```
//utills.cpp
#include "utills.h"
//função para iniciar  listas com valores padrão
void createLista(Lista *lista)
{
   lista->start=NULL;
   lista->end=NULL;
   lista->tam=0;
}
//Tratar alguns erros de inputs de usuários
int handle_exception()
{
   int numero;
   while (true)
   {
      std::cout<<"Digite um inteiro válido:"<<std::endl;
      try
      {
         std::cin>>numero;
         return numero;
      }
      catch (const char* err)
      {
         std::cerr<<"Erro:"<<err<<std::endl;
      }
      catch (...)
      {
         std::cerr<<"Erro não tratado"<<std::endl;
      }

   }
}
//Função de adiciona no começo
void addStart(Lista *lista,int numero)
{
   //ponteiro auxilar do numero
   no *aux=(no*)malloc(sizeof(no));
   aux->dado=numero;
   //Verifica se o começo é nulo
   if (lista->start!=NULL)
   {
      //atribui o numero e muda o ponteiro.
      aux->prox=lista->start;
      lista->start=aux;
   }
   else
   {
      //como é nulo basta apontar para nulo e atribuir os valores da variavel auxiliar para a lista
      aux->prox=NULL;
      lista->start=aux;
      lista->end=aux;
   }
```

```
    //Importante aqui é para controlar o tamanho da lista
    lista->tam++;
}
//Função de adicionar no final, perceba as diferenças? Encontrou alguma?
void addEnd(Lista *lista,int numero)
{
    //aqui para quem usa c, sabe que precisa alocar
    no *aux=(no*)malloc(sizeof(no));
    aux->dado=numero;
    aux->prox=NULL;
    if (lista->start!=NULL)
    {
        lista->end->prox=aux;
        lista->end=aux;
    }
    else
    {
        lista->start=aux;
        lista->end=aux;
    }
    lista->tam++;
}
//Função para mostrar a lista
void showLista(Lista *lista)
{
    no *start=lista->start;
    for(int i=0;i<lista->tam;i++)
    {
        std::cout<<"index: "<<i<<" "<<start->dado<<std::endl;
        start=start->prox;
    }
}
void tamLista(Lista *lista)
{
    std::cout<<"Tamanho da lista: "<<lista->tam<<std::endl;
}
void removeLista(Lista *lista, int numero)
{
    //ponteiros auxiliares
    no *start= lista->start;
    no *ptrash=NULL;
    //Caso o elemento a ser removido seja o primeiro
    if (lista->start->dado==numero && start!=NULL)
    {
        ptrash=lista->start;
        lista->start=ptrash->prox;
        //Apontamos nosso fim para nulo
        if (lista->start==NULL)
        {
            lista->end= NULL;
        }
    }
```

```
    else
    {
        //movemos o ponteiro//lista até encontrar o elemento
        while(start->prox !=NULL && start->prox->dado !=numero && start !=NULL)
        {
            start=start->prox;
        }
        if (start !=NULL && start->prox !=NULL)
        {
            ptrash= start->prox;
            start->prox=ptrash->prox;
            //Caso o último elemento for removido
            if (start->prox==NULL)
            {
                lista->end=start;
            }
        }
    }
    //limpamos o lixo e diminuimos o tamanho da lista
    if (ptrash)
    {
        free(ptrash);
        lista->tam--;
    }
}
no* findElement(Lista *lista,int numero)
{
    no *start= lista->start;
    for(start;start!=NULL;start=start->prox)
    {
        if (start->dado ==numero) return  start;
    }
    return NULL;
}
bool inLista(Lista *lista,int numero)
{
    no *start= lista->start;
    for(start;start!=NULL;start=start->prox)
    {
        if (start->dado ==numero) return true;
    }
    return false;
}

int particao(std::vector<int>& usagi,int e_esq,int e_dir)
