VETORES

1- Modularização - Ordenação e busca sequencial devetor de registros - Bolão da Copa

Seus amigos descobriram que você "mexe com informática" e pediram para você fazer um programa para ajudar a organizar o bolão da copa. Para isso, você verifica que precisará implementar um algoritmo para cadastrar, ordenar e buscar valores num vetor de registros. Cada registro conterá os seguintes dados: número do time, nome do time (sem espaços), quantidade de gols marcados. Como outro colega irá melhorar seu aplicativo, vocês decidem que o programa será construído utilizando módulos. Assim, o programa deve ter módulos para:

- ler valores, preenchendo o vetor de registros;
- ordenar os elementos de um vetor, utilizando o número do time como chave, utilizando um dos métodos vistos em sala de aula;
- procurar um elemento num vetor, usando busca sequencial, retornando sua posição no vetor.

Outros módulos são opcionais.

O módulo principal deve:

1. ler a quantidade de elementos a ser processada;
2. ativar o módulo que lê os elementos do vetor;
3. ler o valor procurado, no caso o número do time;
4. ativar o módulo que ordena o vetor pelo número do time;
5. ativar o módulo que busca um elemento num vetor, para procurar o elemento no vetor ordenado e depois escrever a posição do valor no vetor ordenado. Caso o elemento não esteja no vetor, a posição deve ser -1. Caso o elemento seja encontrado, imprimir ainda o nome do time e o número de gols marcados.

A ordem de entrada dos dados dos registros é a seguinte:

```
num_do_time   nome_do_time   gols_marcados
```

Entradas:

- Tamanho do vetor (inteiro).
- Elementos do registro (struct).
  - Número do time (inteiro).
  - Nome do time (string).
  - Gols marcados (inteiro)
- Valor a ser procurado (inteiro).

Saídas:

- Posição do time no vetor (inteiro).

- Nome do time (string)
- Gols do time (inteiro)

Exemplos de Entradas e Saídas:

Entradas:

3

1 mexico 4

3 alemanha 2

2 porto_rico 1

3

Saídas:

2

alemanha 2

2- SUGESTÃO DE SOLUÇÃO:

Ideia geral:

Um vetor é uma variável que consegue armazenar mais de um valor, ela é declarada como: tipoDaVariável `nomeDoVetor[QUANTIDADE_DE_ELEMENTOS]`, por exemplo, um vetor de inteiros de 5 posições seria declarado dessa forma: `int vetor[5];`

Uma struct é um estrutura de dados que consegue armazenar vários tipos de dados, sua declaração é feita por: `struct nomeDaEstrutura { tipoDasVariáveis variáveis; };` e no módulo que for usar essa estrutura: `nomeDaEstrutura nomeVariávelUsadaPraManipularAEstrutura;`

A ordenação de um vetor é feita usando alguma regra de ordenação, no caso desse problema usaremos uma ordenação em ordem crescente. Usaremos o algoritmo Merge Sort, para realizar essa tarefa.

A procura de dados será feita, através de um percorrimento das posições do vetor, comparando o identificador do time com o identificador fornecido pelo usuário.

Passo 1: Armazenamento dos dados de entrada: antes que o usuário possa inserir o número de elementos e os seus valores na entrada padrão, é necessário que haja um espaço de memória reservado para armazená-los. Deve-se primeiro declarar uma variável inteira para armazenar o número de elementos e fazer a leitura do valor inserido pelo usuário. Como o enunciado pede que os valores sejam armazenados em um vetor, deve-se declarar uma variável composta homogênea do tipo struct (um vetor da estrutura) com o número de elementos lido anteriormente para depois realizar a leitura de cada elemento e armazenar em cada posição do vetor. No caso dessa struct, teremos dois elementos do tipo inteiro(identificador e gols_marcados) e um elemento do tipo string(nome). É preciso utilizar

uma estrutura de repetição com uma variável contadora para controlar o número de iterações, já que o número total de elementos a serem lidos é conhecido. Para isso iremos fazer um módulo, em que sua função será a leitura dos elementos.

```
struct time {
   int identificador;
   string nome;
   int gols_marcados;
};
int tamanho;
cin >> tamanho;
time meuVetorDeTimes[tamanho];
lerVetorDeTimes( meuVetorDeTimes, tamanho );


void lerVetorDeTimes( time meuVetorDeTimes[ ], int tamanho ) {
   for ( int i = 0; i < tamanho; i++ ) {
                cin >> meuVetorDeTimes[i].identificador >> meuVetorDeTimes[i].nome >> meuVetorDeTimes[i].gols_marcados;
   }
}
```

Depois de lermos o vetor de times, nós leremos o identificador(tipo inteiro) fornecido pelo usuário:

```
int identificador;
cin >> identificador;
```

