（1）针对带附加头结点的单链表，编写求最大值函数max：通过一趟遍历在单链表中确定值最大的结点。

解：// 求链表list中的值最大值的结点

template<typename T, typename E>

E max(const List<T,E>& list)

{

E maximum, Element; // 最大值和元素

// 链表中的第1个元素作为最初的最大值

list.getData(1,maximum);

int length=list.Length();

for(int i=2;i<=length;i++)

{

// 取链表中的第i个元素Element

list.getData(i,Element);

if(Element>maximum)

maximum=Element;

}

return maximum;

}

（2）已知函数A定义如下，写出它的递归求解算法。

A(0,n)=n+1 　当n≥0时

A(m,0)=A(m-1,1) 　当m>0时

A(m,n)=A(m-1,A(m,n-1))　　　当m>0, n>0时

解：根据函数A定义，写出它的递归求解算法：

double A(double m, double n)

{

if(m==0 && n>=0)

return n+1;

else if(m>0 && n==0)

return A(m-1,1);

else if(m>0 && n>0)

return A(m-1,A(m,n-1));

}

（3）已知针对带头结点的单链表类模板定义和部分成员函数代码如下所示，请设计如下成员函数的代码：

1、设计输出函数 void Output(void);输出单链表中元素个数和全部元素；（本题3分）

#include <iostream>

#include <stdlib.h>

using namespace std;

//链结点类模板定义

template <class T, class E>

class LinkNode

{

public:

LinkNode(void); // 构造函数

E data; // 结点数据

LinkNode<T, E>\* link; // 后继指针

LinkNode(E item, LinkNode<T, E>\* next = NULL); // 构造函数

};

// 构造函数

template <class T, class E>

LinkNode<T,E>::LinkNode(void)

{

link = NULL;

}

// 构造函数

template<typename T, typename E>

LinkNode<T,E>::LinkNode(E item, LinkNode<T, E>\* next)

{

data = item;

link = next;

}

// 单链表类模板定义

template <class T, class E>

class List

{

public:

List(void); // 构造函数

~List(void); // 析构函数

protected:

// 链表头指针，头结点地址

LinkNode<T,E>\* first; // 单链表由若干结点组成

public:

List(E x); // 构造函数

void makeEmpty(void); // 清空链表(保留表头结点)

void setFirst(LinkNode<T,E>\* f);// 将f 设为新的链首first←f

void inputRear(T endTag); // 后插法建立单链表

void Output(void); // 输出单链表

// 定位第i个结点，若i=0，定位头结点，若i<0或i超出表长返回NULL

LinkNode<T,E>\* Locate(int i) const;

};

// 构造函数

template <class T, class E>

List<T,E>::List(void)

{

first = new LinkNode<T,E>;

}

// 析构函数

template <class T, class E>

List<T,E>::~List(void)

{

makeEmpty();

delete first;

}

// 构造函数

template<typename T, typename E>

List<T,E>::List(E x)

{

first = new LinkNode<T,E>(x);

}

// 清空链表(保留表头结点)

template<typename T, typename E>

void List<T,E>::makeEmpty(void)

{

LinkNode<T, E> \*del;

//当链第 1 个元素存在时

while (first->link != NULL)

{

del = first->link; //保存原第 1 个元素地址

//链第 1 个元素的后继结点成为新的第 1 个元素

first->link = del->link;

delete del; //删除原第 1 个元素

}

}

//将f设为新的链首first←f

template<typename T, typename E>

void List<T,E>::setFirst(LinkNode<T, E>\* f)

{

first = f;

}

//从标准输入(键盘)逐个输入数据，建立单链表，以endTag值为结束标志

template<typename T, typename E>

void List<T,E>::inputRear(T endTag)

{

cout<<"开始输入单链表，以 "<<endTag<<" 值为结束标志：";

LinkNode<T,E> \*newNode, \*last;

