função é determinado pelo valor que ela retorna via comando **return**, e não pelo tipo de argumentos que ela recebe.
- Podemos escrever funções no corpo do programa de duas formas:
  - Prototipando a função (protótipo externo ou interno);
  - Escrevendo o código da função e após o fim do bloco iniciar a bloco **`main`**. Neste caso não é necessário prototipar a(s) função(s).

Funções

- Prototipando a função (externamente)

```
0  int soma(int a,int b);
1  main(){
2        int n1,n2,res;
3        scanf("%d",&n1);
4        scanf("%d",&n2);
5        res=soma(n1,n2); /*a variavel 'res' recebe o valor calculado na função
6        'soma. A função recebe como argumento os valores de 'n1' e 'n2'.*/
7        printf("\n%d\n",res);
8        system("pause");
9        }
10 int soma(int a,int b)
11 {
12       return(a+b);
13 }
```

Funções

- Não prototipando a função

```
0  int soma(int a,int b)
1  {
2       return(a+b);
3  }
4  main(){
5       int n1,n2,res;
6       scanf("%d",&n1);
7       scanf("%d",&n2);
8       res=soma(n1,n2); /*a variavel 'res' recebe o valor calculado na função
9       'soma. A função recebe como argumento os valores de 'n1' e 'n2'.*/
10      printf("\n%d\n",res);
11      system("pause");
12      }
```

Funções

`Retornando valores`

- Necessidade de indicar o tipo de dados que a função irá retornar;
- Utilizar o **`return()`**, onde o dentro dos parênteses pode-se ter:
  - Um valor;
  - Uma variável;
  - Uma fórmula

Funções

Retornando valores

- Exemplo de retorno de um `int`

```
1  #define n 6
2  int qtdpar(int v[]);
3  main(){
4          int vet[n]={34,21,45,6,98,12};
5          printf("%d\n",qtdpar(vet));
6          system("pause");
7          }
8  int qtdpar(int v[]){
9      int i,cont=0;
10     for (i=0;i<6;i++)
11         if ((v[i]%2)==0)
12             cont++;
13     return(cont);
14     }
```

Funções

Retornando valores

- Exemplo de retorno de um `string`

```
0  char *comparacao(char [],char []);
1  main(){
2          char str1[10],str2[10];
3          gets(str1);
4          gets(str2);
5          printf("\n%s\n",comparacao(str1,str2));
6          system("pause");
7          }
8  char *comparacao(char x[10],char y[10]){
9          char res[20];
10         if (strcmp(x,y)==0)
11            return("Iguais");
12         else
13            return("Diferentes");
14 }
```

Funções

Retornando valores

- Exemplo de retorno de um `vetor de double`

```
0  #include <stdio.h>
1  double *saida();
2  int main()
3  {
4      double *m;
5      m = saida();
6      printf("%lf M[0]\n", m[0]);
7      printf("%lf M[1]\n", m[1]);
8      system("pause");
9  }
10 double *saida()
11 {
12     double *u;
13     u=(double*) malloc(2 * sizeof(double));
14     u[0] = 10;
15     u[1] = 20;
16     return u;
17 }
```

Funções

Parâmetros de uma função

As variáveis que receberão as informações enviadas a uma função são chamadas **parâmetros**. A função deve declarar essas variáveis entre parênteses, no cabeçalho de sua definição ou antes das chaves que marcam o início do corpo da função. Os parâmetros podem ser utilizados livremente no

```
1 float celsius(float fahr){
2       return (fahr-32)*5/9;
3       }
```

o do tipo *float*

Funções

Parâmetros de uma função

//Exemplos de declaração de parâmetros do tipo *string*

```c
int tamanho(char txt[]){
    int i=0;
    while (txt[i]!='\0')
          i++;
    return (i-1);
}
main(){
       char nome[30]="Programação C";
       printf("%d\n",tamanho(nome));
       system("pause");
       }
```

```c
#define m 4
#define n 4
int qtdpar(int mat[][n]){
    int l,c,cont=0;
    for (l=0;l<m;l++)
        for (c=0;c<n;c++)
            if ((mat[l][c]%2)==0)
               cont++;
    return(cont-1);
}
main(){
       int num[m][n],i,j;
       for (i=0;i<m;i++)
           for (j=0;j<n;j++)
               num[i][j]=rand()%100;
       printf("%d\n",qtdpar(num));
       system("pause");
       }
```

