# 线性表

## 源代码

|  |  |
| --- | --- |
| 顺序表 |  |
| 单链表 |  |

### 手机查看

#### 顺序表

**SeqList.h**

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  const int MaxSize = 200; // 设置顺序表最多能存储的元素的数量  template <typename DataType = int> // 相当于Java的泛型，在C++叫模板  class SeqList  {  public:  SeqList(); // 无参构造，建立空的顺序表  SeqList(DataType a[], int n); // 有参构造，建立长度为n的顺序表  ~SeqList(); // 析构函数，就是在对象销毁时会被执行  int Length(); // 获取线性表的长度  DataType Get(int i); // 按位查找，查找第i个元素  int Locate(DataType x); // 按值查找，查找值为x的元素  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  int Empty(); // 判断线性表是否为空  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  };  // 无参构造，建立空的顺序表  template <typename DataType>  SeqList<DataType>::SeqList() {  this->length = 0;  };  // 有参构造，建立长度为n的顺序表  template <typename DataType>  SeqList<DataType>::SeqList(DataType a[], int n) {  if (n > MaxSize) throw "参数非法";  for (int i = 0; i < n; i++) {  this->data[i] = a[i];  }  this->length = n;  }  // 析构函数，就是在对象销毁时会被执行  template <typename DataType>  SeqList<DataType>::~SeqList() {  cout << "被销毁了" << endl;  }  // 获取线性表的长度  template <typename DataType>  int SeqList<DataType>::Length() {  return this->length;  }  // 按位查找，查找第i个元素  template <typename DataType>  DataType SeqList<DataType>::Get(int i) {  if (i < 1 || i > this->length) {  throw "位置不存在";  }  return this->data[i - 1];  }  // 按值查找，查找值为x的元素  // 0：没找到  template <typename DataType>  int SeqList<DataType>::Locate(DataType x) {  for (int i = 0; i < length; i++) {  if (this->data[i] == x) {  return i + 1;  }  }  return 0;  }  // 插入操作，在第i个位置插入值为x的元素  template <typename DataType>  void SeqList<DataType>::Insert(int i, DataType x) {  if (length >= MaxSize) throw "已满";  if (i < 1 || i > length + 1) throw "位置错误";  for (int j = length; j >= i - 1; j--) {  this->data[j + 1] = this->data[j];  }  this->data[i - 1] = x;  this->length++;  }  // 删除操作，删除第i个元素  template <typename DataType>  DataType SeqList<DataType>::Delete(int i) {  if (i < 1 || i > length) throw "位置错误";  DataType del = this->Get(i);  for (int j = i-1; j < length - 1; j++) {  this->data[j] = this->data[j + 1];  }  this->length--;  return del;  }  // 判断线性表是否为空  template <typename DataType>  int SeqList<DataType>::Empty() {  return this->length ? 0 : 1;  }  // 遍历操作，按序号依次输出各元素  template <typename DataType>  void SeqList<DataType>::PrintList() {  if (this->Empty()) {  cout << "{}" << endl;  return;  }  int i = 0;  cout << "{";  for (; i < length - 1; i++) {  cout << this->data[i] << ",";  }  cout << this->data[i] << "}" << endl;  } |

