|  |
| --- |
| **《数据结构》**  **实验报告**  项目名称 线性表的操作  专业班级 软工2304  学 号 8214230404  姓 名 王银波  **实验成绩：**  **批阅教师：**  2025年 4月 13日 |

# 实验1《线性表的操作》

实验学时： 2 实验地点： 信息楼404 实验日期： 2025年4月15日

1. **需求分析**

**（实验指导书要求复制过去）**

1、掌握线性表的基本操作，插入、删除、查找，以及线性表合并等运算在顺序存储结构和链接存储结构上的运算。

2、熟练运用掌握的线性表的操作，实现一元n次多项式的加法运算。

**二、实验内容**

顺序存储的线性表有一些弱点，其一，插入与删除元素需要大量移动元素；其二，预先分配存储空间时必须按最大的空间来分配。其三，表长难以扩充。

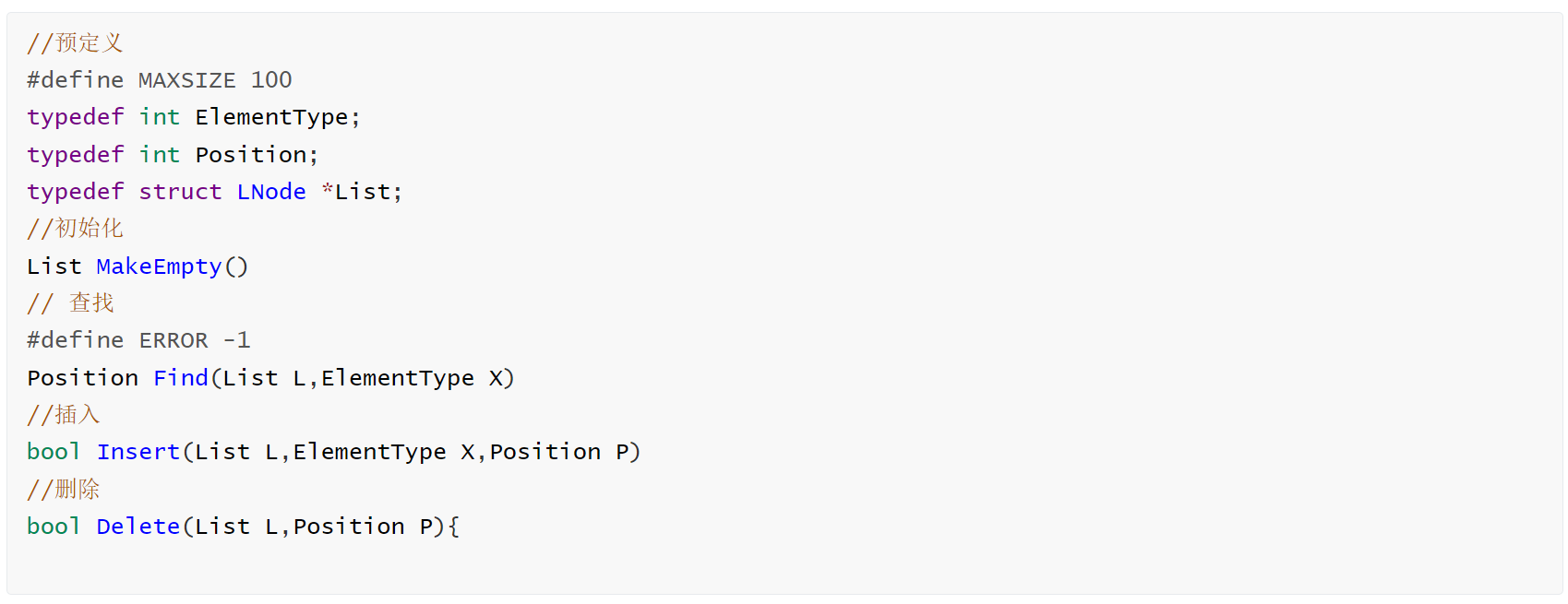
链式存储结构的特点是用一组任意的存储单元存储线性链表的数据元素，与顺序表的区别在于链式存储的存储单元可以是连续的，也可以是不连续的。为了实现这种结构，链表采取由两部分信息组成数据元素ai的存储映像，称为结点。结点包括两个域，其中存储数据信息的域称为数据域，存储直接后继信息的称为指针域。指针域中存储的信息叫做指针或链。这样，n个结点链接成一个链表，即为线性表（a1,a2,a3,•••,an）。

符号多项式的操作，已经成为表处理的典型用例，在数学上，一个一元多项式pn(x)可以按升幂写成：pn(x)=p0+p1•x+p2•（x的2次幂）+•••+pn•（x的n次幂），也可以按照降幂来排列。它由n+1个系数唯一确定。因此，在计算机里，它可用一个线性表P来表示：P=(p0,p1,p2,•••,pn),显然，在通常的应用中，多项式的次数变化很高且很大，此种表示将造成内存的很大浪费。所以在实际应用中通常只记录多项式的系数非零的项，也就是系数非零项的次数和对应的非零系数。

本次实验主要完成以下的内容：

* 线性表基本操作的实现，分别采用数组和链表结构实现线性表，实现线性表的基本操作。
* 利用实现的线性表，存储一元n次多项式，完成多项式的输入、显示；实现多项式的加法操作
* 可选内容：设计实现一个处理100位以内的长整数加减运算的程序。

