一、将下列两个程序分别在VC++中调试出来，要求将顺序表的数据改为键盘输入，并附上调试好的源程序及运行结果(截图）。

1、

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    对应教材2.3.3节，顺表类SeqList的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

const int MaxSize = 100;            //100只是示例性的数据，根据实际问题具体定义

template <class DataType>          //定义模板类SeqList

class SeqList

{

public:

       SeqList( );                     //无参构造函数，建立空的顺序表

       SeqList(DataType a[ ], int n);      //有参构造函数，建立长度为n的顺序表

       ~SeqList( );                    //析构函数

       int Length( );                   //求线性表的长度

       int Empety();

       DataType Get(int i);              //按位查找，查找第i个元素的值

       int Locate(DataType x );          //按值查找，查找值为x的元素序号

       void Insert(int i, DataType x);      //插入操作，在第i个位置插入值为x的元素

       DataType Delete(int i);            //删除操作，删除第i个元素

       void PrintList( );                 //遍历操作，按序号依次输出各元素

private:

       DataType data[MaxSize];          //存放数据元素的数组

       int length;                       //线性表的长度

};

template<class DataType>

SeqList<DataType> :: ~SeqList()

{

}

template <class DataType>

SeqList<DataType> :: SeqList()

{

    length = 0;

}

template <class DataType>

int SeqList<DataType> :: Empety()

{

    if(length == 0)

        return 1;

    else

        return 0;

}

template <class DataType>

int SeqList<DataType> :: Length()

{

    return length;

}

template <class DataType>

SeqList<DataType> :: SeqList(DataType a[ ], int n)

{

    if (n > MaxSize)

        throw "参数非法";

    for (int i = 0; i < n; i++)

        data[i] = a[i];

    length = n;

}

template <class DataType>

void SeqList<DataType> :: PrintList( )

{

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

        cout << data[i];                   //依次输出线性表的元素值

}

template <class DataType>

int SeqList<DataType> :: Locate(DataType x)

{

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

        if (data[i] == x) return i+1;        //返回其序号i+1

       return 0;                         //退出循环，说明查找失败

}

template <class DataType>

DataType SeqList<DataType> :: Get(int i)

{

    if (i < 1 && i > length)

        throw "查找位置非法";

    else

        return data[i - 1];

}

template <class DataType>

DataType SeqList<DataType> :: Delete(int i)

{

    if (length == 0)

        throw "下溢";

    if (i < 1 || i > length)

        throw "位置";

    int x = data[i - 1];                  //取出位置i的元素

    for (int j = i; j < length; j++)

        data[j - 1] = data[j];           //此处j已经是元素所在的数组下标

    length--;

    return x;

}

template <class DataType>

void SeqList<DataType> :: Insert(int i, DataType x)

{

      if (length >= MaxSize)

          throw "上溢";

      if (i < 1 || i > length + 1)

          throw "位置";

      for (int j = length; j >= i; j--)

        data[j] = data[j - 1];              //第j个元素存在数组下标为j-1处

      data[i - 1] = x;

      length++;

}

int main( )

{

    int r[5] = {1, 2, 3, 4, 5}, i, x;

       SeqList<int> L{r, 5};                  //建立具有5个元素的顺序表

       cout << "当前线性表的数据为：";

       L.PrintList( );                        //输出当前线性表1 2 3 4 5

       try

       {

           L.Insert(2, 8);                        //在第2个位置插入值为8的元素

           cout << endl << "执行插入操作后数据为：";

           L.PrintList( );                        //输出插入后的线性表1 8 2 3 4 5

           cout << endl;

       }catch(char\* str){

           cout << str << "插入操作错误！" << endl;

       }

       cout << "当前线性表的长度为：" << L.Length( );    //输出线性表的长度6

       cout << endl;

       cout << "请输入查找的元素值：";

       cin >> x;

       i = L.Locate(x);

       if (0 == i) cout << "查找失败" << endl;

       else cout << "元素" << x << "的位置为：" << i << endl;

       try

       {

           cout << "请输入查找第几个元素值：";

           cin >> i;

           cout << "第" << i << "个元素值是" << L.Get(i) << endl;

       }catch(char\* str){

           cout << "线性表中没有该元素" << endl;

       }

       try

       {

        cout << "请输入要删除第几个元素：";

           cin >> i;

           x = L.Delete(i);                              //删除第i个元素

           cout << "删除的元素是" << x <<"，删除后数据为：";

              L.PrintList( );                           //输出删除后的线性表

       }catch(char\* str){

           cout << "删除错误！" << endl;

       }

    return 0;

}

2、

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    对应教材2.4.2节, 单链表类LinkList的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>                  //引入输入输出流

using namespace std;

template <typename DataType>

struct Node

{

    DataType data;               //数据域

    Node<DataType> \*next;       //指针域

};

template <typename DataType>

class LinkList

{

public:

    LinkList( );                      //无参构造函数，建立只有头结点的空链表

    LinkList(DataType a[ ], int n);       //有参构造函数，建立有n个元素的单链表

    ~LinkList( );                     //析构函数

    int Length( );                     //求单链表的长度

    int Empety();

    DataType Get(int i);               //按位查找。查找第i个结点的元素值

    int Locate(DataType x);            //按值查找。查找值为x的元素序号

    void Insert(int i, DataType x);       //插入操作，第i个位置插入值为x的结点

    DataType Delete(int i);            //删除操作，删除第i个结点

    void PrintList( );                  //遍历操作，按序号依次输出各元素

private:

    Node<DataType> \*first;           //单链表的头指针

};

template <typename DataType>

LinkList<DataType> :: LinkList( )

{

    first = new Node<DataType>;              //生成头结点

      first->next = nullptr;                      //头结点的指针域置空

}

template <class DataType>

LinkList<DataType> :: ~LinkList( )

{

      Node<DataType> \*q = NULL;

      while (first != NULL)        //释放单链表的每一个结点的存储空间

      {

        q = first;                 //暂存被释放结点

        first = first->next;         // first指向被释放结点的下一个结点

        delete q;

     }

}

template <typename DataType>

int LinkList<DataType> :: Empety()

{

    if(first->next == nullptr)

        return 1;

    else

        return 0;

}

template <typename DataType>

void LinkList<DataType> :: PrintList( )

{

    Node<DataType> \*p = first->next;                //工作指针p初始化

      while (p != nullptr)

      {

        cout << p->data << "\t";

        p = p->next;                 //工作指针p后移，注意不能写作p++

      }

}

template <typename DataType>

int LinkList<DataType> :: Length( )

{

       Node<DataType> \*p = first->next;   //工作指针p初始化为开始接点

       int count = 0;                    //累加器count初始化

       while (p != nullptr)

       {

        p = p->next;

        count++;

       }

    return count;              //注意count的初始化和返回值之间的关系

}

template <typename DataType>

DataType LinkList<DataType> :: Get(int i)

{

    Node<DataType> \*p = first->next;    //工作指针p初始化

    int count = 1;                     //累加器count初始化

      while (p != nullptr && count < i)

      {

        p = p->next;                   //工作指针p后移

        count++;

    }

    if (p == nullptr) throw "位置";

    else return p->data;

}

template <typename DataType>

int LinkList<DataType> :: Locate(DataType x)

{

    Node<DataType> \*p = first->next;   //工作指针p初始化

      int count = 1;                     //累加器count初始化

      while (p != nullptr)

      {

          if (p->data == x) return count;     //查找成功，结束函数并返回序号

          p = p->next;

          count++;

      }

      return 0;                        //退出循环表明查找失败

}

template <typename DataType>

void LinkList<DataType> :: Insert(int i, DataType x)

{

    Node<DataType> \*p = first, \*s = nullptr ;        //工作指针p初始化

    int count = 0;

    while (p != nullptr && count < i - 1)            //查找第i – 1个结点

    {

        p = p->next;                              //工作指针p后移

          count++;

    }

    if (p == nullptr) throw "位置";        //没有找到第i – 1个结点

    else {

          s = new Node<DataType>; s->data = x;      //申请结点s，数据域为x

          s->next = p->next; p->next = s;     //将结点s插入到结点p之后

    }

 }

// 头插法构造

//template <typename DataType>

//LinkList<DataType> :: LinkList(DataType a[ ], int n)

//{

//    first = new Node<DataType>; first->next = nullptr;     //初始化一个空链表

//    for (int i = 0; i < n; i++)

//    {

//        Node<DataType> \*s;

//        s = new Node<DataType>; s->data = a[i];

//         s->next = first->next; first->next = s;    //将结点s插入到头结点之后

//    }

//}

template <typename DataType>

LinkList<DataType> :: LinkList(DataType a[ ], int n)

{

      first = new Node<DataType>;                    //生成头结点

      Node<DataType> \*r = first, \*s = nullptr;           //尾指针初始化

      for (int i = 0; i < n; i++)

      {

          s = new Node<DataType>; s->data = a[i];

          r->next = s; r = s;                 //将结点s插入到终端结点之后

      }

    r->next = nullptr;        //单链表建立完毕，将终端结点的指针域置空

}

template <typename DataType>

DataType LinkList<DataType> :: Delete(int i)

{

    DataType x;

      Node<DataType> \*p = first, \*q = nullptr;        //工作指针p指向头结点

      int count = 0;

      while (p != nullptr && count < i - 1)           //查找第i-1个结点

      {

        p = p->next;

        count++;

      }

      if (p == nullptr || p->next == nullptr)  //结点p不存在或p的后继结点不存在

          throw "位置";

      else {

           q = p->next; x = q->data;         //暂存被删结点

           p->next = q->next;              //摘链

           delete q;

           return x;

      }

}

int main( )

{

    int r[5] = {1, 2, 3, 4, 5}, i, x;

      LinkList<int> L{r, 5};

      cout << "当前线性表的数据为：";

      L.PrintList( );                         //输出当前链表1 2 3 4 5

      cout << endl;

      try

      {

        L.Insert(2, 8);                         //在第2个位置插入值为8的结点

        cout << "执行插入操作后数据为：";

        L.PrintList( );                         //输出插入后链表1 8 2 3 4 5

        cout << endl;

      } catch(char\* str){

          cout << str << endl;

      }

    cout << "当前单链表的长度为：" << L.Length( ) << endl;      //输出单链表长度6

    cout << "请输入查找的元素值：";

    cin >> x;

    i = L.Locate(x);

    if (1 <= i) cout << "元素" << x << "的元素位置为：" << i << endl;

    else cout << "单链表中没有元素" << x << endl;

    try

    {

        cout << "请输入要删除第几个元素：";

        cin >> i;

        x = L.Delete(i);                                 //删除第i个元素

        cout << "删除的元素值是" << x << "执行删除操作后数据为：";

        L.PrintList( );                                 //输出删除后链表

    } catch(char\* str) {

          cout << str << endl;

  }

  return 0;

}