**Main.cpp**

|  |
| --- |
| #include <iostream>  #include "SeqList.h"  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, SeqList<>\* list);  // 清除控制台（未使用）  void Clear();  int main() {  SeqList<> list;  int choose = -1;  while (choose) {  cout << "--------------------------------" << endl;  cout << "元素[" << list.Length() << "]个|最大容量[" << MaxSize << "]个|顺序表状态";  list.PrintList();  cout << "1、插入一个数字" << endl;  cout << "2、指定位置插入一个数字" << endl;  cout << "3、获取某个数的具体位置" << endl;  cout << "4、获取一个具体位置的数" << endl;  cout << "5、删除一个指定位置的数" << endl;  cout << "0、退出演示" << endl;  cout << "请输入对应服务编号：";  cin >> choose;  try {  Menu(choose, &list);  }  catch (const char \* e) {  cout << "出错啦，错误消息：" << e << endl;  }  cout << endl << "--------------------------------" << endl;  }  }  // 菜单  template <typename DataType = int>  void Menu(int choose, SeqList<>\* list) {  int in\_1 = 0, in\_2 = 0, temp\_1 = 0;  switch (choose) {  // 插入一个数字  case 1:  cout << "请输入要插入的数字：";  cin >> in\_1;  list->Insert(list->Length() + 1, in\_1);  break;  // 指定位置插入一个数字  case 2:  cout << "请输入指定位置与要插入的数字：";  cin >> in\_1 >> in\_2;  list->Insert(in\_1, in\_2);  break;  // 获取某个数的具体位置  case 3:  cout << "请输入要查询的数：";  cin >> in\_1;  temp\_1 = list->Locate(in\_1);  if (temp\_1) {  cout << "所查数第一次出现的位置是：" << temp\_1 << endl;  }  else {  cout << "没有查到哦~" << endl;  }  break;  // 获取一个具体位置的数  case 4:  cout << "请输入要查找的位置：";  cin >> in\_1;  temp\_1 = list->Get(in\_1);  cout << "所在位置的数是：" << temp\_1 << endl;  break;  // 删除一个指定位置的数  case 5:  cout << "请输入要删除的位置：";  cin >> in\_1;  list->Delete(in\_1);  break;  case 0:  cout << "感谢体验，再见~" << endl;  break;  default:  cout << "输入有误" << endl;  }  }  // 清除屏幕（未使用）  void Clear()  {  cout << "\x1B[2J\x1B[H";  } |

#### 单链表

**LinkList.h**

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  // 定义单链表的结点  template <typename DataType = int>  struct Node {  DataType data;  Node<DataType>\* next;  };  template <typename DataType = int> // 相当于Java的泛型，在C++叫模板  class LinkList  {  public:  LinkList(); // 无参构造，建立空的顺序表  LinkList(DataType a[], int n); // 有参构造，建立长度为n的顺序表  ~LinkList(); // 析构函数，就是在对象销毁时会被执行  int Length(); // 获取线性表的长度  DataType Get(int i); // 按位查找，查找第i个结点  int Locate(DataType x); // 按值查找，查找值为x的结点  void Insert(int i, DataType x); // 插入操作，在第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  int Empty(); // 判断线性表是否为空  void PrintList(); // 遍历操作，按序号依次输出各结点  private:  Node<DataType>\* header; // 单链表的头结点  };  // 无参构造，建立只有头节点的单链表  template <typename DataType>  LinkList<DataType>::LinkList() {  this->header = new Node<DataType>;  this->header->next = nullptr;  }  // 有参构造，建立长度为n的顺序表，头插法  template <typename DataType>  LinkList<DataType>::LinkList(DataType