1. **概要设计**
2. 线性表核心伪代码：



1. 链表伪核心代码：

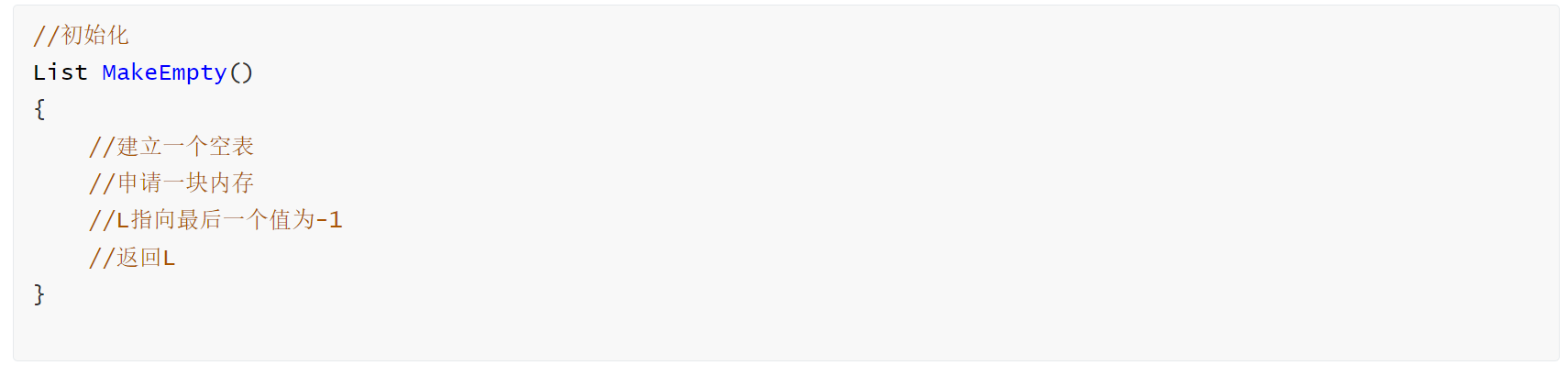
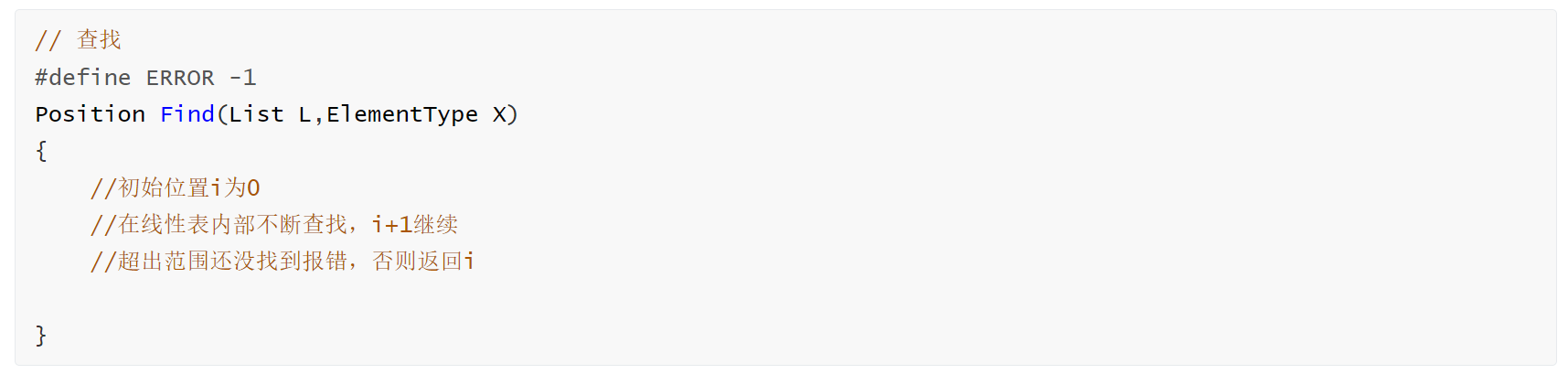
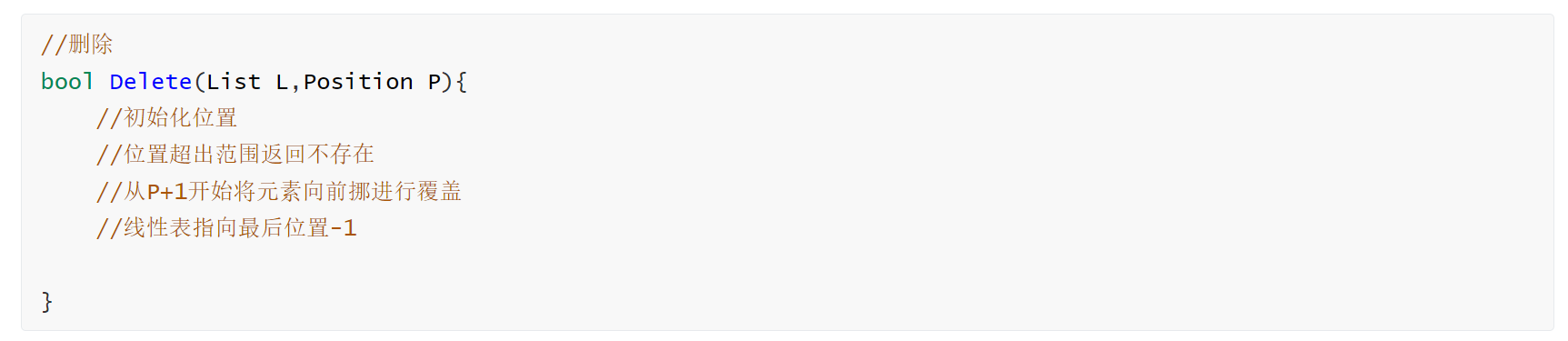
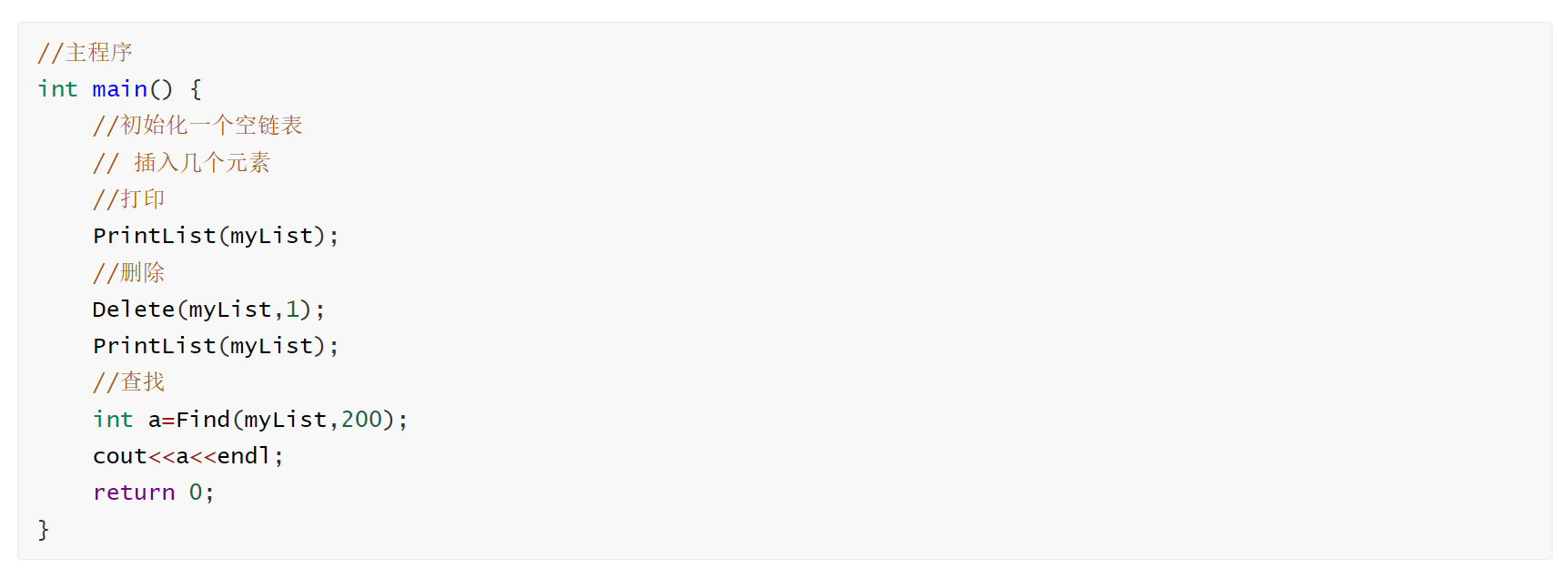