E val;

last = new LinkNode<T,E>; //建立链表的头结点

setFirst(last); //将链表的尾结点设置为链头结点

cin >> val; //输入新的值

while ( val != endTag )//新输入的值不是结束值

{

newNode = new LinkNode<T,E>(val);//构造新元素

last->link = newNode; //新元素成为链尾元素的新后继

last = newNode; //last指向新的链尾(新)元素

cin >> val; //输入新的值

}

last->link = NULL; //链尾元素的后继为NULL(无后继)

}

解：

1、// 输出单链表元素个数和全部元素

template<typename T, typename E>

void List<T,E>::Output(void)

{

}

参考答案

解：1、设计输出函数：(每行代码0.3分，3分)

// 输出单链表中元素个数和全部元素

template<typename T, typename E>

void List<T,E>::Output(void)

{

//输出单链表头结点数据域

cout<<"单链表：[头结点]";

//定义元素计数器，初始化为0

int Count = 0;

//定义当前结点指针，初始化为头结点的后继地址

LinkNode<T,E>\* current = first->link;

//当前结点地址未到链尾时，执行循环体

while ( current != NULL )

{

//输出后继域[-]和后继元素值

cout<<"[-]→[ "<<current->data<<" ]";

Count++; //元素计数器增 1

//后继结点地址成为新当前结点地址

current = current->link;

}

//当前结点地址为链尾，输出[^]和元素个数后换行

cout<<"[^]，共 "<<Count<<" 个表项。"<<endl;

}

（4）已知针对带头结点的单链表类模板定义和部分成员函数代码如下所示，请设计如下成员函数的代码：

2、设计定位函数 LinkNode<T,E>\* Locate(int i) const; 若找到第i个结点，则返回其地址；当i=0时，返回表头结点的地址；若 i<0 或 i 超出表长，则返回NULL；（本题4分）

#include <iostream>

#include <stdlib.h>

using namespace std;

//链结点类模板定义

template <class T, class E>

class LinkNode

{

public:

LinkNode(void); // 构造函数

E data; // 结点数据

LinkNode<T, E>\* link; // 后继指针

LinkNode(E item, LinkNode<T, E>\* next = NULL); // 构造函数

};

// 构造函数

template <class T, class E>

LinkNode<T,E>::LinkNode(void)

{

link = NULL;

}

// 构造函数

template<typename T, typename E>

LinkNode<T,E>::LinkNode(E item, LinkNode<T, E>\* next)

{

data = item;

link = next;

}

// 单链表类模板定义

template <class T, class E>

class List

{

public:

List(void); // 构造函数

~List(void); // 析构函数

protected:

// 链表头指针，头结点地址

LinkNode<T,E>\* first; // 单链表由若干结点组成

public:

List(E x); // 构造函数

void makeEmpty(void); // 清空链表(保留表头结点)

void setFirst(LinkNode<T,E>\* f);// 将f 设为新的链首first←f

void inputRear(T endTag); // 后插法建立单链表

void Output(void); // 输出单链表

// 定位第i个结点，若i=0，定位头结点，若i<0或i超出表长返回NULL

LinkNode<T,E>\* Locate(int i) const;

};

// 构造函数

template <class T, class E>

List<T,E>::List(void)

{

first = new LinkNode<T,E>;

}

// 析构函数

template <class T, class E>

List<T,E>::~List(void)

{

makeEmpty();

delete first;

}

// 构造函数

template<typename T, typename E>

List<T,E>::List(E x)

{

first = new LinkNode<T,E>(x);

}

// 清空链表(保留表头结点)

template<typename T, typename E>

void List<T,E>::makeEmpty(void)

{

LinkNode<T, E> \*del;

//当链第 1 个元素存在时

while (first->link != NULL)

{

del = first->link; //保存原第 1 个元素地址

//链第 1 个元素的后继结点成为新的第 1 个元素

first->link = del->link;

delete del; //删除原第 1 个元素

}

}

//将f设为新的链首first←f

template<typename T, typename E>

void List<T,E>::setFirst(LinkNode<T, E>\* f)

{

first = f;

}

//从标准输入(键盘)逐个输入数据，建立单链表，以endTag值为结束标志

template<typename T, typename E>

void List<T,E>::inputRear(T endTag)