{
    int e_pivo=usagi[e_dir];
    int rabbit=e_esq-1;
    for(int elem=e_esq;elem<e_dir;elem++)
    {
        if (usagi[elem]<=e_pivo)
        {
```

```
            rabbit=rabbit+1;
            int ino=usagi[rabbit];
            usagi[rabbit]=usagi[elem];
            usagi[elem]=ino;
        }
    }
    int ino=usagi[e_dir];
    usagi[e_dir]=usagi[1+rabbit];
    usagi[1+rabbit]=ino;
    return 1+rabbit;
}
//use quick_sort(vetor,0,tamanho do vetor-1)
void quick_sort(std::vector<int>& usagi,int e_esq,int e_dir)
{
    if (usagi.size()==1 || usagi.size()==0)
    {
        return;
    }
    if (e_esq < e_dir)
    {
        int e_pivo = particao(usagi,e_esq,e_dir);
        quick_sort(usagi,e_esq,e_pivo-1);
        quick_sort(usagi,e_pivo+1,e_dir);
    }
}
void trocar(std::vector<int>& vetor, int i, int j)
{
    int aux= vetor[i];
    vetor[i] = vetor[j];
    vetor[j] =aux;
}
int particiona(std::vector<int>& vetor, int inicio, int fim)
{
    int e_pivo= vetor[fim];
    int rabbit= (inicio - 1);

    for (int j = inicio; j <= fim- 1; j++)
    {
        if (vetor[j] <=e_pivo)
        {
            rabbit++;
            trocar(vetor,rabbit,j);
        }
    }
    trocar(vetor,rabbit+1,fim);
    return (rabbit + 1);
}
//
void quick_sortImperativo(std::vector<int>& vetor,int inicio,int fim)
{
    int pilha[fim-inicio+1];
    int top=-1;
```

```
    pilha[++top]=inicio;
    pilha[++top]=fim;
    while (top>=0)
    {
        int fim= pilha[top--];
        int inicio =pilha[top--];
        int p=particiona(vetor,inicio,fim);
        if (p-1>inicio)
        {
            pilha[++top]=inicio;
            pilha[++top]=p-1;
        }
        if (p+1<fim)
        {
            pilha[++top]=p+1;
            pilha[++top]=fim;
        }
    }
}
void mostrarVetor(std::vector<int>& vetor)
{
    for(int i=0;i<vetor.size();i++)
    {
        std::cout<<vetor[i]<<std::endl;
    }
}
std::vector<int> toVector(Lista *lista)
{
    no* start=lista->start;
    std::vector<int>vetor;
    int i=0;
    for(start;start!=NULL;start=start->prox)
    {
        if (start!=NULL)
        {
            vetor.push_back(start->dado);
            i++;
        }
    }
    return vetor;
}
void deleteLista(Lista *lista)
{
    std::vector<int>aux=toVector(lista);
    for(int i=0;i<aux.size();i++)
    {
        removeLista(lista,aux[i]);
    }
    std::cout<<"Lista limpa"<<std::endl;
}
void toLista(std::vector<int>& vetor,Lista *lista)
{
```

```
    for(int i=0;i<vetor.size();i++)
    {
        addEnd(lista,vetor[i]);
    }
}
void orderLista(Lista *lista)
{
    std::vector<int> vetor=toVector(lista);
    quick_sort(vetor,0,vetor.size()-1);
    deleteLista(lista);
    toLista(vetor,lista);
}
//primeiro,ultimo-1,vetor,numero
int buscaBinariaRecursiva(int primeiro,int ultimo,int* vetor,int numero)
{
    int meio=(int)(primeiro+ultimo)/2;
    if(vetor[meio]==numero)
    {
        return meio;
    }
    else if (vetor[meio]>numero)
    {
        buscaBinariaRecursiva(primeiro,meio-1,vetor,numero);
    }
    else
    {
        buscaBinariaRecursiva(meio+1,ultimo,vetor,numero);
    }
}
//tamanho do vetor,vetor,numero desejado
int buscaBinariaImperativa(int tam,int* vetor,int numero)
{
    int primeiro=0,ultimo=tam-1;
    bool found=false;
    while(primeiro<=ultimo && found==false)
    {
        int meio=(int)(primeiro+ultimo)/2;
        if (vetor[meio]==numero)
        {
            found=true;
            return meio;
        }
        else
        {