1. 多项式计算核心伪代码：

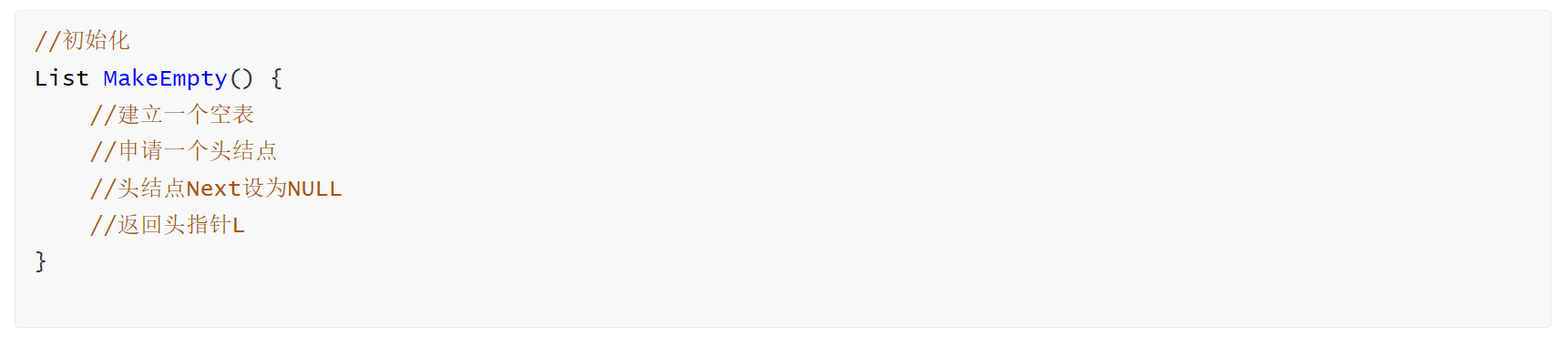


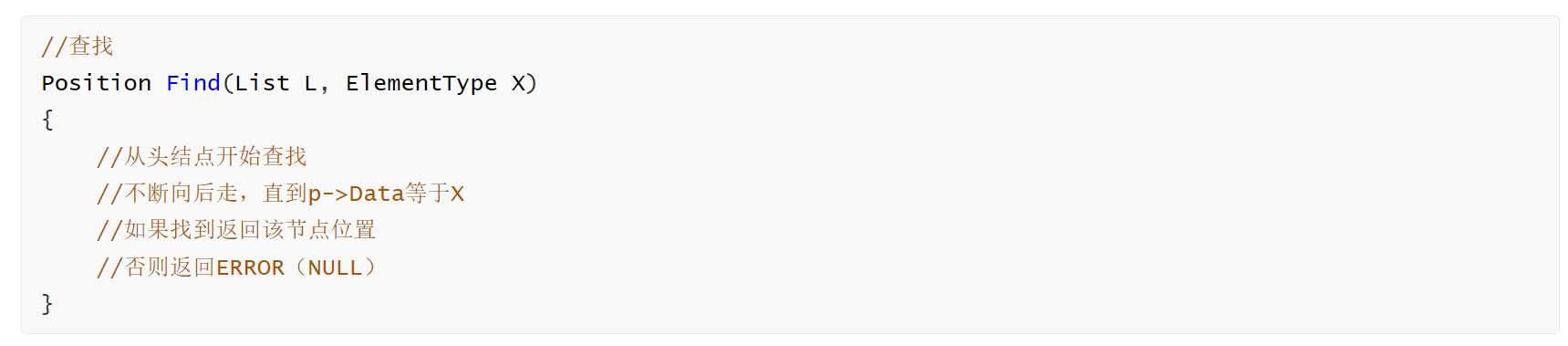
**3．详细设计**

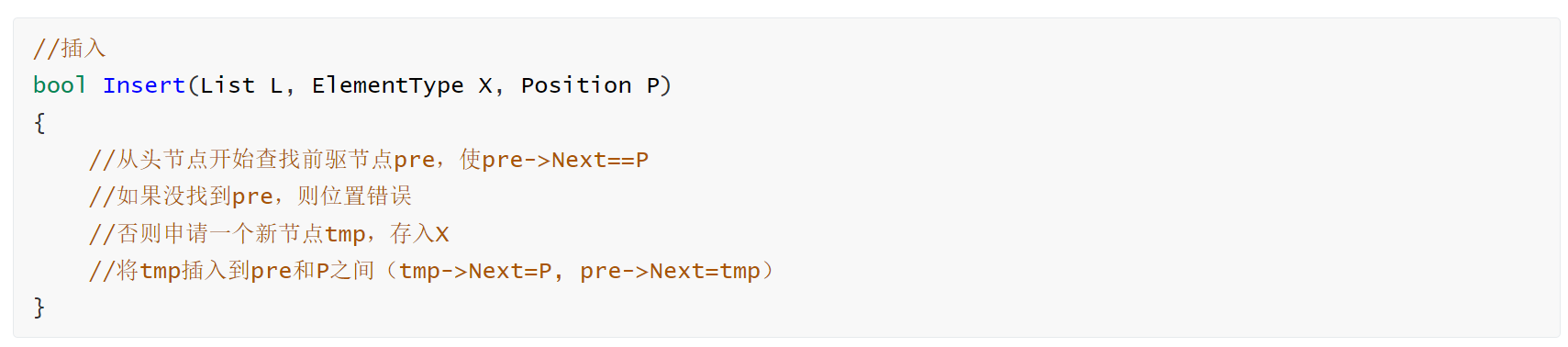
1. 线性表的详细设计：

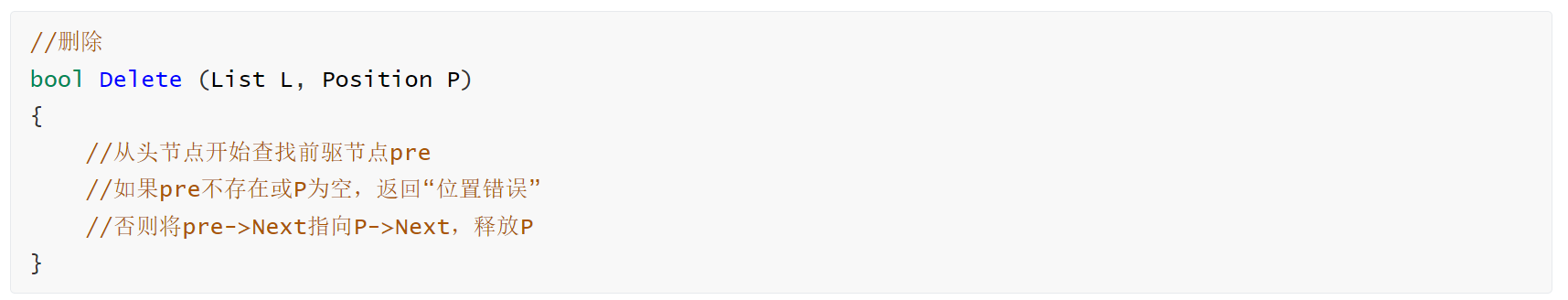
    

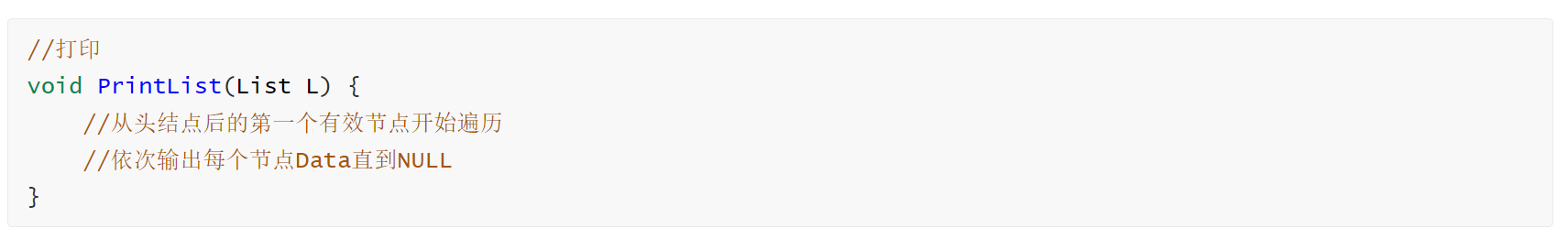
1. 链表结构的详细设计









