

UNINOVE

Estrutura de Dados

Prof. Mailson

Email: mailson.silva@uninove.br



Conteúdo programático

- Revisão de Procedimentos Funções e Parâmetro;
- Algoritmos de ordenação:
 - ⇒ Trocas;
 - ⇒ Seleção
 - ⇒ Inserção;
- Algoritmos de pesquisa
 - ⇒ Sequencial;
 - ⇒ Binária;
- Fila;
- Pilha;
- Lista ligada;
- Outros;

▪ Material do AVA

Bibliografia

3

• BIBLIOGRAFIA BÁSICA:

SCHILDT, Hebert. C – Completo e total. Ed. Makron Books, 1995.

Wirth, Niklaus – Algoritmos e estrutura de dados, Ed. Rio de Janeiro LTC, 2001

PEREIRA, Silvio do Lago; Estruturas de Dados Fundamentais – Conceitos e Aplicações. 4ª Ed. São Paulo, Érica, 2000, 264 p.

• BIBLIOGRAFIA COMPLEMENTAR:

VILLAS, Marcos Viana. Estruturas de Dados: Conceitos e Técnicas de Implementação. Ed. Campus, 1993.

ZIVIANI, Nivio. Projeto de algoritmos com implementação em Pascal e C. Ed. Thomson, 2000.

ASCENCIO, Ana Fernanda Gomes. Lógica de programação com Pascal. Ed. Makron Books, 1999.

MIZRAHI, Victorine Viviane. Treinamento em Linguagem C – Módulo I. São Paulo: Ed. Makron Books, 1990.

MIZRAHI, Victorine Viviane. Treinamento em Linguagem C – Módulo II. São Paulo: Ed. Makron Books, 1990.

- 1º Semestre – Algoritmos e Lógica de Programação;
- 2º Semestre – Práticas de Programação;
- 3º Semestre – Estruturas de Dados;
- 4º Semestre – Pesquisa e Ordenação;

- Outras disciplinas relacionadas;
 - ⇒ Linguagem de programação procedural;
 - ⇒ Java;
 - ⇒ Programação para banco de dados;
 - ⇒ Etc.

Revisão de Procedimentos Funções e Parâmetro

- “Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado”;
- “Estes dizeres servem também para a construção de programas”;
- “Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema”;

■ Vantagens:

- ⇒ Subdivisão de algoritmos complexos, facilitando o seu entendimento;
- ⇒ Estruturação de algoritmos, facilitando, principalmente, a detecção de erros e a documentação de sistemas;
- ⇒ Modularização de sistemas que facilita a manutenção de software e a reutilização de subalgoritmos já implementados;
- ⇒ Garante maior legibilidade;

- Executa uma sequência de passos, mas não retorna nenhum valor.

```
#include <stdio.h>
#include <stdlib.h>
int n1,n2;
void mostra()
{
    if (n1==n2)
        printf("\nNumero iguais....\n");
    else if (n1>n2)
        printf("\nPrimeiro numero e o maior....\n\n");
    else
        printf("\nSegundo numero e o maior....\n\n");
}
void recebe()
{
    printf("Entre com o primeiro numero:... ");
    scanf("%d",&n1);
    printf("Entre com o segundo numero:... ");
    scanf("%d",&n2);
}
main()
{
    system("cls");
    printf("*****\n");
    printf("*           E X E M P L O   D E   P R O C E\n");
    printf("*****\n");
    recebe();
    mostra();
    system("pause");
}
```

- Executa uma sequência de passos, e sempre retorna um valor;

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

float a,b,c;

float prod ( ) {
    return a * b ;
}

int main ( ) {
    a = 2;
    b = 110;
    c = prod() ;
    printf ( "%f\n\n",c ) ;
    system ( "pause" ) ;
    return 0 ;
}
```

- Para que a função seja executada, primeiro ela precisa receber alguma(s) informação(ções), essa(s) informação(ões) é (são) chamada(s) de parâmetro;

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

float prod ( float x, float y ) {
    return x * y ;
}

int main ( ) {
    float a, b, c;
    a = 2;
    b = 110;
    c = prod ( a, b ) ;
    printf ( "%f\n\n",c ) ;
    system ( "pause" ) ;
    return 0 ;
}
```

