数据结构实验报告

1. 实验序号与名称

实验 2.2：不带头结点形式的单链表。

2. 学号，姓名，专业，实验时间

学号：2016141223037　　姓名：宋运翔　　专业：计算金融（方向）　　实验时间：第 5~7 周。

3. 实验内容与目标

实现不带头结点形式的单链表的初始化，计算长度，插入，删除等功能。培养实践能力，并熟悉线性表，培养设计数据结构与算法的能力。

4．实验工具

Microsoft Visual Studio 2013

5.实验分析

5.1 算法基本思想

一个线性表的不带头结点的单链表结构通常如图 5.1 所示，当单链表中没有数据元素时，这是便没有头节点，也就是说 first==NULL。

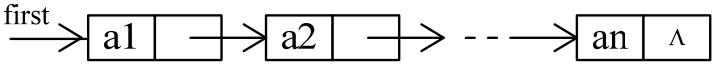


图 5.1 不带头结点的单链表结构示意图

对于该单链表，定义类 NOHEAD，类的成员与功能如表 5.1 所示。

表 5.1 NOHEAD 类的成员和功能

类型	成员	功能
数据成员	Node<ElemType>* first	指向首元素结构的指针
辅助函数	Node<ElemType>*GetElemptr(int position) const	返回第 position 个节点的指针
	void Init()	初始化线性表

方法	int Length() const	求线性表长度
	bool Empty() const	判断线性表是否为空
	void Clear()	将线性表清空
	void Traverse(void(*Visit)(ElemType&))	遍历线性表
	StatusCode GetElem(int position, ElemType&e) const	求指定位置的元素
	StatusCode SetElem(int position, const ElemType&e)	设置指定位置的元素
	StatusCode Delete(int position, ElemType&e)	删除元素
	StatusCode Insert(int position, const ElemType&e)	插入元素

5.2 成员函数代码

对于不带头结点的单链表，第一个元素的前驱为为 NULL，其他元素则有非空的前驱，对于插入，删除等操作都需要对第一个元素单独进行讨论。

5.2.1 GetElemptr(int position)

```
Node<ElemType>* pt=first;
        int curposition = 1;          //pt 所指结点的位置
        while (pt != NULL && curposition < position)
        {    pt = pt->next;
             curposition++;
        }

        if (pt != NULL && curposition == position)
        {
             return pt;
        }

        else
        {
             retuen NULL;
        }
```

5.2.2 Init()

```
first = NULL;
```

5.2.3 Length()

```
int count = 0;//初始化统计元素个数
        for (Node<ElemType>*pt = first; pt != NULL; pt = pt->next)//对线性表的每个元素进行计数
        {
            count++;
        }
        return count;
```

5.2.4 Empty()

```
return first == NULL;
```

5.2.5 Clear()

```
ElemType a;
        while (Length()>0)
        {
            Delete(1,a)
        }
```

5.2.6 Traverse(void(*Visit)(ElemType&))

```
for (Node<ElemType>*pt = first; pt != NULL; pt = pt->next)
        {
            (*Visit)(pt->data);
        }
```

5.2.7 GetElem(int position, ElemType&e)

```
if (position<1 || position>Length())
        {
            return RANGE_ERROR;
        }
        else
        {
            Node<ElemType>*pt;
            pt = GetElemptr(position);
```

```
        e = pt->date;
        return ENTRY_FOUND;
    }
```

5.2.8 SetElem(int position, const ElemType&e)

```
if (position<1 || position>Length())
    {
        return RANGE_ERROR;
    }
    else
    {

        Node<ElemType>*pt;
        pt = GetElemptr(position);
        pt->date = e;
        return ENTRY_FOUND;
    }
```

5.2.9 Delete(int position, ElemType&e)

```
if (position<1 || position>Length())
    {
        return RANGE_ERROR;
    }
    else
    {

        Node<ElemType>*pt;
        if (position > 1)//删除非第 1 个元素
        {
            pt = GetElemptr(position - 1);
            Node<ElemType>*pm = pt->next;
            pt->next = pm->next;
            e = pm->data;
            delete pm;
        }

        else//删除第 1 个元素
        {
```

```
            pt = first;//暂存 first
            first = first->next;//first 指向后继
            delete pt;
        }

        return SUCCESS;
    }
```