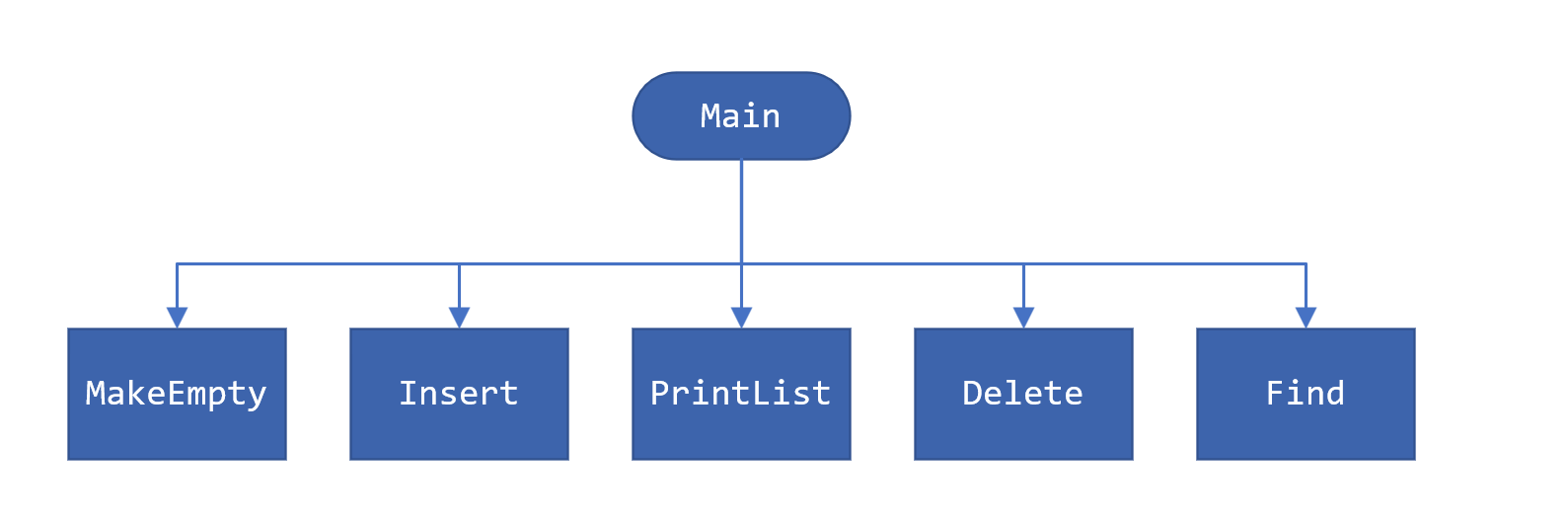




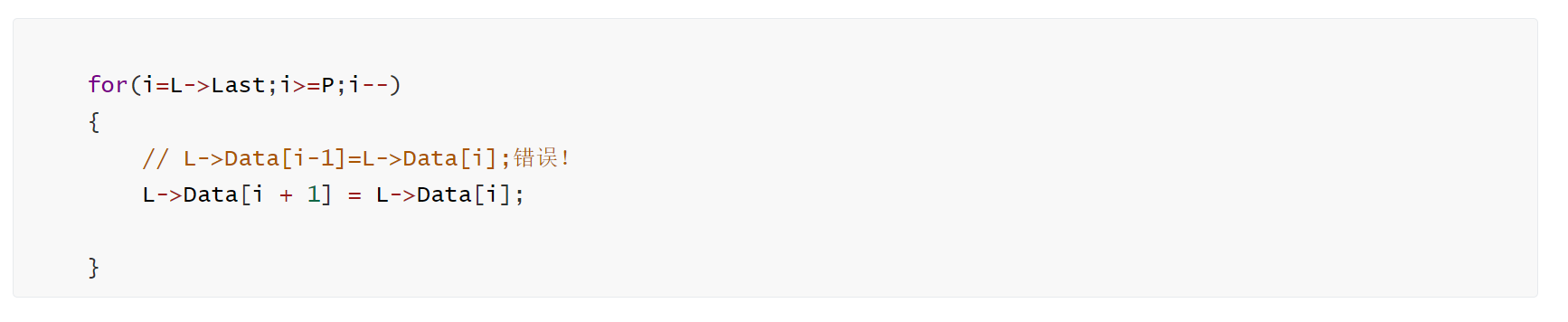
1. 多项式计算的详细设计

多项式利用线性表作为存储，通过下标和数值分别存储多项式的指数和系数，进而进行求解。



流程图（链表和线性表相同）  


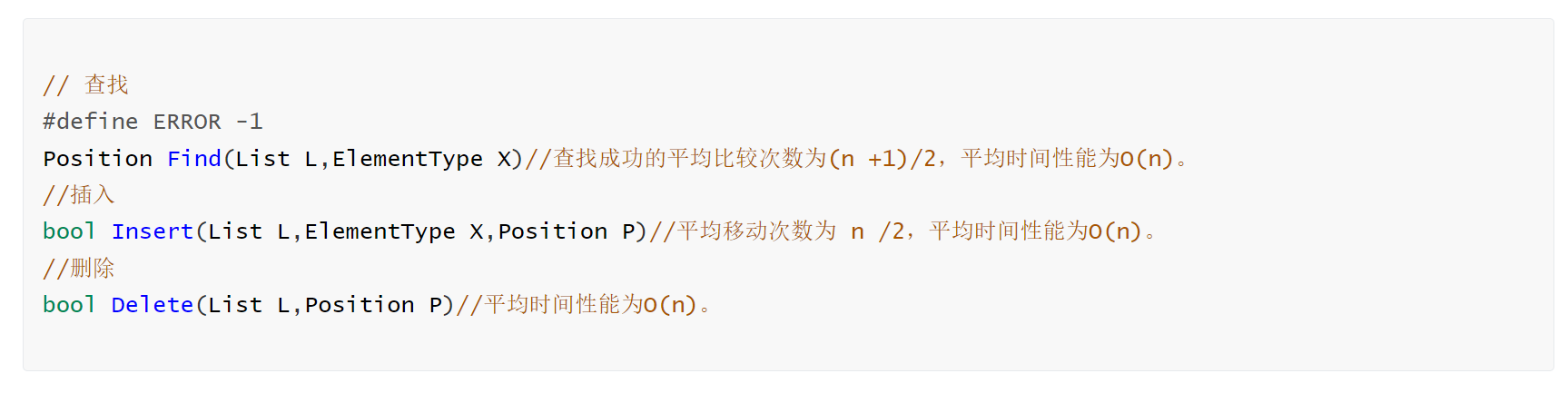
**4．调试分析**

1、线性表的设计，在插入操作中，索引更新错误导致地址溢出，线性表最后一个指向一个无效的数字，检查过后进行了更正。  


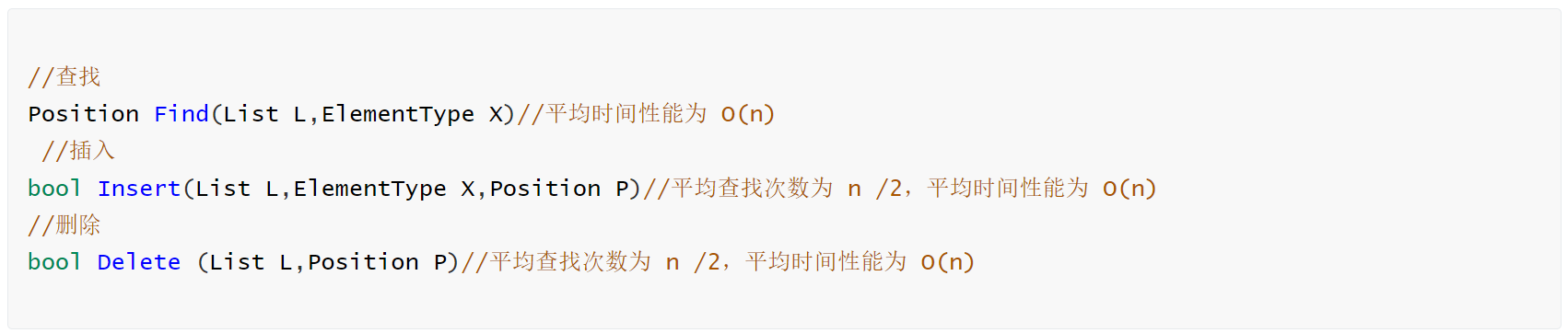
2、参考课本资料预定义一些值时，未能完全理解含义而写错。

typedef struct LNode \*PtrToNode;zh这里LNode 是一个结构体指针，起了一个别名叫做Lnode,所以下面结构体中它都是一个指针。

线性表各操作的时间复杂度：



链表的各操作的时间复杂度：



**5．用户使用说明**

1、线性表

建立一个新表List myList=MakeEmpty();直接调用函数即可，插入元素也是Insert(myList, 100, 0); 查找返回的是位置的下标, int a=Find(myList,200);需要这样调用。

