实验一：线性表的物理实现

计科 1903 陈旭 201914020128

日期：2020.10.13

当天任务：**实现一个顺序表。**

产生问题：为什么在基类中需要用纯虚函数来声明

解决办法：多重继承可能会在构造的时候，同一个基类被构造多次。

产生问题：void clear() 函数在实现时，起初考虑使用 for 循环遍历顺序表，使用 delete 删除每一个元素，并将当前位置赋值 0，线性表元素个数赋值为 0。此方法时间复杂度太高，并且可读性很差。

解决办法：直接用 delete 删除原有线性表，再重新调用构造函数建个新的线性表。

当日心得：顺序表的本质就是一个数组，与其封装在一起的还有一些对其元素的操作。但操作的时间复杂度、空间复杂度直接决定了该种数据结构的效率与实用性。日常应多做算法设计练习，确保思维长期保持敏捷，增加顺序表的效率。

日期：202010.15

当天任务：**实现一个单向链表。**

产生问题：

纯虚函数 E getValue() 未在子类中被实现。
具体代码:

```
E& getvalue ()
{
    if (curr==tail) return;
    return curr->next->element;
}
```

基类中的声明:

```
virtual const E& getvalue() = 0; //返回当前元素值
```

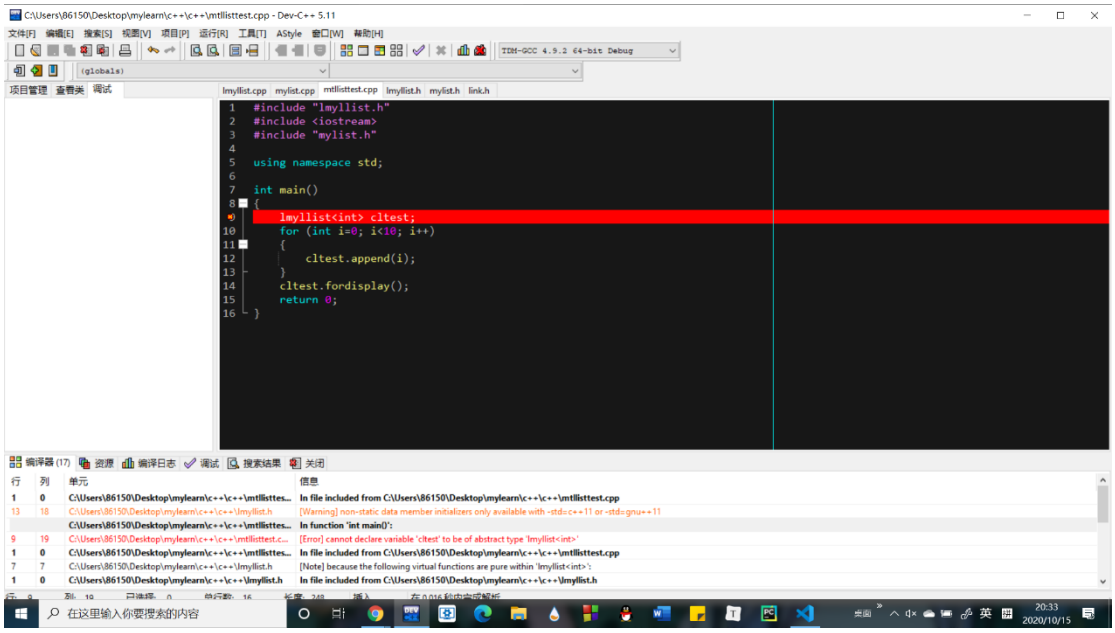
解决方式: 第一行加 const (声明时必须加 const)。

产生问题: 实现之后，增加了一个函数用来方便调试:

输出链表中每一个元素的值。

```
void fordisplay ()
{
    assert(cnt > 0);
    link<E> *ltemp;
    ltemp=curr;
    curr=head;
    while (curr!=NULL)
    {
        cout << curr->element << ' ';
        curr=curr->next;
    }
    cout << endl;
    curr=ltemp;
    delete ltemp;
}
```

在调用中产生了很多不可名状的错误，如溢出，空指针报错以及下图几个错误。



代码如下：

```
void init()
{
    curr=new link<E>;
    tail=new link<E>;
    head==new link<E>;
    cnt=0;
}
```

```
void append(const E& it)
{
    if (cnt==0)
    {
        head->next->element=it;
        curr=curr->next;
        tail=head->next;
        cnt++;
    }
    else
    {
        tail=tail->next=new link<E>(it, NULL);
        cnt++;
    }
```