{

cout<<"开始输入单链表，以 "<<endTag<<" 值为结束标志：";

LinkNode<T,E> \*newNode, \*last;

E val;

last = new LinkNode<T,E>; //建立链表的头结点

setFirst(last); //将链表的尾结点设置为链头结点

cin >> val; //输入新的值

while ( val != endTag )//新输入的值不是结束值

{

newNode = new LinkNode<T,E>(val);//构造新元素

last->link = newNode; //新元素成为链尾元素的新后继

last = newNode; //last指向新的链尾(新)元素

cin >> val; //输入新的值

}

last->link = NULL; //链尾元素的后继为NULL(无后继)

}

解：

2、设计定位函数

//若找到第i个结点，则返回其地址；当i=0时，返回表头结点的地址；

//若 i<0 或 i 超出表长，则返回NUL

template<typename T, typename E>

LinkNode<T, E>\* List<T,E>::Locate(int i) const

{

}

参考答案

解：

2、设计定位函数：(每行代码0.4分，4分)

//若找到第i个结点，则返回其地址；若i=0，定位头结点地址，

//若i<0或i超出表长，则返回NULL

template<typename T, typename E>

LinkNode<T, E>\* List<T,E>::Locate(int i) const

{

if (i < 0) //i不合理

return NULL; //返回NULL

//定义当前结点地址，初始化为头结点地址

LinkNode<T,E>\* current = first;

//定义元素位置，初始化为0(头结点位置)

int k = 0;

//当前结点地址未到链尾且未到第i个元素位置

while ( current != NULL && k < i )

{ //后继结点地址成为新当前结点地址

current = current->link;

//元素的位置后移 1 位

k++;

}

//返回第 i 号结点地址或NULL

return current;

}

（5）已知针对带头结点的单链表类模板定义和部分成员函数代码如下所示，请设计如下成员函数的代码：

3、在主函数构造关键码类型T和元素类型E都为int型的单链表list，输入数据输入结束标志、待定位的第几个结点和单链表，输出单链表、待定位的结点的地址，若待定位的结点不存在，则输出该结点不存在。（本题4分）

#include <iostream>

#include <stdlib.h>

using namespace std;

//链结点类模板定义

template <class T, class E>

class LinkNode

{

public:

LinkNode(void); // 构造函数

E data; // 结点数据

LinkNode<T, E>\* link; // 后继指针

LinkNode(E item, LinkNode<T, E>\* next = NULL); // 构造函数

};

// 构造函数

template <class T, class E>

LinkNode<T,E>::LinkNode(void)

{

link = NULL;

}

// 构造函数

template<typename T, typename E>

LinkNode<T,E>::LinkNode(E item, LinkNode<T, E>\* next)

{

data = item;

link = next;

}

// 单链表类模板定义

template <class T, class E>

class List

{

public:

List(void); // 构造函数

~List(void); // 析构函数

protected:

// 链表头指针，头结点地址

LinkNode<T,E>\* first; // 单链表由若干结点组成

public:

List(E x); // 构造函数

void makeEmpty(void); // 清空链表(保留表头结点)

void setFirst(LinkNode<T,E>\* f);// 将f 设为新的链首first←f

void inputRear(T endTag); // 后插法建立单链表

void Output(void); // 输出单链表

// 定位第i个结点，若i=0，定位头结点，若i<0或i超出表长返回NULL

LinkNode<T,E>\* Locate(int i) const;

};

// 构造函数

template <class T, class E>

List<T,E>::List(void)

{

first = new LinkNode<T,E>;

}

// 析构函数

template <class T, class E>

List<T,E>::~List(void)

{

makeEmpty();

delete first;

}

// 构造函数

template<typename T, typename E>

List<T,E>::List(E x)

{

first = new LinkNode<T,E>(x);

}

// 清空链表(保留表头结点)

template<typename T, typename E>

void List<T,E>::makeEmpty(void)

{

LinkNode<T, E> \*del;

//当链第 1 个元素存在时

while (first->link != NULL)