2、链表

使用 List list = MakeEmpty(); 来创建一个空链表（带头结点），使用 Insert(list, 200, list->Next);表示在链表 list 中，向 list->Next 这个位置（即第一个元素之前）插入 200。

因为链表插入需要提供插入位置的前驱节点，所以这里传的是 list（头结点）或 某节点的前驱。使用 Position a = Find(list, 300);返回的是指向该节点的指针（Position 类型），如果查找失败，返回 NULL。删除元素：先用 Position p = Find(list, 200); 找到需要删除的节点，

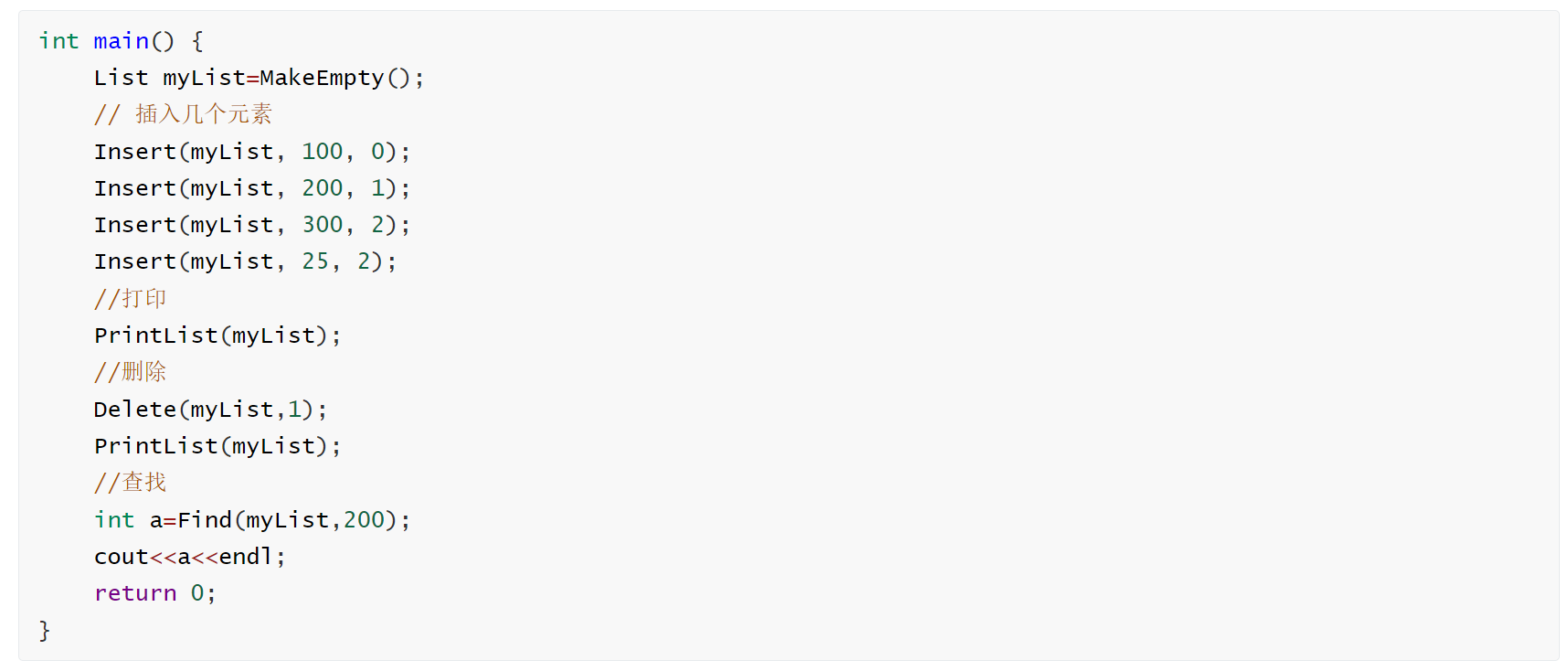