最终调试了很久，得出结论。

解决方法:

```
void init()
{
    tail=head=curr=new link<E>;//用于被构造函数调用
    cnt=0;
}
```

```
void append(const E& it)
{
    tail = tail->next = new link<E>(it, NULL);
    cnt++;
}
```

init 函数中构造的的三个指针不能按照一开始的方法定义，这样指示的是三个新对象。

append 函数(尾插函数)使用了多余的判断，增加了不必要的复杂度。

产生问题：在链表实现完成后，对照老师的代码，发现以下部分不相同:

```
LList(int size=100) { init(); }
```

产生疑问，链表使用的是动态数组来存放数据的，为什么在构造函数里声明它的最大尺寸，而且在之后的实现文件里未再见 size 被使用过。

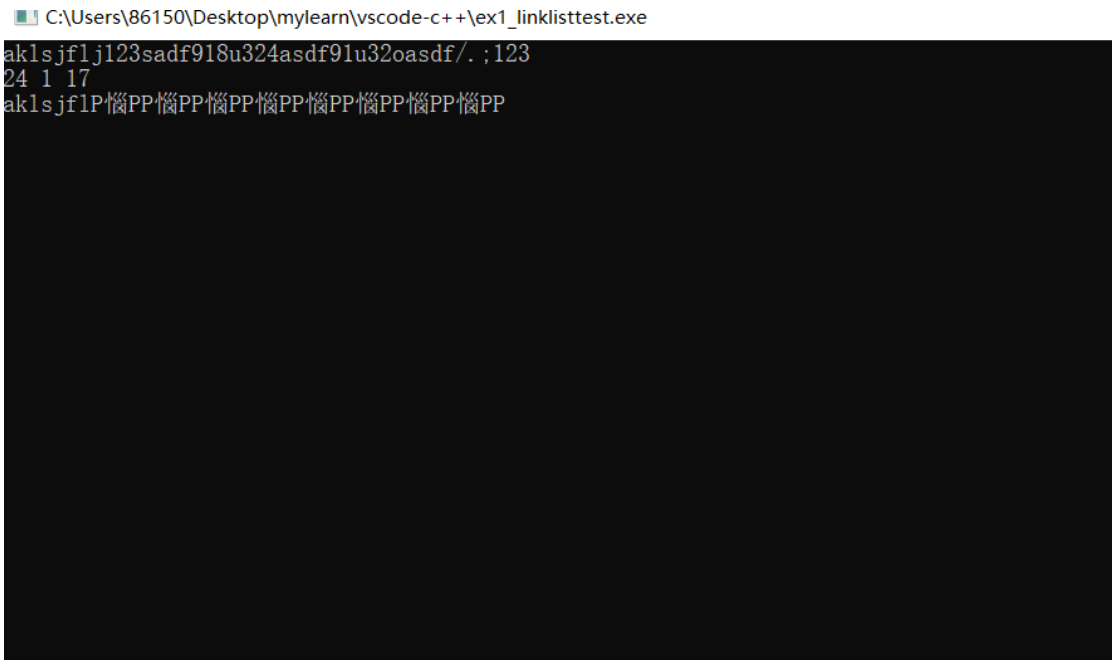
问题解答: size 作为一个默认的大小，如果存储数据量比较少，则使用 size 作为最大尺寸，防止造成内存的浪费。

当日心得：今天碰到很多问题，调试起来非常痛苦，但是最终成功实现之后觉得收获很多，今天的问题中大部分是粗心造成的，比如前两个问题，并且这两个问题也是修改用时最长的。数据结构的学习中必须注意时刻保持细心

日期： 202010.17-202010.21

当天任务: 完成实验内容，检验链表的是否可以实现

产生问题： 实现时输入目标字符串后，输出了一系列乱码字符。



而调用自主定义的 fordisplay 函数逐块测试输出后锁定 prev 函数(取前驱元素)出现问题，下是其代码。

```
void prev()
{
    if (curr==head)
    {
        return;
    }
    link<E> *ltemp=curr;
    curr=head;
    while (curr->next!=ltemp)
    {
        curr=curr->next;
    }
    delete ltemp;
}
```

问题解答: 经过查找资料和请教同学, 得知最后一行 delete 操作删除了中间的指针变量。

这一步骤错误。ltemp 只是一个指针, 没有开辟一个 link<E>的空间，删除操作删掉的就是 ltemp 指向的内容，所以造成了链表元素的丢失。并且一个地址不能 delete 两次，必须 new 出来的东西才能 delete,删完后那个地方就成了垃圾地址，所以产生了乱码。

修改后的代码:

```
void prev()
{
    if (curr==head)
    {
```

```
            return;
        }
        link<E> *ltemp=curr;
        curr=head;
        while (curr->next!=ltemp)
        {
            curr=curr->next;
        }
        //delete ltemp; /*把他注释掉*/
    }
```

之后可以正常的输出。

当日心得：当 delete 操作的对象是指针时，其代表的意义是删除指针指向的对象的内容。C\C++的内存管理是 C++中不可忽略的一部分内容，并且要时刻保持警惕。