a[], int n) {  this->header = new Node<DataType>;  this->header->next = nullptr;  Node<DataType>\* temp = nullptr;  for (int i = 0; i < n; i++) {  temp = this->header->next;  this->header->next = new Node<DataType>;  this->header->next->data = a[i];  this->header->next->next = temp;  }  }  //// 有参构造，建立长度为n的顺序表，尾插法  //template <typename DataType>  //LinkList<DataType>::LinkList(DataType a[], int n) {  // this->header = new Node<DataType>;  // Node<DataType>\* temp = this->header;  // for (int i = 0; i < n; i++) {  // temp->next = new Node<DataType>;  // temp->next->data = a[i];  // temp = temp->next;  // }  // temp->next = nullptr;  //}  // 析构函数，就是在对象销毁时会被执行  template <typename DataType>  LinkList<DataType>::~LinkList() {  cout << "被销毁了" << endl;  }  // 获取线性表的长度  template <typename DataType>  int LinkList<DataType>::Length() {  int cnt = 0;  Node<DataType>\* temp = this->header;  while (temp->next != nullptr) {  temp = temp->next;  cnt++;  }  return cnt;  }  // 按位查找，查找第i个结点  template <typename DataType>  DataType LinkList<DataType>::Get(int i) {  if (i < 1 || i > this->Length()) {  throw "位置不存在";  }  Node<DataType>\* temp = this->header;  for (int j = 0; j < i; j++) {  temp = temp->next;  }  return temp->data;  }  // 按值查找，查找值为x的结点  // 0：没找到  template <typename DataType>  int LinkList<DataType>::Locate(DataType x) {  Node<DataType>\* temp = header;  for (int i = 0; i < this->Length(); i++) {  temp = temp->next;  if (temp->data == x) {  return i + 1;  }  }  return 0;  }  // 插入操作，在第i个位置插入值为x的结点  template <typename DataType>  void LinkList<DataType>::Insert(int i, DataType x) {  if (i < 1 || i > this->Length() + 1) throw "位置错误";  Node<DataType>\* temp = header, \*new\_node = new Node<DataType>;  for (int j = 0; j < i - 1; j++) {  temp = temp->next;  }  new\_node->data = x;  new\_node->next = temp->next;  temp->next = new\_node;  }  // 删除操作，删除第i个结点  template <typename DataType>  DataType LinkList<DataType>::Delete(int i) {  if (i < 1 || i > this->Length()) throw "位置错误";  DataType del = this->Get(i);  Node<DataType>\* node = header, \*temp = nullptr;  for (int j = 0; j < i - 1; j++) {  node = node->next;  }  temp = node->next;  node->next = node->next->next;  delete(temp);  return del;  }  // 判断线性表是否为空  template <typename DataType>  int LinkList<DataType>::Empty() {  if (this->header->next == nullptr) {  return 1;  }  return 0;  }  // 遍历操作，按序号依次输出各结点  template <typename DataType>  void LinkList<DataType>::PrintList() {  if (this->Empty()) {  cout << "{}" << endl;  return;  }  Node<DataType>\* temp = header->next;  cout << "{" << temp->data;  while (temp->next != nullptr) {  temp = temp->next;  cout << "," << temp->data;  }  cout << "}" << endl;  } |