{

del = first->link; //保存原第 1 个元素地址

//链第 1 个元素的后继结点成为新的第 1 个元素

first->link = del->link;

delete del; //删除原第 1 个元素

}

}

//将f设为新的链首first←f

template<typename T, typename E>

void List<T,E>::setFirst(LinkNode<T, E>\* f)

{

first = f;

}

//从标准输入(键盘)逐个输入数据，建立单链表，以endTag值为结束标志

template<typename T, typename E>

void List<T,E>::inputRear(T endTag)

{

cout<<"开始输入单链表，以 "<<endTag<<" 值为结束标志：";

LinkNode<T,E> \*newNode, \*last;

E val;

last = new LinkNode<T,E>; //建立链表的头结点

setFirst(last); //将链表的尾结点设置为链头结点

cin >> val; //输入新的值

while ( val != endTag )//新输入的值不是结束值

{

newNode = new LinkNode<T,E>(val);//构造新元素

last->link = newNode; //新元素成为链尾元素的新后继

last = newNode; //last指向新的链尾(新)元素

cin >> val; //输入新的值

}

last->link = NULL; //链尾元素的后继为NULL(无后继)

}

解：

3、设计主函数

// 主函数

int \_tmain(int argc, \_TCHAR\* argv[])

{

}

参考答案

解：

3、主函数：(每行代码0.3分，4分)

int \_tmain(int argc, \_TCHAR\* argv[])

{

//构造1个元素和关键值都为int型的单链表list

List<int,int> list;

//定义表示数据输入结束标志和元素位置

int endTag,Index;

//输出提示信息"输入一个整数表示数据输入结束的标志："

cout<<"输入一个整数表示数据输入结束的标志：";

//输入表示数据输入的值

cin>>endTag;

//逐个输入数据，后插法建立单链表，以endTag值为结束标志

list.inputRear(endTag);

//输出单链表中元素个数和全部元素

list.Output();

//输出提示信息"输入要定位第几个数的地址？"

cout<<"输入要定位第几个数的地址？";

//输入要定位的第几个数

cin>>Index;

//定义结点地址存放定位第Index个结点地址，

//若Index=0，定位头结点，若Index<0或Index超出表长返回NULL

LinkNode<int,int>\* Address=list.Locate(Index);

//定位的结点地址不存在

if(Address==NULL)

//输出第Index个数不存在

cout<<"第 "<<Index<<" 个数不存在。"<<endl;

else

//输出第Index个数的地址为Address的值

cout<<"第 "<<Index<<" 个数的地址为："<<Address<<endl;

return 0;

}

（6）已知顺序表类模板定义和部分成员函数代码如下所示，请设计如下成员函数的代码：

1、设计输出函数void output(void);输出顺序表中元素个数和全部元素；

2、设计调整函数void Part(void);将顺序表调整为前面的元素都为奇数，后面的元素都为偶数，调整完成后输出提示信息：顺序表已调整为前面的元素都为奇数，后面的元素都为偶数；

3、在主函数构造关键码类型T和元素类型E都为int型的顺序表seqList，输入顺序表，输出初始顺序表，将其调整为前面的元素都为奇数，后面的元素都为偶数，输出调整后的顺序表。(本题11分)

#include <iostream>

#include <tchar.h>

using namespace std;

const int defaultSize = 100;

// 顺序表类定义

template <class T,class E>

class SeqList :

{

public:

// 构造函数

SeqList(int sz = defaultSize);

// 析构函数

~SeqList(void);

protected:

// 存放数组

E\* data;

// 最大可容纳表项的项数

int maxSize;

// 当前已存表项数

int currentSize;

public:

// 输入顺序表中元素个数和全部元素

void input(void);

// 输出顺序表中元素个数和全部元素

void output(void);

// 将顺序表调整为前面的元素都为奇数，后面的元素都为偶数

void Part(void);

};

// 构造函数

template <class T,class E>

SeqList<T,E>::SeqList(int sz)

{

if (sz > 0)

{

maxSize = sz;

currentSize = 0;

data = new E[maxSize]; //创建表存储数组

if (data == NULL) //动态分配失败

{

cerr << "存储分配错误！" << endl;

exit(1);