            if (vetor[meio]>numero)
            {
                ultimo=meio-1;
            }
            else
            {
                primeiro=meio+1;
            }
```

```
        }

    }
    std::cout<<"número:"<<numero<<" não existe no vetor"<<std::endl;
    return -1;
}

//Aqui é a biblioteca de cabeçalho utills.h
#ifndef UTILLS_H
#define UTILLS_H
#include <vector>
#include <iostream>
/**
 * define um tipo de struct no que armazena inteiros e o proximo no
 *
 * Example:
 * no* aux=lista->start;
 * aux->dado;
 * aux->prox;
 */
typedef struct no
{
    int dado;
    no *prox;
} no;
/**
 * define um tipo Lista que guarda o começo e o fim como também o tamanho dela dinamicamente
 *
 * Example:
 * Lista lista;
 * lista->tam=0;
 * lista->start=NULL;
 * lista->end=NULL;
 */
typedef struct Lista
{
    no *start, *end;
    int tam;
} Lista;
/**
 * iniciar uma lista, primeira operação
 * Example:
 * createLista(&lista);
 */
void createLista(Lista *);
//uma função que gera inteiros seguros
int handle_exception();
/**
 * inserção no começo
 *
 * Example:
 * addStart(&lista,5);
```

```
 */
void addStart(Lista *, int);
/**
 * inserção no fim
 *
 * Example:
 * addEnd(&lista,5);
 */
void addEnd(Lista *, int);
/**
 * exibe o conteudo de uma lista
 * Example:
 * showLista(&lista);
 */
void showLista(Lista *);
/**
 * exibe o tamanho da lista
 *
 * Example:
 * tamlista(&lista);
 */
void tamLista(Lista *);
/**
 * remove um elemento
 *
 * Example:
 * removeLista(&lista,5);
 */
void removeLista(Lista *, int);
/**
 * verifica se um elemento está na lista e retorna true ou false
 *
 * Example:
 * inLista(&lista,2);
 */
bool inLista(Lista *, int);
/**
 * verifica se um elemento está na lista e retorna o no
 *
 * Example:
 * findElement(&lista,2);
 */
no *findElement(Lista *, int);
/**
 * transforma uma lista em vetor de mesmo tamanho
 *
 * Example:
 * toVector(&lista);
 */
std::vector<int> toVector(Lista *);
/**
 * transforma em lista a partir de um std::vector<int>
```

```
 *
 * Example:
 * toLista(vetor,&lista);
 */
void toLista(std::vector<int> &, Lista *);
/**
 * exibe um vetor std::vector<int>
 *
 * Example:
 * mostrarVetor(vetor);
 */
void mostrarVetor(std::vector<int> &);
/**
 * particiona um vetor função do quick_sort
 *
 * Example:
 * particao(vetor,inicio,fim);
 */
int particao(std::vector<int> &, int, int);
/**
 * ordenar um elemento std::vector<int> um array primitivo tambem é possivel com uma pequena alteração;
 *
 * Example:
 * quick_sort(vetor,0,vetor.size()-1);
 */
void quick_sort(std::vector<int> &, int, int);
/**
 * troca a posição de dois números em um vetor
 * Example:
 * trocar(vetor,4,5);
 */
void trocar(std::vector<int> &, int, int);
/**
 * particiona um vetor função do quick_sort
 *
 * Example:
 * particao(vetor,inicio,fim);
 */
int particiona(std::vector<int> &, int, int);
/**
 * ordenar um elemento std::vector<int> um array primitivo tambem é possivel com uma pequena alteração;
 * de modo imperativo ou interado
 * Example:
 * quick_sort(vetor,0,vetor.size()-1);
 */
void quick_sortImperativo(std::vector<int> &, int, int);
/**
 * excluir os elementos de uma lista e deixa vazia
 * Example:
 * deleteLista(&lista);
```

```
 */
void deleteLista(Lista *);
/**
 * ordenar uma lista utilizando a classe de std::vector como controle, todos os elementos serão excluidos e será
 * chamado a função de ordenação quick_sort o resultado será transformado em uma lista e adicionado na lista original.
