山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：201700130033	姓名： 武学伟	班级： 2017 级 2 班
实验题目：链式描述线性表		
实验学时：4	实验日期：2018/10/22	

实验目的：

1.掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。

2.掌握链表迭代器的实现与应用

软件环境：

Win10home，codeblocks

1. 实验内容（题目内容，输入要求，输出要求）

题目内容：

1) 创建线性表类：线性表的存储结构使用单链表；提供操作：自表首插入元素、删除指定元素、搜索表中是否有指定元素、输出链表。
2) 接受键盘录入的一系列整数作为节点的元素值，创建链表，输出链表内容。
3) 输入一个整数，在链表中进行搜索，输出元素的索引，如果不存在输出-1。
4) 设计实现链表迭代器，使用链表迭代器实现链表的反序输出
5) 创建两个有序链表，使用链表迭代器实现链表的合并。

输入要求：

1) 创建第一个链表，初始化为 6 5 4 3 2 1 并输出
2) 查找 0，查找 6
3) 删除-1，删除 2
4) 对链表进行反序并输出
5) 创建另一个链表，初始化为 2 2 6 8 10 并输出
6) 将第一个链表和第二个链表合并并输出

输出要求（预计输出）：

1) 6 5 4 3 2 1
2) -1 4
3) error succeed
4) 1 2 3 4 5
5) 2 2 6 8 10
6) 1 2 2 2 3 4 5 6 8 10

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

数据结构：链表描述的线性表

思路描述：创建结构体 chainNode，创建类 chain，定义功能函数插入，删除，查询、输出。插入，删除，查询均利用索引值来实现。在链表类的定义中加入 lastNode，并实现尾插入。

链表反序输出：

①定义指向链表头的指针 p，指向链表尾的指针 t，指向链表头指针指向的元素的指针 q（q = firstNode->next）。

②t 指向 q->next，q->next 指向 p，p 指向 q，q 指向 t，即让 q->next 前进两

位，p 顺延一位，p 和 t 同时指向 q->next，用来判断是否达到了尾节点。
③当 q 的值为 NULL 时，说明原链表到了尾部，这是链表完成了倒序
④设定倒叙完成的链表头和尾，此时链表尾应该是 p 所指向的位置，链表头是 firstNode->next
合并两个链表：
①定义类外的函数 merge，参数是两个链表的常引用 a，b。
②定义两个迭代器 A，B 分别指向链表 a，链表 b 的首节点，定义一个链表 c，用于链表合并之后的输出。
③利用迭代器遍历链表，并对 a 和 b 的元素进行比较，当 a 的元素大于 b 时，将 B 解引用的值插入到 c 的尾部，否则插入 A 解引用的值。
④判断链表 a，b 是否遍历完成，若 a 为遍历完成，将 A 的剩余元素插入到 c 的尾部，否则将 b 的剩余元素插入到 c 的尾部。

3. 测试结果（测试输入，测试输出，结果分析）

测试输入：

1) 创建第一个链表，初始化为 6 5 4 3 2 1 并输出
2) 查找 0，查找 6
3) 删除-1，删除 2
4) 对链表进行反序并输出
5) 创建另一个链表，初始化为 2 2 6 8 10 并输出
6) 将第一个链表和第二个链表合并并输出

测试输出：

1) 6 5 4 3 2 1
2) -1 4
3) error succeed
4) 1 2 3 4 5
5) 2 2 6 8 10
6) 1 2 2 2 3 4 5 6 8 10