5.2.10 Insert(int position, const ElemType&e)

```
if (position<1 || position>Length())
    {
        return RANGE_ERROR;
    }
    else
    {

        Node<ElemType>*pm;//指向被插入结点
        if (position > 1）//插入元素不是第 1 个元素
        {
            Node<ElemType>*pt = GetElemptr(position - 1);
            pt = GetElemptr(position - 1);
            pm =new Node<ElemType>(e,pt->next);
            pt->next = pm;
        }

        else//插入元素是第 1 个元素
        {
            pm = new Node<ElemType>(e,first);//生成新结点
            first = pm;//pm 是新的第 1 个元素的结点
            delete pt;
        }

        return SUCCESS;
    }
```

5.3 主函数流程图

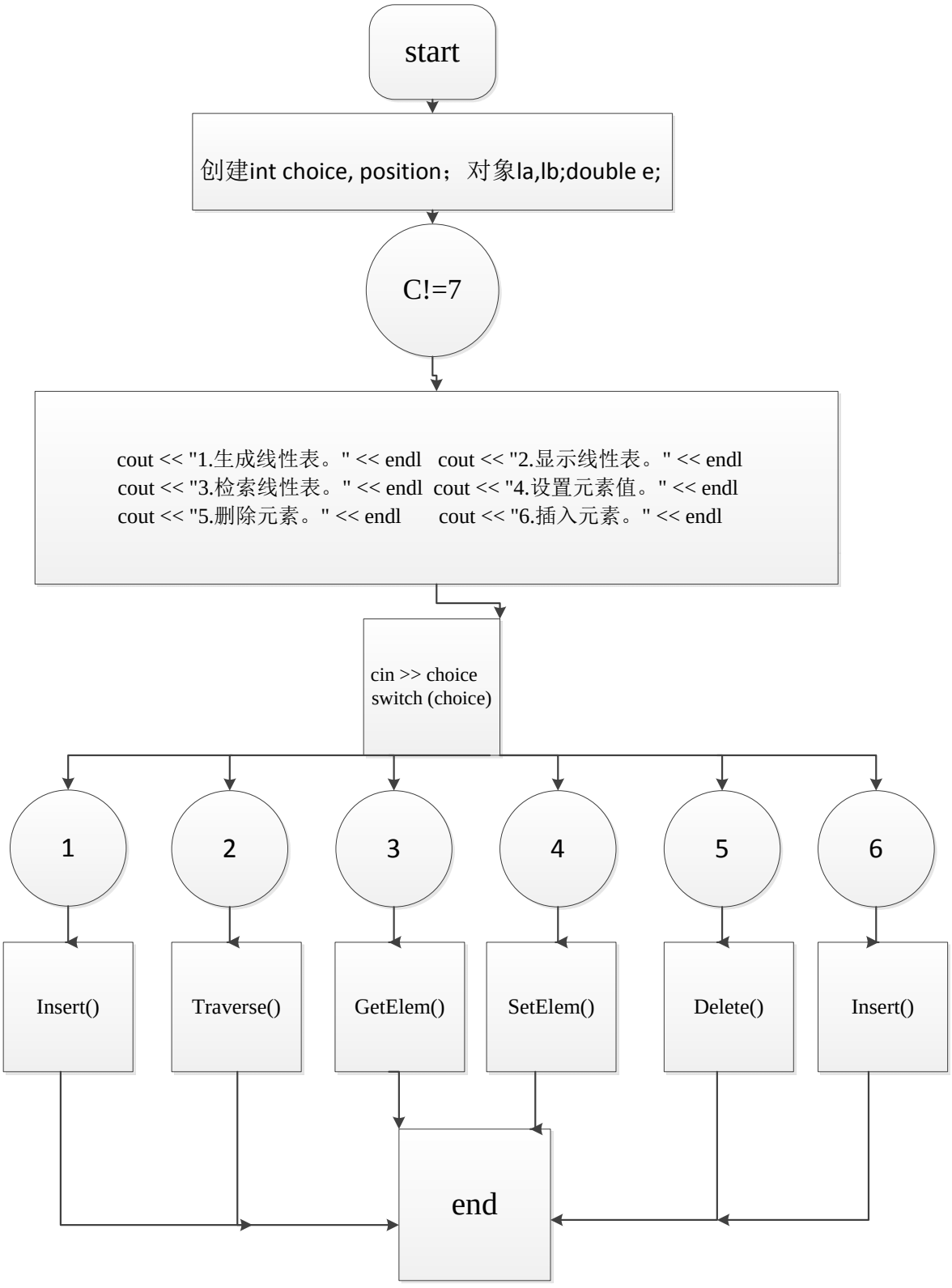


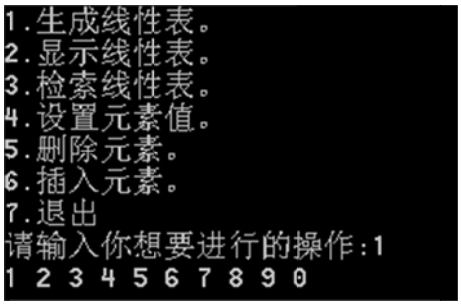
图 5. 2 主函数流程图

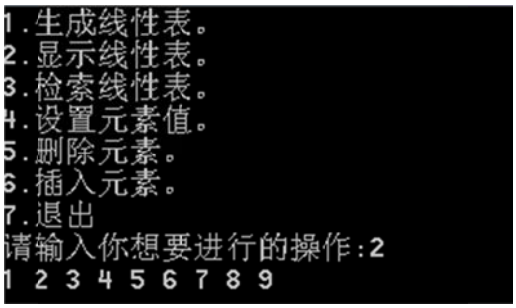
6. 实验步骤

1.建立项目 Simple_list_without_head_node。
2.将软件包中的 utility.h 和 node.h 复制到 Simple_list_without_head_node 文件夹中，

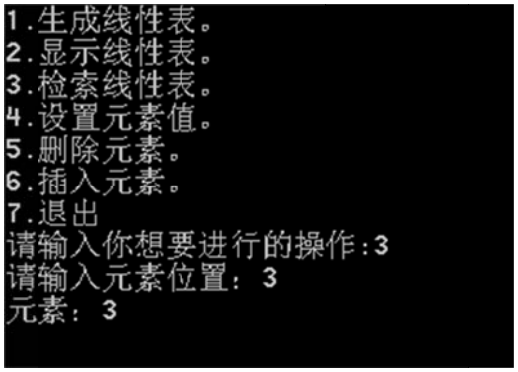
并将 utility.h 和 node.h 加入到项目中。
3.建立头文件 Simple_list_without_head_node.h, 声明不带头结点的单链表类，并实现相关的成员函数。建立源程序文件 main.cpp, 实现 main()函数。
4. 建立源程序文件 main.cpp, 实现 main()函数。