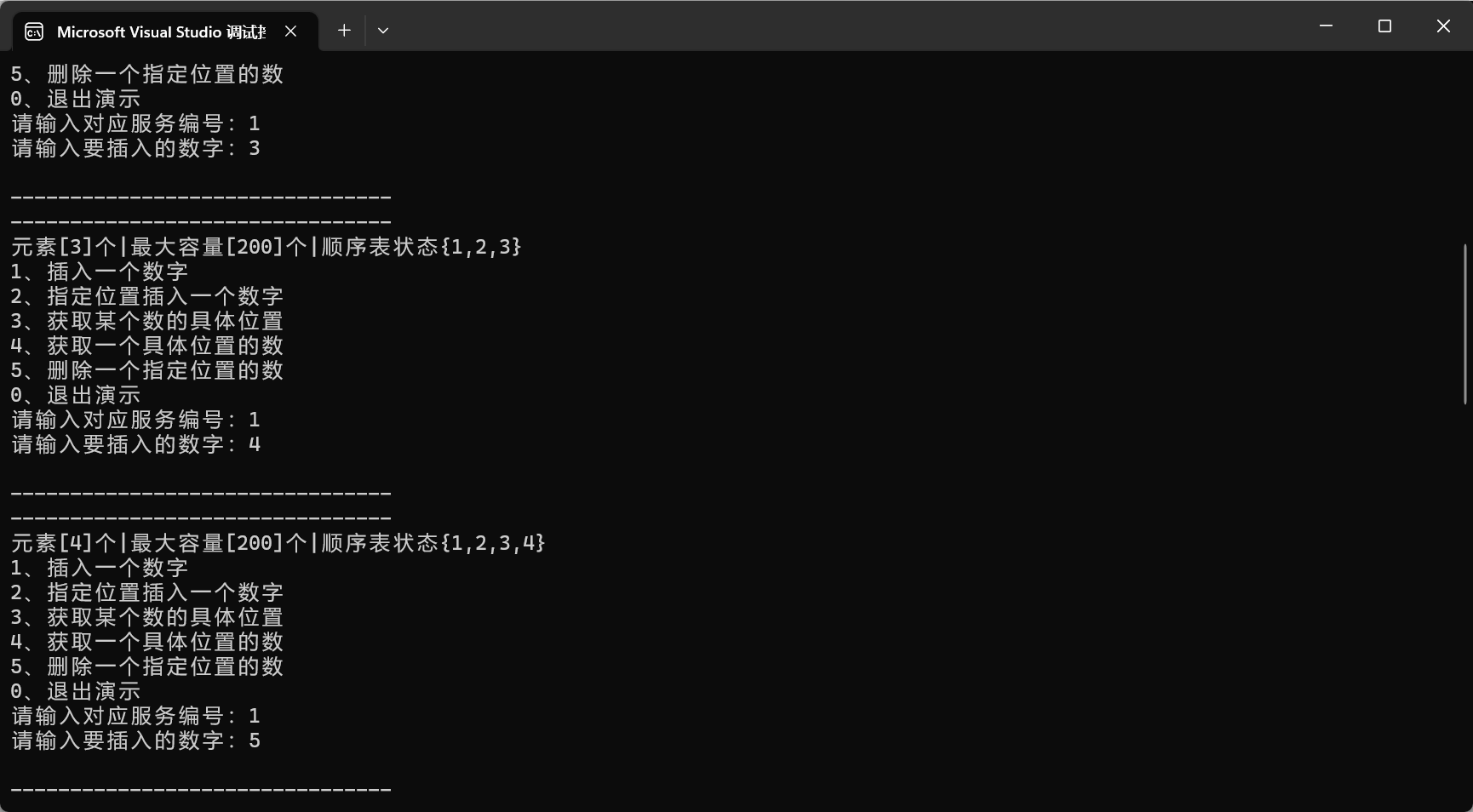
**Main.cpp**

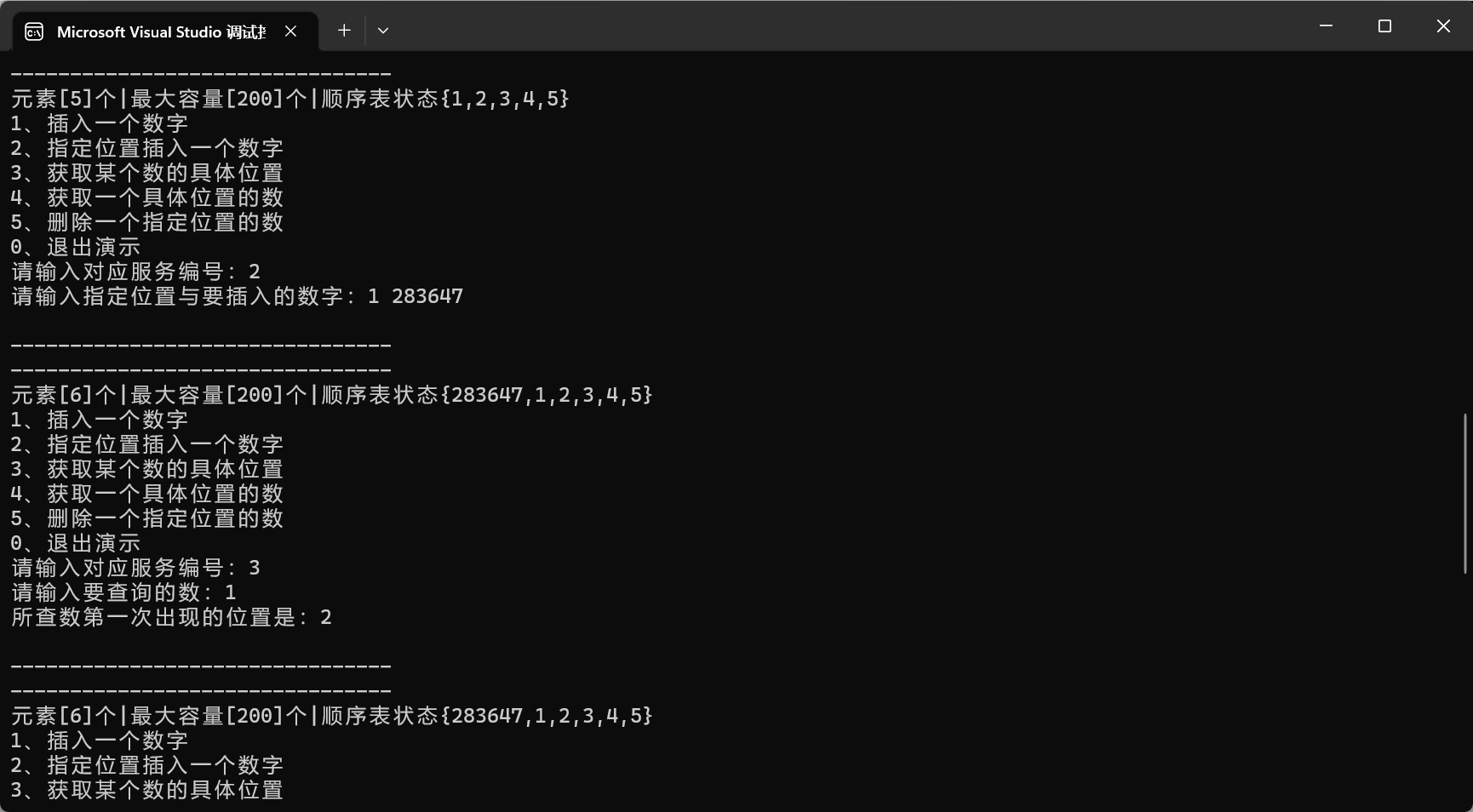
|  |
| --- |
| #include <iostream>  #include "LinkList.h"  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  // 清除控制台（未使用）  void Clear();  int main() {  int init[] = { 1, 2, 3, 4, 5, 6, 7, 8 };  LinkList<> list(init, 8);  int choose = -1;  while (choose) {  cout << "--------------------------------" << endl;  cout << "元素[" << list.Length() << "]个|单链表状态";  list.PrintList();  cout << "1、插入一个数字" << endl;  cout << "2、指定位置插入一个数字" << endl;  cout << "3、获取某个数的具体位置" << endl;  cout << "4、获取一个具体位置的数" << endl;  cout << "5、删除一个指定位置的数" << endl;  cout << "0、退出演示" << endl;  cout << "请输入对应服务编号：";  cin >> choose;  try {  Menu(choose, &list);  }  catch (const char\* e) {  cout << "出错啦，错误消息：" << e << endl;  }  cout << endl << "--------------------------------" << endl;  }  }  