Funções

Parâmetros de uma função

//Exemplos de um programa que retorna a quantidade de vezes que determinado caracter aparece no vetor;

```
1  #include<stdio.h>
2  #include<conio.h>
3  #define DIM 80
4  char conta (char v[], char c);
5  void main(){
6        char linha[DIM];
7        char c;
8        int maiusculas[26], minusculas[26];
9        puts("Entre com uma linha");
10       gets (linha);
11       for (c='a'; c<='z'; c++)
12           minusculas[c-'a'] = conta(linha, c);
13       for (c='A'; c<='Z'; c++)
14           maiusculas[c-'A'] = conta(linha, c);
15       for (c='a'; c<='z'; c++)
16           if (minusculas[c-'a'])
17              printf("%c apareceu %d vezes\n", c, minusculas[c-'a']);
18       for (c='A'; c<='Z'; c++)
19           if (maiusculas[c-'A'])
20              printf("%c apareceu %d vezes\n", c, maiusculas[c-'A']);
21      getch();
22 }
23 char conta (char v[DIM], char c){
24       int i=0, vezes=0;
25       while (v[i] != '\0')
26             if (v[i++] == c) vezes++;
27       return vezes;
28 }
```

Funções

Pode-se passar argumentos para as funções de duas maneiras:

- **Chamada por valor**
  - Copia o valor de um argumento no parâmetro da função. Alterações feitas nos parâmetros da função não tem nenhum efeito nas variáveis usadas pra chamá-la.
- **Chamada por referência**
  - O endereço de um argumento é copiado no parâmetro. As alterações feitas no parâmetro afetam a variável usada para a chamar a rotina. Utiliza-se os conceitos de ponteiros.
- Tais argumentos podem ser: int, float, double, char, ponteiros...

Funções

- Exemplo de chamada por valor

```
0  troca(int a, int b);
1  main(){
2          int x=2,y=3;
3          troca(x,y);
4          printf("x=%d y=%d\n",x,y);
5  }
6  troca (int a, int b){
7         int temp;
8         temp=a;
9         a=b;
10        b=temp;
11 }
```

- Qual é o valor de x e y na 2ª linha e depois ao término da 4a linha?

Funções

- Exemplo de chamada por referência

```
0  void troca(int *a, int *b);
1  main(){
2         int x=10,y=20;
3         troca(&x,&y);//passa os endereços de 'x' e 'y'
4         printf("x=%d y=%d\n",x,y);
5         system("pause");
6         }
7  void troca(int *a,int *b){
8        int temp;
9        temp=*a; //'temp' recebe conteudo de apontado por a
10       *a=*b; //poe 'b' em 'a'
11       *b=temp; //poe 'temp' em 'b'
12       }
```

- Qual é o valor de x e y na 2ª linha e depois ao término da 4ª linha?

Funções

Escrevendo seu arquivo de funções

- os arquivos de funções, também conhecidos por cabeçalhos (*headers*), são arquivos contendo uma coleção de funções desenvolvidas pelo programador;
- Estes arquivos (com extensão .h) ficam na pasta `include` do seu compilador C;
- Para incluí-los em seu programa, é necessário utilizar a diretiva **`#include`**. Na verdade o compilador substitui a diretiva **`#include`** de nosso programa pelo conteúdo do arquivo indicado. Exemplo:

```
#include <minha_biblioteca.h> /*com os sinas < e > o arquivo é procurado somente na pasta include*/
ou
#include “minha_biblioteca.h” /*com as aspas duplas o arquivo é procurado primeiramente na pasta atual e depois, se não for encontrado, na pasta include*/
```