然后调用 Delete(list, p); 删除该节点。

1. 多项式

首先输入多项式的项数，然后从小到大依次输入对应多项式的系数，最后调用Add函数,对新的表L3做处理，就是返回结果。

**6．测试结果（程序结果截图）**

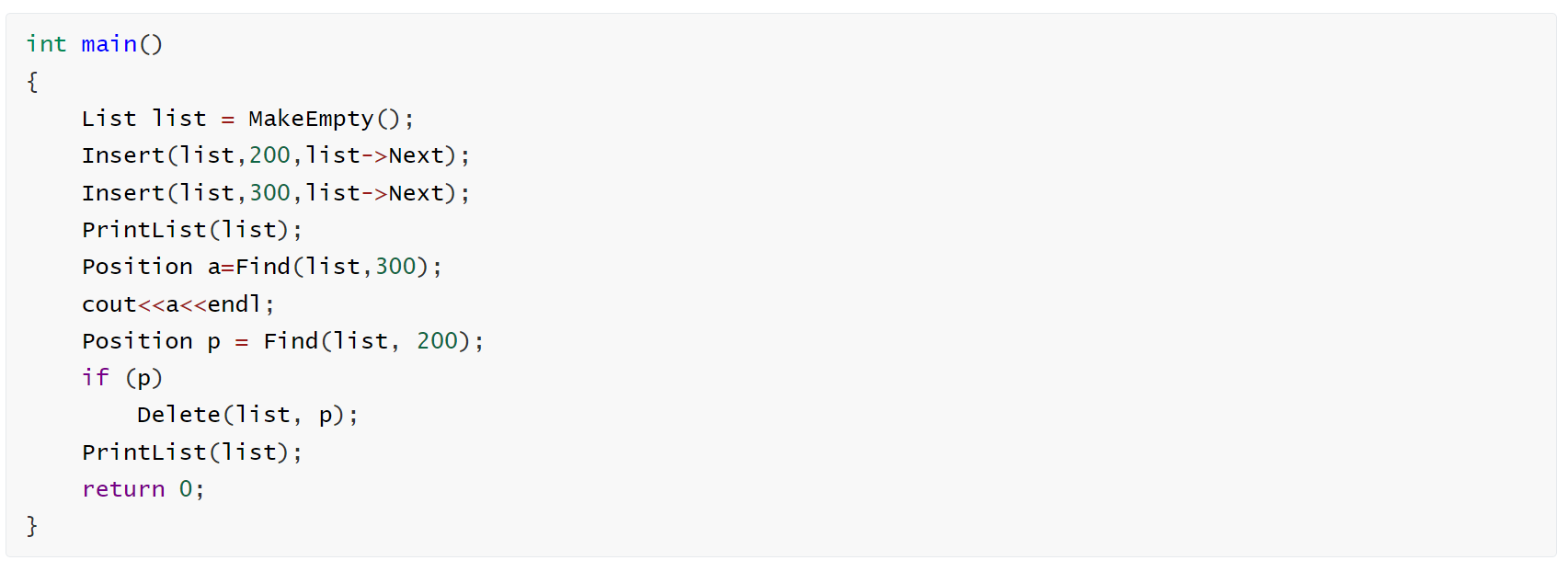
**1、线性表**

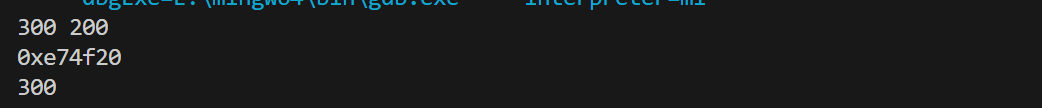
****

****

可见成功插入了100，200，25，300，并删除了两个元素，最后查找200没找到返回-1。

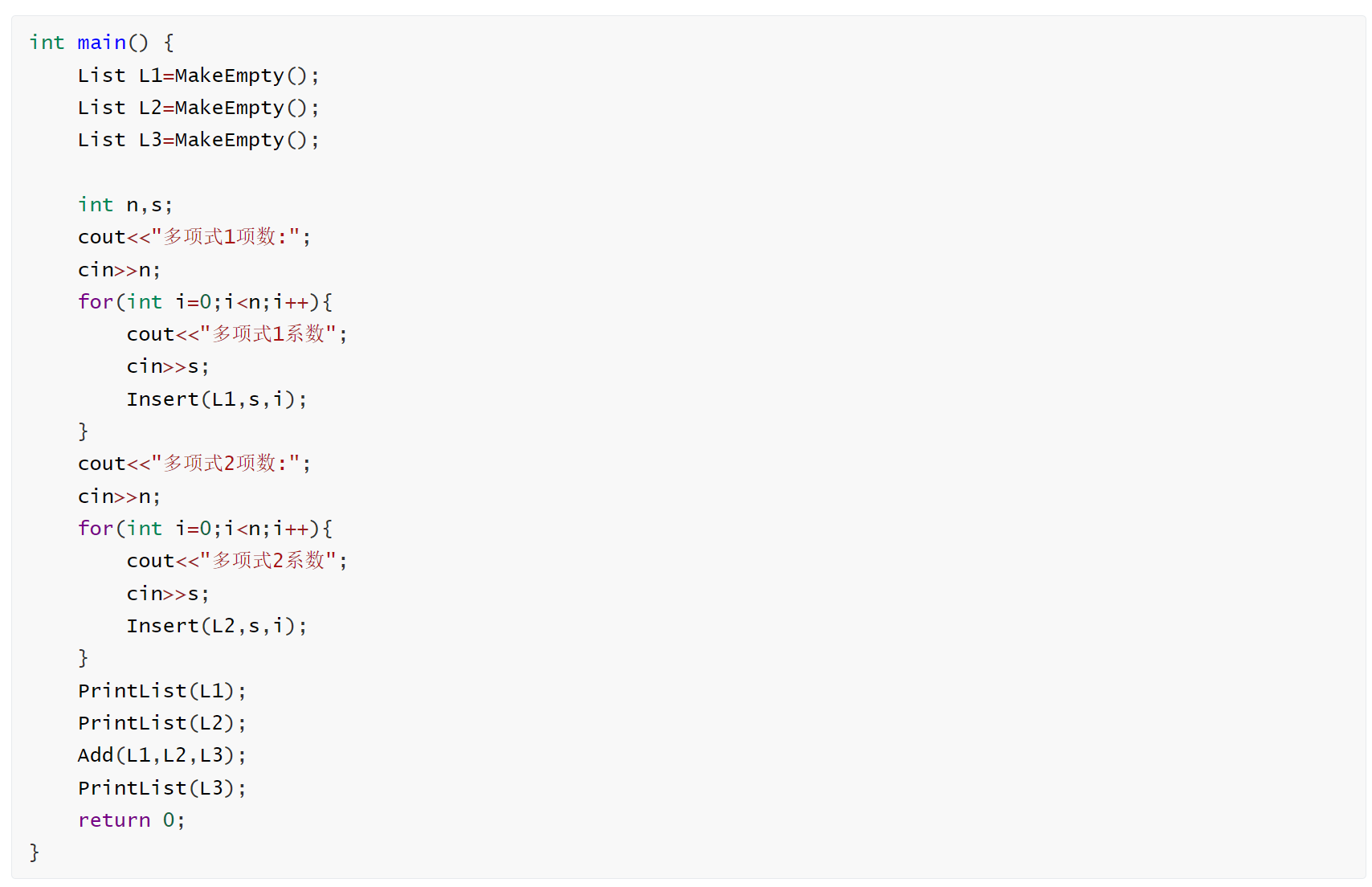
2、线性表





可见插入了两个元素300，200，并返回了300的地址，然后删掉200，最后链表只剩下200.