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list) {  int in\_1 = 0, in\_2 = 0, temp\_1 = 0;  switch (choose) {  // 插入一个数字  case 1:  cout << "请输入要插入的数字：";  cin >> in\_1;  list->Insert(list->Length() + 1, in\_1);  break;  // 指定位置插入一个数字  case 2:  cout << "请输入指定位置与要插入的数字：";  cin >> in\_1 >> in\_2;  list->Insert(in\_1, in\_2);  break;  // 获取某个数的具体位置  case 3:  cout << "请输入要查询的数：";  cin >> in\_1;  temp\_1 = list->Locate(in\_1);  if (temp\_1) {  cout << "所查数第一次出现的位置是：" << temp\_1 << endl;  }  else {  cout << "没有查到哦~" << endl;  }  break;  // 获取一个具体位置的数  case 4:  cout << "请输入要查找的位置：";  cin >> in\_1;  temp\_1 = list->Get(in\_1);  cout << "所在位置的数是：" << temp\_1 << endl;  break;  // 删除一个指定位置的数  case 5:  cout << "请输入要删除的位置：";  cin >> in\_1;  list->Delete(in\_1);  break;  case 0:  cout << "感谢体验，再见~" << endl;  break;  default:  cout << "输入有误" << endl;  }  }  // 清除屏幕（未使用）  void Clear()  {  cout << "\x1B[2J\x1B[H";  } |

