const int MAXLISTSIZE = 100000;

template<class ElemType>

class SqList

{

private:

ElemType \*elem; // 存储空间基址

int length; // 当前长度

int listsize; // 允许的最大存储容量(以sizeof(ElemType)为单位

public:

//初始化顺序表

SqList(int ms = 20);

//初始化顺序表

SqList(ElemType List[], int Len);

//删除顺序表

~SqList(){delete [] elem;}

//将顺序表置为空表

void ListClear( ){length = 0;}

//返回顺序表的长度

int ListLength() const {return length;}

//设置顺序表的长度

bool SetListLength(int len);

//判断顺序表是否为空表

bool ListEmpty() const;

//判断顺序表是否为满表

bool ListFull() const;

//用e返回顺序表的第i个元素

ElemType GetElem(int i) const;

//用e设置顺序表的第i个元素

bool SetElem(int i, ElemType e);

//在顺序表的第pos个位置之前插入e元素

bool ListInsert(int pos,ElemType e);

//删除顺序表的第pos个位置的元素

bool ListDelete(int pos, ElemType &e);

//compare函数，用来判断a和b是否相等

bool compare(ElemType a, ElemType \*b);

//按指定条件查找

int LocateElem(ElemType e);

//逆置顺序表

void Invert(int, int);

//返回线性表给定数据元素的前驱数据元素的值

bool PriorElem(ElemType cur\_e, ElemType &pri\_e);

//返回线性表给定数据元素的后继数据元素的值

bool NextElem(ElemType cur\_e, ElemType &nex\_e);

//销毁线性表

void ListDestroy();

//遍历顺序表

int ListTraverse() const;

};

/\* 单链表的结点定义 \*/

template<class ElemType>

struct LinkNode

{

ElemType data;

LinkNode<ElemType> \*next;

LinkNode(LinkNode<ElemType> \*ptr = NULL){next = ptr;} //构造函数1，用于构造头结点

LinkNode(const ElemType &item, LinkNode<ElemType> \*ptr = NULL) //构造函数2，用于构造其他结点

//函数参数表中的形参允许有默认值，但是带默认值的参数需要放后面

{

next = ptr;

data = item;

}

ElemType getData(){ return data;} //取得结点中的数据

void SetLink( LinkNode<ElemType> \*link ){ next = link; } //修改结点的next域

void SetLink( ElemType value ){ data = value; } //修改结点的next域

};

//带头结点的单链表

template<class ElemType>

class LinkList

{

private:

LinkNode<ElemType> \*head; // 头结点

LinkNode<ElemType> \*tail; // 尾结点

public:

//无参数的构造函数

LinkList(){head = new LinkNode<ElemType>;}

//带参数的构造函数

LinkList(const ElemType &item){head = new LinkNode<ElemType>(item);}

//拷贝构造函数

LinkList(LinkList<ElemType> &List);

//析构函数

~LinkList(){ListDestroy();}

//重载函数:赋值

//LinkList<ElemType>& operator=(LinkList<ElemType> &List);

//销毁链表

void ListDestroy();

//清空链表

void ListClear();

//返回链表的长度

int ListLength() const;

//判断链表是否为空表

bool ListEmpty() const;

//在首节点之前插入一个结点

bool InsFirst( ElemType e );

//获取链表头结点

LinkNode<ElemType>\* GetHead() { return head;}

//设置链表头结点

void SetHead(LinkNode<ElemType> \*p){ head = p;}

//获取链表尾结点

LinkNode<ElemType>\* GetTail() { return tail;}

//用e返回链表的第i个元素

ElemType GetElem(int pos);

//在链表的第pos个位置之前插入e元素

bool ListInsert\_prior(int pos,ElemType e);

//在链表的第pos个位置之后插入e元素

bool ListInsert\_next(int pos,ElemType e);

//删除链表的首结点

//bool DelFirst( ElemType &e);

//表头插入法动态生成链表

void CreateList\_Head(int n, ElemType \*A);

//表尾插入法动态生成链表

void CreateList\_Tail(int n, ElemType \*A);

//删除链表的第pos个位置的元素

ElemType ListDelete(int pos);

//compare函数，用来判断a和b是否相等

//bool compare(ElemType a, ElemType \*b);

//按序号查找，从链表的第一个结点开始，判断当前结点是否是第i个，

//若是，则返回该结点的数据域的值；否则继续后一个，直至表结束。没有第i个结点时返回空。

bool FindElem( int k, ElemType &e);

//bool compare(ElemType a, ElemType \*b);

//按值查找，即定位。从链表的第一个结点开始，判断当前结点值是否等于e。

//若是，则返回该结点的序号；否则继续后一个，直至表结束。找不到时返回0。

int SearchElem( const ElemType &e) const;

//返回链表给定数据元素的前驱数据元素的值

//bool PriorElem(ElemType cur\_e, ElemType &pri\_e);

//返回链表给定数据元素的后继数据元素的值

bool NextElem(LinkNode<ElemType> \*p, ElemType &e);

//遍历链表

bool ListTraverse() const;

};