}

}

}

// 析构函数

template <class T,class E>

SeqList<T,E>::~SeqList(void)

{

delete[] data;

}

// 输入顺序表中元素个数和全部元素

template <class T,class E>

void SeqList<T,E>::input(void)

{

//从标准输入(键盘)逐个输入数据,建立顺序表

cout<<"开始建立顺序表，请输入表中元素个数：";

cin>>currentSize;

if(currentSize>maxSize)

{

delete []data;

maxSize=2\*currentSize;

data = new E[maxSize]; //创建表存储数组

if (data == NULL) //动态分配失败

{

cerr << "存储分配错误！" << endl;

exit(1);

}

}

cout<<"输入表中共 "<<currentSize<<" 个元素：";

for(int i=0;i<currentSize;i++)

cin>>data[i];

}

十、解：

// 输出顺序表中元素个数和全部元素

template <class T,class E>

void SeqList<T,E>::output(void)

{

}

// 将顺序表调整前面的元素都为奇数，后面的元素都为偶数

template<typename T, typename E>

void SeqList<T,E>::Part(void)

{

}

int \_tmain()

{

}

解：

1、输出函数：(每行代码0.5分，2分)

// 输出顺序表中元素个数和全部元素

template <class T,class E>

void SeqList<T,E>::output(void)

{

//输出顺序表中的元素个数currentSize

cout<<"顺序表共 "<<currentSize<<" 个表项(";

//逐个输出顺序表中的元素，并以"，"分隔

for(int i=0;i<currentSize;i++)

cout<<data[i]<<", ";

cout<<")"<<endl;

}

2、调整函数：(每行代码0.5分，6分)

// 将顺序表调整为前面的元素都为奇数，后面的元素都为偶数

template<typename T, typename E>

void SeqList<T,E>::Part(void)

{

//定义奇数下标old从0开始，

//偶数下标even从末元素开始

int old=0,even=currentSize-1;

//定义交换奇偶元素的中间元素Element

E Element;

//当奇数下标old和偶数下标even未重叠时，

//执行循环体

while(old<even)

{

//当元素data[old]为奇数时，执行循环体

//当元素data[old]为偶数时，退出循环

while(data[old]%2==1)

old++; //奇数下标old后移

//当元素data[even]为偶数时，执行循环体

//当元素data[old]为奇数时，退出循环

while(data[even]%2==0)

even--; //偶数下标even前移

//如果奇数下标old和偶数下标even未重叠

if(old<even)

{

//交换偶数元素data[old]和奇数元素data[even]

Element=data[old];

data[old]=data[even];

data[even]=Element;

}

}

cout<<"顺序表已调整为前、后2部分，前面的元素都为奇数，后面的元素都为偶数。"<<endl;

}

3、主函数：(每行代码0.5分，3分)

int \_tmain()

{

//构造关键码和元素类型都为int型的顺序表seqList

SeqList<int,int> seqList;

//输入顺序表中的元素个数和全部元素

seqList.input();

//输出顺序表中的元素个数和全部元素

seqList.output();

// 将顺序表调整为前面的元素都为奇数，后面的元素都为偶数

seqList.Part();

//输出顺序表中的元素个数和全部元素

seqList.output();

return 0;

}

（7）设链表类的头文件list.h已经实现其有关成员函数和重载了流输出。试编写应用程序，将链表la（元素类型自定）中的正数集中到其前段,负数集中到后段，再输出结果。

解：

using namespace std;

#include ".\list.cpp"

int \_tmain()

{

List<double,double> la;

double endTag,Number;

cout<<"输入一个表示数据输入结束的整数：";

cin>>endTag;

la.inputRear(endTag);

la.output();

List<double,double> lb(la);

la.makeEmpty();

while(lb.IsEmpty()==false)

{

if( lb.Remove(1,Number)==true && Number>=0 )

la.insertFront(Number);

else if( Number<0 )

la.insertRear(Number);

}

la.output();

return 0;

}

// 程序设计题不设标准答案。