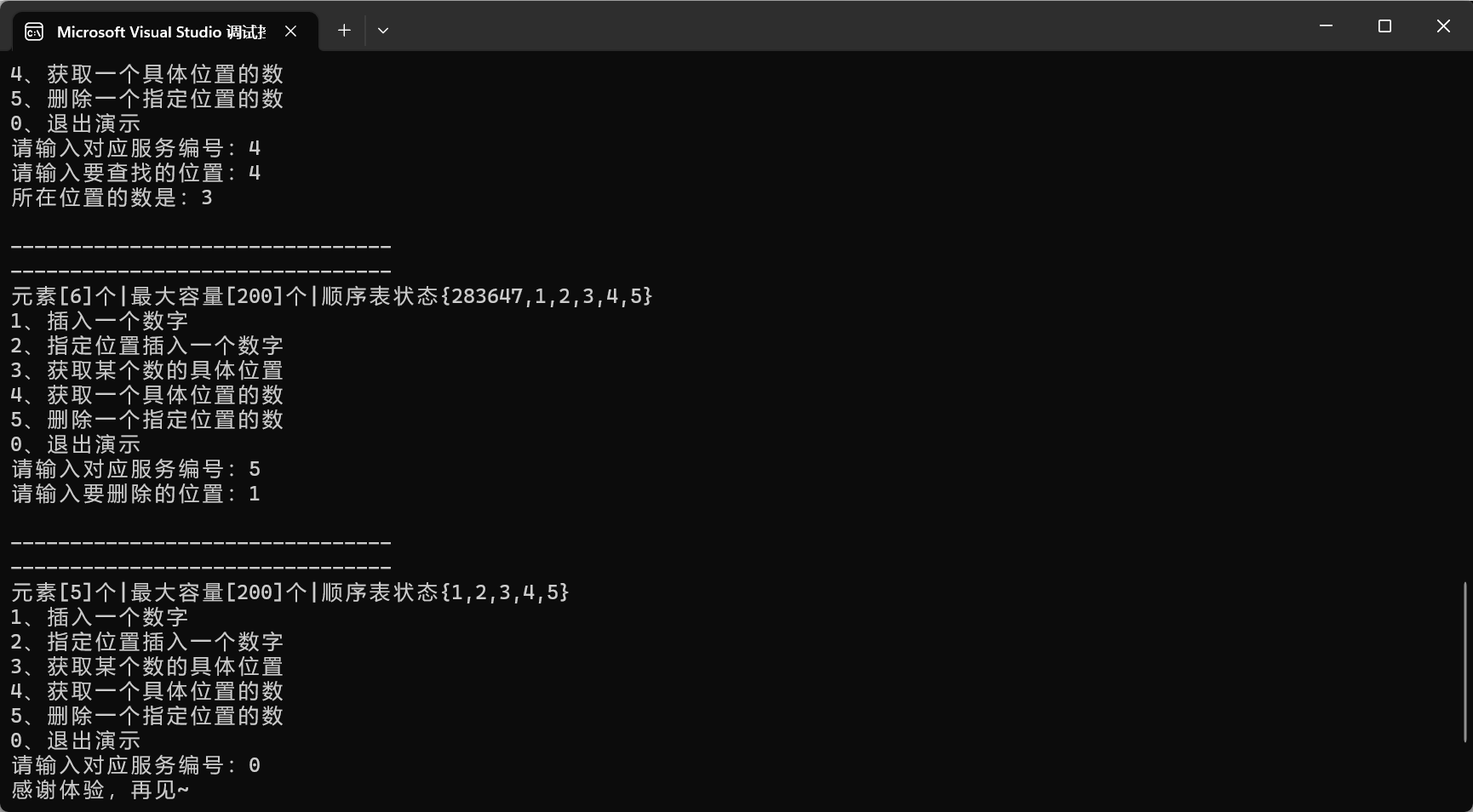
## 运行结果

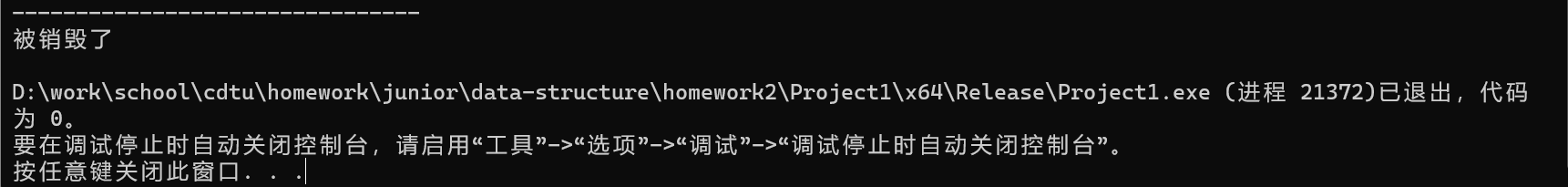
### 顺序表











### 单链表

界面和功能与顺序表一样