Funções

Escrevendo seu arquivo de funções

- Exemplo de um programa que inclui a biblioteca criada, além de usar uma de suas funções.

```c
/*programa principal utilizando
função da biblioteca "opbasicas.h"*/
#include "opbasicas.h"
#include <stdio.h>
main(){
        float a;
        a=div(1,2);
        printf("%.2f\n",a);
        system("pause");
        }
```

```c
//arquivo opbasicas.h
float soma(float x, float y){
     return(x+y);
}
float sub(float x, float y){
    return(x-y);
}
float mult(float x, float y){
    return(x*y);
}
float div(float x, float y){
    return(x/y);
}
```

Funções

Recursividade

- São funções que podem chamar a si mesmas. Tal comando deve estar presente no corpo do função.
- São úteis para criar versões mais claras e simples de vários algoritmos. Sua utilização não caracteriza uma melhora de performance.
- **Deve ter uma condição de parada**. Deve ter um comando if em algum lugar para forçar a função a retornar sem que a chamada recursiva seja executada.

Funções

Recursividade

```
0  int fatorial(int num);
1  main(){
2          int fat=0;
3          fat=fatorial(4);
4          printf("%d\n",fat);
5          system("pause");
6  }
7  int fatorial(int num){
8      if (num==1)
9          return(1);
10     return (num*fatorial(num-1));
11 }
```

Funções

Recursividade

- Passos do algoritmo

1- Recebe valor 4.

2- Passo 1: fatorial := 4*fatorial(3);

3- Fatorial é chamada de dentro com o valor 3.

4- Passo 2: fatorial := 3*fatorial(2);

5- Passo 3: fatorial := 2*fatorial(1);

6- Se valor for 1, retorna 1. Fim de loop, mas não de contas.

7- Voltamos aos passos, ao contrário.

8- Passo 3: fatorial := 2*1;

9- Passo 2: fatorial := 3*2;

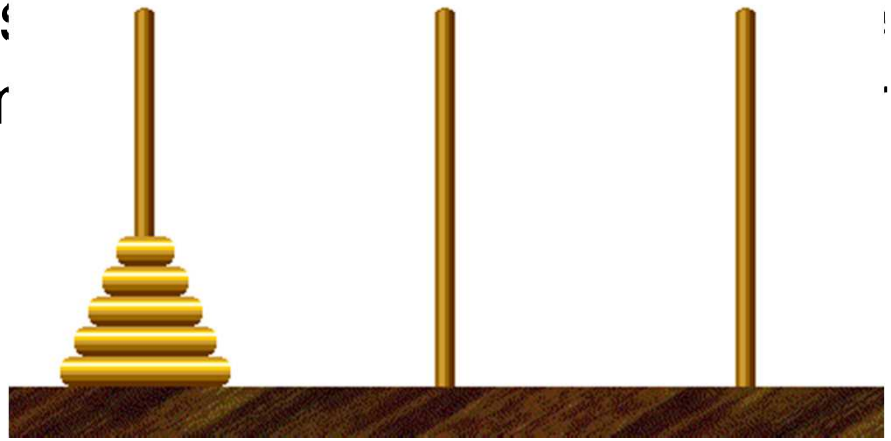
10- Passo 1: fatorial := 4*6;

11- A primeira retorna o valor 24.

Funções

- **`Recursividade`** – Exercício – **Torre de Hanói**

Neste jogo temos três hastes, que chamaremos de Origem, Destino e Temporária, e um número qualquer de discos de tamanhos diferentes posicionados inicialmente na haste da Origem. Os discos são dispostos em ordem de tamanho. O objetivo do jogo é movimentar um a um os discos da haste de origem para a haste Destino,utilizando a haste temporária como auxiliar. Nenhum di
um disco menor. Som
movimentado por vez.



Funções

- **`Recursividade`** – Exercício – **Torre de Hanói**

```
0  void mover(int, char, char, char);
1  main(){
2          mover(3,'O','D','T');
3          system("pause");
4          }
5  void mover(int n, char Orig, char Temp, char Dest){
6       if (n==1)
7          printf("Mova o disco 1 da haste %c para a haste %c\n",Orig,Dest);
8       else
9            {mover(n-1,Orig,Dest,Temp);
10           printf("Mova o disco %d da haste %c para a haste %c\n",n,Orig,Dest);
11           mover(n-1,Temp,Orig,Dest);}
12      }
```