7.测试与结论
7.1 生成并显示线性表

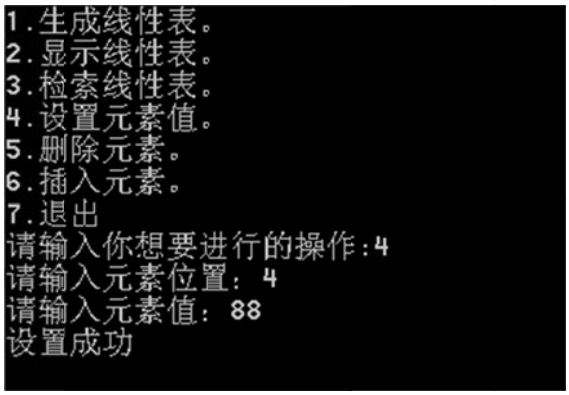




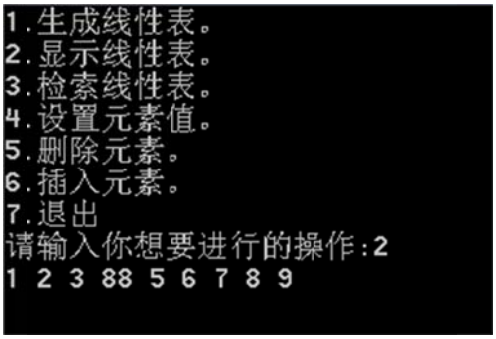
7. 2 检查模拟元素



7. 3 设置元素值

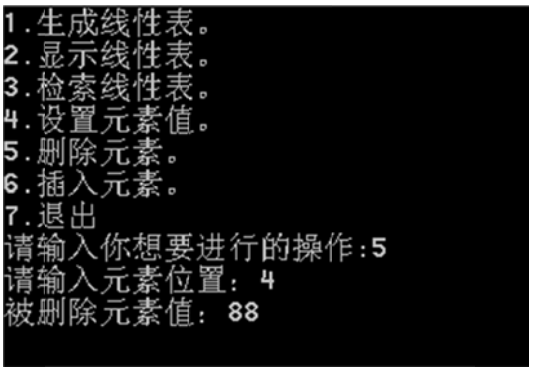


之后通过显示线性表，检查是否设置成功。

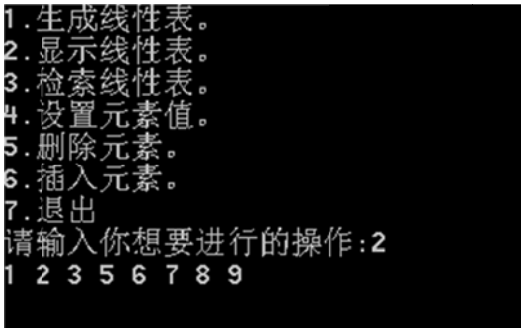


元素“4”被替换为“88”，设置成功。

7.4 删除元素

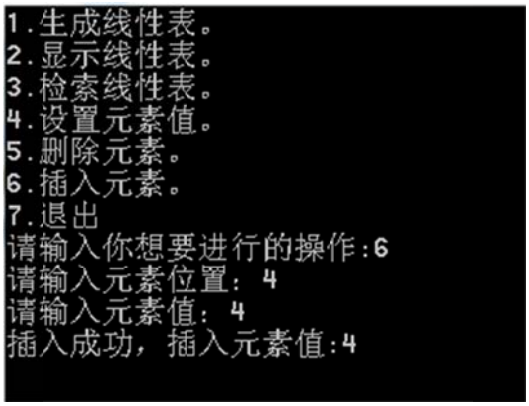


之后通过显示线性表，检查是否设置成功。

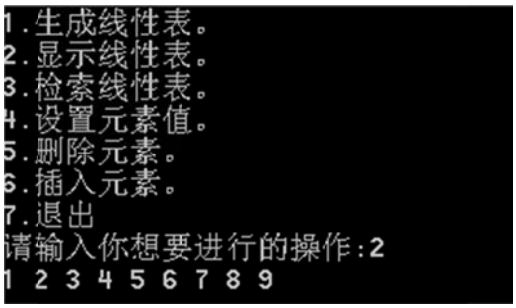


元素“88”被删除。

7.5 插入元素



之后通过显示线性表，检查是否设置成功。



元素“4”插入成功。

8. 思考与感悟

该算法是一种简单的实现方式，但是算法的效率很低，可以通过在不带头结点的单链表结构中保存当前位置和元素个数以提高算法效率。

这次编程任务完成地较为艰辛，但做完之后大大加深了自己对书中各个知识点的印象和理解。也学会了一些编写算法的小技巧，要有耐心，多看书复习知识。总之，这次实验使我印象深刻。