Passo 2: Ordenação dos elementos: Para realizarmos a ordenação dos elementos, usaremos o algoritmo Merge Sort, esse algoritmo divide o vetores em metades até ser somente dois elementos e assim ele vai ordenando recursivamente.

```
void intercala( time meuVetorDeTimes[ ], int inicio, int meio, int fim ) {
   int i = inicio, j = meio + 1;
```

```
    int tamanho = fim - inicio + 1;
    time aux[tamanho]; // vetor auxiliar
    for ( int k = 0; k < tamanho; k++ ) {
        if ( ( i <= meio ) and ( j <= fim ) ) {
            if ( meuVetorDeTimes[i].identificador <= meuVetorDeTimes[j].identificador ) {
                aux[k] = meuVetorDeTimes[i]; // copia trecho1 em aux[]
                i++; // avança em trecho1
            }
            else { //
                aux[k] = meuVetorDeTimes[j]; // copia trecho2 em aux[]
                j++; // avanca em trecho2
            }

        }
        else if ( i > meio ) { // terminou o trecho1
            aux[k] = meuVetorDeTimes[j];
            j++;
        }
        else { // terminou o trecho2
            aux[k] = meuVetorDeTimes[i];
            i++;
        }
    }
    // terminando: copiar de aux[] em a[inicio:fim]
    for ( int k = 0; k < tamanho; k++ ) {
        meuVetorDeTimes[inicio + k] = aux[k];
    }
}

void mergesort( time meuVetorDeTimes[ ], int inicio, int fim ) {
    int meio;
    if ( inicio < fim ) {
        meio = ( inicio + fim ) / 2;
        mergesort( meuVetorDeTimes, inicio, meio );
        mergesort( meuVetorDeTimes, meio + 1, fim );
        intercala( meuVetorDeTimes, inicio, meio, fim );
    }
}
```

Passo 3: Procura do elemento. Por fim, deve-se procurar o elemento com o mesmo identificador fornecido pelo usuário, caso esse identificador não exista no vetor deve-se retornar -1, caso exista deve-se retornar a posição do elemento no vetor. Para isso, iremos percorrer o vetor novamente e a cada iteração iremos comparar os valores do identificador fornecido com o identificador do time de tal posição, caso seja igual, retornaremos a posição do time no vetor, caso seja diferente, fazemos mais uma iteração, até atingir o limite do vetor. Caso atinja e não tenha encontrado o time, retornaremos -1.

```
int procuraTime( time meuVetorDeTimes[ ], int identificador, int tamanho ) {

    int timeInexistente = -1;

    for ( int posicao = 0; posicao < tamanho; posicao++ ) {

        if ( meuVetorDeTimes[posicao].identificador == identificador ) {
```

```
            return posicao;
        }
    }
    return timeInexistente;
}
```

Passo 4: Impressão da saída de dados: o enunciado pede que a posição retornada pela função seja impresso na tela. Assim, é impresso a posição, e caso ela seja diferente de -1, é impresso o nome do time e os gols marcados:

```
int posicao = procuraTime( meuVetorDeTimes, identificador, tamanho );
if ( posicao >= 0 ) {
        cout << meuVetorDeTimes[posicao].nome << " " << meuVetorDeTimes[posicao].gols_marcados << endl;
}
```

Exemplo de código:

```
#include <iostream>
using namespace std;


struct time {
    int identificador;
    string nome;
    int gols_marcados;
};


int procuraTime( time meuVetorDeTimes[ ], int identificador, int tamanho ) {
    int timeInexistente = -1;
    for ( int posicao = 0; posicao < tamanho; posicao++ ) {
        if ( meuVetorDeTimes[posicao].identificador == identificador ) {
            return posicao;
        }
    }
```

```
    return timeInexistente;
}

void intercala( time meuVetorDeTimes[ ], int inicio, int meio, int fim ) {
    int i = inicio, j = meio + 1;
    int tamanho = fim - inicio + 1;
    time aux[tamanho]; // vetor auxiliar
    for ( int k = 0; k < tamanho; k++ ) {
        if ( ( i <= meio ) and ( j <= fim ) ) {
            if ( meuVetorDeTimes[i].identificador <= meuVetorDeTimes[j].identificador ) {
                aux[k] = meuVetorDeTimes[i]; // copia trecho1 em aux[]
                i++; // avança em trecho1
            }
            else { //
                aux[k] = meuVetorDeTimes[j]; // copia trecho2 em aux[]
                j++; // avanca em trecho2
            }

        }
        else if ( i > meio ) { // terminou o trecho1
            aux[k] = meuVetorDeTimes[j];
            j++;
        }
        else { // terminou o trecho2
            aux[k] = meuVetorDeTimes[i];
            i++;
        }
    }
    // terminando: copiar de aux[] em a[inicio:fim]
    for ( int k = 0; k < tamanho; k++ ) {
        meuVetorDeTimes[inicio + k] = aux[k];
    }
}

void mergesort( time meuVetorDeTimes[ ], int inicio, int fim ) {
    int meio;
```

```
    if ( inicio < fim ) {
        meio = ( inicio + fim ) / 2;
        mergesort( meuVetorDeTimes, inicio, meio );
        mergesort( meuVetorDeTimes, meio + 1, fim );
        intercala( meuVetorDeTimes, inicio, meio, fim );
    }
}

void lerVetorDeTimes( time meuVetorDeTimes[ ], int tamanho ) {
    for ( int i = 0; i < tamanho; i++ ) {
                cin >> meuVetorDeTimes[i].identificador >> meuVetorDeTimes[i].nome >>
meuVetorDeTimes[i].gols_marcados;
    }
}

int main() {
    int tamanho;
    cin >> tamanho;
    time meuVetorDeTimes[tamanho];
    lerVetorDeTimes( meuVetorDeTimes, tamanho );
    int identificador;
    cin >> identificador;
    mergesort( meuVetorDeTimes, 0, tamanho - 1 );
    int posicao = procuraTime( meuVetorDeTimes, identificador, tamanho );
    cout << posicao << endl;
    if ( posicao >= 0 ) {
                                cout << meuVetorDeTimes[posicao].nome << " " <<
meuVetorDeTimes[posicao].gols_marcados << endl;
    }

    return 0;
}
```

Passo 5: Teste do programa:

TESTE DE MESA - Clique aqui!

MAIS MATERIAL DE APOIO:

GitHub

Replit