程序验证：

```
Input the number of this chain:
6
Input No.1: 6
Input No.2: 5
Input No.3: 4
Input No.4: 3
Input No.5: 2
Input No.6: 1
Input succeed!
Chain A is:
6 5 4 3 2 1
Input what you want to search:0
0 is No.-1 in this chain
0 has not been found!
Continue? 1. Yes 2. No
1
Input what you want to search:6
6 is No.0 in this chain
Continue? 1. Yes 2. No
2
Input what you want to pop:-1
-1 has not been found!
Continue? 1. Yes 2. No
1
Input what you want to pop:2
2 is No.4 in this chain
succeed!
Continue? 1. Yes 2. No
2
Reversed chain A is:
1 3 4 5 6
Input the number of this chain:
5
Input No.1: 2
Input No.2: 2
Input No.3: 6
Input No.4: 8
Input No.5: 10
Input succeed!
Chain B is:
2 2 6 8 10
Chain C (merge A and B) is:
1 2 2 3 4 5 6 6 8 10
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

1) 反序链表时，将指针 p 和 q 向后移了一位发现这样无法将链表的第一个元素倒序，所以更改了 p 和 q 的定义
2) 定义迭代器 A，B 时报错，发现时没有指明 chain<T>::Iterator A 的类型，将 T 改为 int 即可

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

/*实验四*/

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

/*定义结构体*/
template <class T>
struct chainNode
{
    T element;
    chainNode<T> * next;    //数据成员

    chainNode() {}
    chainNode(const T& element)
    {
        this->element = element;
    }
    chainNode(const T& element, chainNode<T>* next)
    {
        this->element = element;
        this->next = next;
    }     //方法函数
};

/*定义链表描述的线性表*/
template <class T>
class chain
{
    protected:
        chainNode<T> *firstNode;     //链表的头指针
        chainNode<T> *lastNode; //链表的尾指针
        int listsize;     //链表的长度
        bool trueindex;
    public:
        chain(int initialCapacity = 10);  //构造函数
        chain(const chain<T>&);   //复制构造函数
        ~chain();       //析构函数

        bool Empty() const {return listsize == 0;}    //判断链表是否
为空
        int Size() const {return listsize;}  //获取线性表长度
        T& Get(int theIndex) const;    //获取索引元素值
        int Find(const T& theElement) const;    //查找元素索引
        void Erase(int theInedx);    //删除索引元素
        void Insert(int theIndex, const T& theElement);    //在索引
处插入元素
```

```
        void Output() const;     //输出线性表
        void Reverse(); //倒序
        void Push_back(const T& theElement);//尾插入
        class Iterator;
        Iterator Begin() {return Iterator(firstNode);}
        Iterator End() {return Iterator(NULL);}
        class Iterator
        {
            protected:
                chainNode<T> *node;
            public:
                /*构造函数*/
                Iterator(chainNode<T> *theNode = NULL)
                {
                    node = theNode;
                }
                T& operator * () const {return node->element;}  //
解引用，获取值
                T* operator -> () const {return &node->element;}  //
获取地址
                /*加法*/
                Iterator& operator++ ()
                {
                    node = node->next;
                    return *this;
                }
                Iterator operator++ (int)
                {
                    Iterator old = *this;
                    node = node->next;
                    return old;
                }
                /*相等与不等校验*/
                bool operator != (const Iterator Right) const {return
node != Right.node;}
                bool operator == (const Iterator Right) const {return
node == Right.node;}
        };
};


/*构造函数*/
template <class T>
chain<T>::chain(int initialCapacity)
```

```
{
    if (initialCapacity < 1)
    {
        cout << "Initial capacity = " << initialCapacity << "must
be > 0" << endl;
        throw "wrong initial capacity";
    }
    firstNode = NULL;
    lastNode = NULL;
    listsize = 0;
    trueindex = true;
}

/*复制构造函数*/
template <class T>
chain<T>::chain(const chain<T>& theList)
{
    trueindex = true;
    listsize = theList.listsize;
    lastNode = NULL;
    if (listsize==0)
    {
        firstNode = NULL;
        lastNode = NULL;
        trueindex = true;
        return;
    }
    chainNode<T> *sourceNode = theList.firstNode;