• PARÂMETRO POR VALOR

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

int x;
void porvalor ( int a ) ;

int main ( ) {
    x = 10;
    porvalor ( x ) ;
    printf ( "x = %d\n\n", x ) ;
    system ( "pause" ) ;
    return 0 ;
}

void porvalor ( int a ) {
    a = 5 ;
    printf ( "x = %d      a = %d\n\n", x, a ) ;
}
```

• PARÂMETRO POR REFERÊNCIA

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

int x;
void por_ref ( int *a ) ;

int main ( ) {
    x = 10;
    por_ref ( &x ) ;
    printf ( "x = %d\n\n", x ) ;
    system ( "pause" ) ;
    return 0 ;
}

void por_ref ( int *a ) {
    *a = 5 ;
    printf ( "x = %d      a = %d\n\n", x, *a ) ;
}
```

Ordenação

- Por que ordenar?

⇒ “Ordenar os dados também pode ser uma etapa preliminar para procurá-los, que é muito mais rápido do que uma procura linear”;

Ordenação para dois valores

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

main()
{
    int a, b;
    printf("Entre com o valor de a: ");
    scanf("%d", &a);
    printf("Entre com o valor de b: ");
    scanf("%d", &b);

    if (a==b)
        printf("Valores iguais\n");
    else
        if (a < b)
            printf("%d, %d\n", a,b);
        else
            printf("%d, %d\n", b,a);

    system("pause");
}
```

- Desenvolva um programa que seja capaz de ordenar três valores inteiros quaisquer digitados.


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

main()
{
    int a, b, c;
    printf("Entre com o valor de a:");
    scanf("%d",&a);
    printf("Entre com o valor de b:");
    scanf("%d",&b);
    printf("Entre com o valor de c:");
    scanf("%d",&c);

    if((a==b) && (a==c))
        printf("Valores iguais\n");
    else
        if ((a <= b) && (a <= c))
        {
            printf("%d, ", a);
            if (b < c)
                printf("%d, %d\n ",b,c);
            else
                printf("%d, %d\n ",c,b);
        }
    else
        if ((b <= a) && (b <= c))
        {
            printf("%d, ", b);
            if (a < c)
                printf("%d, %d\n ",a,c);
            else
                printf("%d, %d\n ",c,a);
        }
    else
        {
            printf("%d, ", c);
            if (a < b)
                printf("%d, %d\n ",a,b);
            else
                printf("%d, %d\n ",b,a);
        }
    system("pause");
}
```

Ordenação de Vetor Bubble Sort

Ordenação por trocas (Bubble Sort)

- Também conhecido como "método da bolha" ("bubble sort") é um dos métodos mais simples de ordenação.
- Compara dois elementos consecutivos de um vetor e se o da esquerda é maior que o da direita trocam de posição. Quando existem trocas, os elementos maiores tendem a deslocar-se para a direita e os menores para a esquerda.
- Consiste em percorrer o vetor, comparando-se cada elemento do vetor com o elemento imediatamente seguinte ($V[\text{índice}]$ com $V[\text{índice} + 1]$).

| | | | | | | | | |
|---|---|---|---|----------|---|---|---|---|
| 6 | 2 | 1 | 3 | <u>4</u> | 5 | 8 | 7 | 0 |
|---|---|---|---|----------|---|---|---|---|

- <https://www.youtube.com/watch?v=IIX2SpDkQDc>
- <https://www.youtube.com/watch?v=lyZQPjUT5B4>

Ordenação por trocas (Bubble Sort)

```
void bsort(int vet[], int t)
{
    int i, j, k=0;

    for (i=0; i<t-1; i++)
    {
        for (j=0; j<t-(i+1); j++)
        {
            if (vet[j] > vet[j+1])
            {
                k=vet[j];
                vet[j]=vet[j+1];
                vet[j+1]=k;
            }
        }
    }
}
```

vet – recebe o vetor que será ordenado;

t – recebe a quantidade de elementos do vetor;

i – determina o número de etapas para ordenação;

j – determina o número de comparações em cada etapa e os índices a serem pesquisados para a comparação;

k – variável auxiliar para ajudar na troca de posição dos valores no vetor;

Ordenação por trocas (Bubble Sort)

```
void bsort(int vet[], int t)
{
    int i,j,k=0;

    for (i=0;i<t-1;i++)
    {
        for (j=0;j<t-(i+1);j++)
        {
            if (vet[j] > vet[j+1])
            {
                k=vet[j];
                vet[j]=vet[j+1];
                vet[j+1]=k;
            }
        }
    }
}
```

[illegible]