 * Example:
 * orderLista(&lista);
 */
void orderLista(Lista *);
/**
 * busca um número em um vetor e retorna sua posição de forma recursiva
 * Example:
 * buscaBinariaRecursiva(primeiro,ultimo,vetor,numero_desejado);
 */
int buscaBinariaRecursiva(int, int, int *, int);
/**
 * busca um número em um vetor e retorna sua posição de forma imperativa
 * Example:
 * buscaBinariaImperativa(tamanho_vetor,vetor,numero_desejado);
 */
int buscaBinariaImperativa(int, int *, int);

#endif

//Aqui é o software principal main.cpp


#include "utills.h"
#include <iostream>
int menu(Lista *lista,int op)
{
   int numero;
   if (op==1 || op==2 || op==3)
   {
      numero=handle_exception();
      if (op==1)
         addStart(lista,numero);
      else if (op==2)
         addEnd(lista,numero);
      else
         removeLista(lista,numero);
   }
   else if (op==4)
      showLista(lista);
   else if (op==5)
      tamLista(lista);
   else if (op==7)
      return 999;
   else if(op==6)
   {
```

```
        numero=handle_exception();
        if(inLista(lista,numero))
        {
            std::cout<<"Elemento na lista:"<<numero<<std::endl;
        }
        else
            std::cout<<"Elemento nao existe na lista:"<<numero<<std::endl;
    }
    else if(op==8)
    {
        orderLista(lista);
    }
    else if(op==9)
    {
        deleteLista(lista);
    }
    else
    {
        std::cout<<"Error 1:Opção inválida, Digite novamente"<<std::endl;
        return 1;
    }
    return 0;
}
int q1()
{
    int op=0;
    Lista lista;
    createLista(&lista);
    do
    {
        std::cout<<"\nDigite 1: Adiciona número no começo\nDigite 2:Adicionar número no fim\
nDigite 3: Remover um número\nDigite 4:Mostrar Lista\nDigite 5:Mostrar tamanho da lista\nDigite
6:Para verificar se um elemento está na lista\nDigite 7:Para encerrar\nDigite 8:Para ordenar a lista\
n\Digite 9:Para excluir a lista\n"<<std::endl;
        op=handle_exception();
        op=menu(&lista,op);
        if (op==999)
            break;

    } while (op!=999);
    return 0;
}
int q2()
{
    std::cout<<"Parta a modo recursivo!"<<std::endl;
    std::vector<int>usagi={1,2,40,10,30,2,34,1,23,1,31,31,3,3,4,5,89};
    quick_sort(usagi,0,usagi.size()-1);
    mostrarVetor(usagi);
    std::cout<<"Parta b modo imperativo!"<<std::endl;
    std::vector<int> usagi1={1,25,3,142,312};
    quick_sortImperativo(usagi1,0,usagi.size()-1);
    mostrarVetor(usagi1);
```

```
    return 0;
}
int q3()
{
    int v1[] = {1, 3, 5, 6, 9, 12, 15, 20, 25};
    std::cout<<"Parte a modo imperativo";
    int n=buscaBinariaImperativa(9,v1,20);
    std::cout<<"Parte b modo recursivo";
    int n1=buscaBinariaRecursiva(0,9,v1,20);
    std::cout<<"Numero encontrado em:"<<n<<std::endl;
    std::cout<<"Numero encontrado em:"<<n1<<std::endl;
    return 0;
}

int main()
{
    q1();
    q2();
    q3();
    return 0;
}
```