    firstNode = new chainNode<T>(sourceNode->element); //复制链表的
元素
    sourceNode = sourceNode->next;
    chainNode<T> *targetNode = firstNode;    //target 赋值当前链表的
最后一个节点
    while (sourceNode!=NULL)
    {
        targetNode->next = new chainNode<T>(sourceNode->element);
        targetNode = targetNode->next;
        sourceNode = sourceNode->next;
    }
    targetNode->next = NULL;
}

/*析构函数*/
template <class T>
```

```
chain<T>::~chain()
{
    while (firstNode != NULL)
    {
        chainNode<T> *nextNode = firstNode->next;
        delete firstNode;
        firstNode = nextNode;
    }
}

/*获取索引元素值*/
template <class T>
T& chain<T>::Get(int theIndex) const
{
    trueindex = true;
    if (theIndex<0 || theIndex>=listsize)
    {
        cout << "wrong index" << endl;
        cout << "index = " << theIndex << " size = " << listsize <<
endl;
        trueindex = false;
    }          //检查元素
    if (trueindex)
    {
        chainNode<T> *currentNode = firstNode;
        for (int i = 0; i< theIndex; i++)
            currentNode = currentNode->next;
        return currentNode->element;
    }

}

/*查找元素索引*/
template <class T>
int chain<T>::Find(const T& theElement) const
{
    chainNode<T> * currentNode = firstNode;
    int index = 0;     //当前节点索引
    while (currentNode != NULL && currentNode->element != theElement)
    {
        currentNode = currentNode->next;
        index++;
    }
    if (currentNode == NULL) //未找到
```

```
        return -1;    //返回-1
    else    //找到
        return index;    //返回当前索引
}

/*删除索引元素*/
template <class T>
void chain<T>::Erase(int theIndex)
{
    trueindex = true;
    if (theIndex<0 || theIndex>=listsize)
    {
        cout << "wrong index" << endl;
        cout << "index = " << theIndex << " size = " << listsize <<
endl;
        trueindex = false;
    }          //检查元素
    if (trueindex)
    {
        chainNode<T> *deleteNode;
        if (theIndex == 0)
        {
            deleteNode = firstNode;
            firstNode = firstNode->next;
        }
        else
        {
            chainNode<T> *p = firstNode;
            for (int i=0; i<theIndex-1; i++)
                p = p->next;
            deleteNode = p->next;
            p->next = p->next->next;
        }
        listsize--;
        delete deleteNode;
    }

}

/*在索引处插入元素*/
template <class T>
void chain<T>::Insert(int theIndex, const T& theElement)
{
    trueindex = true;
```

```
    if (theIndex<0 || theIndex>listsize)
    {
        cout << "wrong index" << endl;
        cout << "index = " << theIndex << " size = " << listsize <<
endl;
        trueindex = false;
    }          //检查元素
    if (trueindex)
    {
        if (theIndex == 0)
            firstNode = new chainNode<T>(theElement, firstNode);
        else
        {
            chainNode<T> *p = firstNode;
            for (int i=0; i<theIndex-1; i++)
                p = p->next;
            p->next = new chainNode<T>(theElement,p->next);
        }
        listsize++;
    }
}

/*输出线性表*/
template <class T>
void chain<T>::Output() const
{
    for (chainNode<T> *currentNode = firstNode; currentNode != NULL;
currentNode = currentNode->next)
        cout << currentNode->element << " ";
}

/*倒序*/
template <class T>
void chain<T>::Reverse()
{
    if (listsize <= 1)
        return;
    chainNode<T> *t = NULL;   //尾节点
    chainNode<T> *p = firstNode;
    chainNode<T> *q = firstNode->next;
    while (q != NULL)
    {
        t = q->next;    //使尾节点指向 q->next
        q->next = p;
```

```
        p = q;      //q->next 前进 2 位，p 延后 1 位
        q = t;      //q,t 同时指向原 q->next,用以判断是否达到原尾节点
    }
    firstNode->next = NULL;  //设置倒序后的链表尾
    firstNode = p;      //设置倒序后的链表头
}

/*尾插入*/
template <class T>
void chain<T>::Push_back(const T& theElement)
{
    chainNode<T> *newNode = new chainNode<T>(theElement, NULL);
    if (firstNode == NULL)
        firstNode = lastNode = newNode;
    else
    {
        lastNode->next = newNode;
        lastNode = newNode;
    }
