实验四

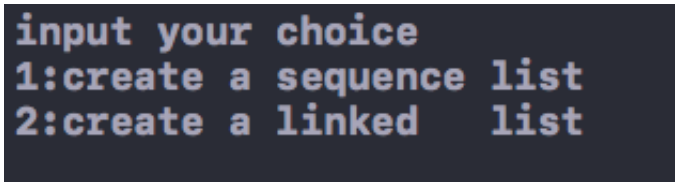
一、实验内容

线性表的顺序存储结构和链接存储结构的建立、插入、删除运算

二、实验操作

关于选单：

主函数通过字符选单测试数据：

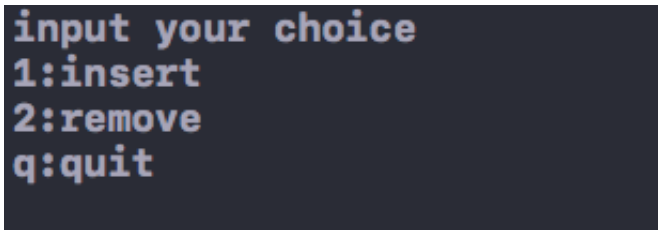


进入分离操作函数：

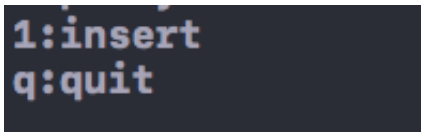
```
void sequence_implementation()

void linked_implementation()
```

内部选单：



当list为空时（无remove操作）：



关于insert：

```
Error_code s_List::insert(int position,const char &x)//sequence_list

Error_code l_List::insert(int position,const char &x)//linked_list
```

首先输入insert的个数，上界为10

然后是每个entry的位置和内容，内容必须为字母

最后会显示当前list的结果

```
Input the number of entry you want to insert
3
Input the position and the entry you want to insert
The entry must be a letter between a to z or A to Z!!!
0
a
You have done
The list is:a
```

positon范围为0-count，否则重新输入

```
Attention to your position value or entry type
Please input again
0
1
Attention to your position value or entry type
Please input again
```

关于remove:

```
Error_code s_List::remove(int position,char &x)//sequence_list

Error_code l_List::remove(int position,char &x)//linked_list
```

输入的remove的个数不得大于size

positon的范围为0-count-1

```
Input the number of entry you want to remove
4
the number is bigger than the size of list,please input again
3
Input the position you want to remove
0
You have deleted a
The list is : b c
```

三、实验关键算法

顺序存储结构数据成员：

```
private:
    int count;

    char entry[max_list];
```

链式存储结构数据成员：

```
private:
    int count;
    Node* head;
    Node* set_position(int position)const;
```

insert

```
Error_code s_List::insert(int position,const char &x)
{
    if(!isalpha(x))
        return fail;
    if(full())
        return fail;
    if(position<0||position>count)
        return fail;
    for(int i=count-1;i>=position;i--){
        entry[i+1]=entry[i];
    }
    entry[position]=x;
    count++;
    return success;
}
```

```
Error_code l_List::insert(int position, const char &x)
{
    if(!isalpha(x))
        return fail;
    if (position < 0 || position > count)
        return fail;
    Node* new_node, * previous=nullptr, * following;
    if (position > 0) {
        previous = set_position(position - 1);
        following = previous->next;
    }
    else following = head;
    new_node = new Node(x, following);
    if (new_node == nullptr)
        return fail;
    if (position == 0)
        head = new_node;
    else{
        previous->next = new_node;
    }
    count++;
    return success;
}
```

```
Error_code s_List::remove(int position,char &x){
    if(empty())
        return  fail;
    if(position<0||position>count-1)
        return fail;
    x=entry[position];
    count--;
    while(position<count){
        entry[position]=entry[position+1];
        position++;
    }
    return success;
}
```

remove

```
Error_code s_List::remove(int position,char &x){
    if(empty())
        return  fail;
    if(position<0||position>count-1)
        return fail;
    x=entry[position];
    count--;
    while(position<count){
        entry[position]=entry[position+1];
        position++;
    }
    return success;
}
```

```
Error_code l_List::remove(int position,char &x){

    Node *pre,*cur;
    if(count==0)
        return fail;
    if(position<0||position>count-1)
        return fail;
    if(position>0){
        pre=set_position(position-1);
        cur=pre->next;
        pre->next=cur->next;
    }
    else{
        cur=head;
        head=cur->next;
    }
    x=cur->entry;
    delete cur;
    cur=nullptr;
    count--;
    return success;
}
```