    listsize++;
}

/*合并两个链表*/
template <class T>
chain<T> Merge(chain<T>&a, chain<T>&b)
{
    chain<T> c;
    chain<int>::Iterator A = a.Begin();     //定义指向 a 的首节点的迭
代器 A
    chain<int>::Iterator B = b.Begin();     //定义指向 b 的首节点的迭
代器 B
    while (A!=a.End() && B!=b.End())
    {
        if (*A>*B)
        {
            c.Push_back(*B);
            B++;
        }
        else
        {
            c.Push_back(*A);
            A++;
        }
    }
```

```
    if (A!=a.End())
    {
        while (A!=a.End())
        {
            c.Push_back(*A);
            A++;
        }
    }
    if (B!=b.End())
    {
        while (B!=b.End())
        {
            c.Push_back(*B);
            B++;
        }
    }
    return c;
}

/*输入链表的数据*/
template <class T>
void Input(chain<T>&a)
{
    int num;    //链表的元素数量
    cout << "Input the number of this chain:" << endl;
    cin >> num;
    for (int i=0; i<num; i++)
    {
        cout << "Input No." << i+1 << ": " ;
        T temp;
        cin >> temp;
        a.Insert(a.Size(),temp);  //在链表的尾部插入键盘输入的数据
    }
    cout << "Input succeed!" << endl;
}

/*在链表中查询*/
template <class T>
void Search(chain<T>&a)
{
    T temp;
    bool sear = true;    //是否继续查询的判断依据
    while(sear)
    {
```

```
        cout << "Input what you want to search:";
        cin >> temp;
        int x = a.Find(temp);  //查询索引
        if (x==-1)    //未查询到，输出错误信息
        {
            cout << temp << " is No." << x << " in this chain" <<
endl;
            cout << temp << " has not been found!" << endl;
            cout << "Continue? 1. Yes 2. No" << endl;
            int choice;
            cin >> choice;
            if (choice==2)
            {
                sear = false;
                return;
            }
            else
                continue;
        }
        else    //查询到，输出查询元素的索引
        {
            cout << temp << " is No." << x << " in this chain" <<
endl;
            cout << "Continue? 1. Yes 2. No" << endl;
            int choice;
            cin >> choice;
            if (choice==2)
            {
                sear = false;
                return;
            }
            else
                continue;
        }
    }
}

/*在链表中删除*/
template <class T>
void Pop(chain<T>& a)
{
    T temp;
    bool poptrue = true;    //是否继续删除的判断依据
    while(poptrue)
```

```
    {
        cout << "Input what you want to pop:";
        cin >> temp;
        int x = a.Find(temp);  //查询需要删除的元素的索引值
        if (x==-1)
        {
            cout << temp << " has not been found!" << endl;
            cout << "Continue? 1. Yes 2. No" << endl;
            int choice;
            cin >> choice;
            if (choice==2)
            {
                poptrue = false;
                return;
            }
            else
                continue;
        }
        else
        {
            cout << temp << " is No." << x << " in this chain" <<
endl;
            a.Erase(x);                    //查询到了需要删除的元素的索
引，利用类内的函数 erase 进行删除
            cout << "succeed!" << endl;
            cout << "Continue? 1. Yes 2. No" << endl;
            int choice;
            cin >> choice;
            if (choice==2)
            {
                poptrue = false;
                return;
            }
            else
                continue;
        }
    }
}

int main()
{
    chain<int> a;   //第一个链表，用于验证插入、删除、搜索、输出
    chain<int> b;   //第二个链表，用于合并
    chain<int> c;   //保存合并而成的新链表
```

```
    Input(a);
    cout << "Chain A is: " << endl;
    a.Output();  //输入链表 A
    cout << endl;
    Search(a);  //在链表 A 中进行查询
    Pop(a);    //在链表 A 中进行删除
    a.Reverse();
    cout << "Reversed chain A is: " << endl;
    a.Output();   //输出反序之后的链表 A
    cout << endl;
    Input(b);    //输入链表 B
    cout << "Chain B is: " << endl;
    b.Output();   //输出链表 B
    cout << endl;
    c = Merge(a,b);   //将链表 A、B 合并，并将结果存储在链表 C 中
    cout << "Chain C (merge A and B) is: " << endl;
    c.Output();     //输出链表 C
    cout << endl;
    return 0;
}
```