3、多项式





可见成功输出了L1+L2的结果。

总结：本次实验对线性表和链表进行了实现和设计，并利用线性表做了一个简单的应用多项式的计算，加深了对于C++ struct 和typedef的使用，熟练了C++的语法，也熟练了数据结构线性表的操作与存储。

**7．附录**

#### 1、线性表

#include <iostream>   
using namespace std;   
//预定义   
#define MAXSIZE 100   
typedef int ElementType;   
typedef int Position;   
typedef struct LNode \*List;   
struct LNode{   
 ElementType Data[MAXSIZE];   
 Position Last;   
};   
//初始化   
List MakeEmpty()   
{   
 List L;   
 L=(List)malloc(sizeof(struct LNode));   
 L->Last=-1;   
 return L;   
}   
// 查找   
#define ERROR -1   
Position Find(List L,ElementType X)   
{   
 Position i=0;   
 while (i<= L->Last && L->Data[i]!=X)   
 i++;   
 if(i>L->Last)return ERROR;   
 else return i;   
}   
   
//插入   
bool Insert(List L,ElementType X,Position P)   
{   
 Position i;   
 if(L->Last==MAXSIZE-1)   
 {   
 cout<<"已满"<<endl;   
 return false;   
 }   
 if(P<0 || P>L->Last+1)   
 {   
 cout<<"位置不合法"<<endl;   
 return false;   
 }   
 for(i=L->Last;i>=P;i--)   
 {   
 // L->Data[i-1]=L->Data[i];错误！   
 L->Data[i + 1] = L->Data[i];   
   
 }   
 L->Data[P]=X;   
 L->Last++;   
 cout<<"插入成功！"<<endl;   
 return true;   
}   
void PrintList(List L){   
 for(int i=0;i<=L->Last;i++)   
 {   
 cout<<L->Data[i]<<endl;   
 }   
}   
//删除   
bool Delete(List L,Position P){   
 Position i;   
 if (P<0 || P> L->Last)   
 {   
 cout<<"位置"<<P<<"不存在元素"<<endl;   
 return false;   
 }   
 for(i=P+1;i<= L->Last;i++)   
 {   
 L->Data[i-1]=L->Data[i];   
 }   
 L->Last--;   
 return true;   
}   
int main() {   
 List myList=MakeEmpty();   
 // 插入几个元素   
 Insert(myList, 100, 0);   
 Insert(myList, 200, 1);   
 Insert(myList, 300, 2);   
 Insert(myList, 25, 2);   
 //打印   
 PrintList(myList);   
 //删除   
 Delete(myList,1);   
 PrintList(myList);   
 //查找   
 int a=Find(myList,200);   
 cout<<a<<endl;   
 return 0;   
}

#### 2、链表

#include <iostream>   
using namespace std;   
   
#define ERROR NULL   
typedef int ElementType;   
   
typedef struct LNode \*PtrToNode;   
struct LNode{   
 ElementType Data;   
 PtrToNode Next;   
};   
typedef PtrToNode Position;   
typedef PtrToNode List;   
//初始化   
List MakeEmpty() {   
 List L = (List)malloc(sizeof(struct LNode));   
 L->Next = NULL;   
 return L;   
}   
//查找   
Position Find(List L,ElementType X)   
{   
 Position p=L;   
 while (p&&p->Data!=X)   
 p=p->Next;   
 if (p)   
 return p;   
 else   
 return ERROR;   
}   
   
//插入   
bool Insert(List L,ElementType X,Position P)   
{   
 Position tmp,pre;   
 for(pre=L;pre&&pre->Next!=P;pre=pre->Next);   
 if(pre ==NULL)   
 {   
 cout<<"插入位置错误"<<endl;   
 return false;   
 }else{   
 tmp=(Position)malloc(sizeof(struct LNode));   
 tmp->Data=X;   
 tmp->Next=P;   
 pre->Next=tmp;   
 return true;   
 }   
}   
   
//删除   
bool Delete (List L,Position P)   
{   
 Position pre;   
 for(pre=L;pre&&pre->Next!=P;pre=pre->Next);   
 if(pre==NULL||P==NULL)   
 {   
 cout<<"位置错误"<<endl;   
 }else{   
 pre->Next=P->Next;   
 free(P);   
 return true;   
 }   
}   
void PrintList(List L) {   
 Position p = L->Next; // 跳过头结点   
 while (p) {   
 cout << p->Data << " ";   
 p = p->Next;   
 }   
 cout << endl;   
}   
int main()   
{   
 List list = MakeEmpty();   
 Insert(list,200,list->Next);   
 Insert(list,300,list->Next);   
 PrintList(list);   
 Position a=Find(list,300);   
 cout<<a<<endl;   
 Position p = Find(list, 200);   
 if (p)   
 Delete(list, p);   
 PrintList(list);   
 return 0;   
}

#### 3、多项式计算

#include <iostream>   
using namespace std;   
//预定义   
#define MAXSIZE 100   
typedef int ElementType;   
typedef int Position;   
typedef struct LNode \*List;   
struct LNode{   
 ElementType Data[MAXSIZE];   
 Position Last;   
};   
//初始化   
List MakeEmpty()   
{   
 List L;   
 L=(List)malloc(sizeof(struct LNode));   
 L->Last=-1;   
 return L;   
}   
// 查找   
#define ERROR -1   
Position Find(List L,ElementType X)   
{   
 Position i=0;   
 while (i<= L->Last && L->Data[i]!=X)   
 i++;   
 if(i>L->Last)return ERROR;   
 else return i;   
}   
   
void PrintList(List L){   
 for(int i=0;i<=L->Last;i++)   
 {   
 if(i>0 && L->Data[i]>0)   
 {   
 cout<<"+";   
 cout<<L->Data[i];   
 };   
 if(i<L->Last)   
 cout<<"x^"<<i<<" ";   
 }   
 cout<<endl;   
}   
//插入系数   
bool Insert(List L,ElementType X,Position P)   
{   
 Position i;   
 if(L->Last==MAXSIZE-1){   
 cout<<"已满，不能插入！"<<endl;   
 return false;   
 };   
 if (P < 0 || P > L->Last + 1) {   
 cout << "位置不合法" << endl;   
 return false;   
 }   
 for (i = L->Last; i >= P; i--) {   
 L->Data[i + 1] = L->Data[i];   
 }   
 L->Data[P] = X;   
 L->Last++;   
 return true;   
}   
//做加法   
void Add(List L1,List L2,List L3)   
{   
 Position i;   
 for(int i=0;i<=L1->Last || i<=L2->Last;i++){   
 int x=(i<=L1->Last)? L1->Data[i]:0;   
 int y=(i<=L2->Last)? L2->Data[i]:0;   
 Insert(L3,x+y,i);   
 }   
}   
int main() {   
 List L1=MakeEmpty();   
 List L2=MakeEmpty();   
 List L3=MakeEmpty();   
   
 int n,s;   
 cout<<"多项式1项数:";   
 cin>>n;   
 for(int i=0;i<n;i++){   
 cout<<"多项式1系数";   
 cin>>s;   
 Insert(L1,s,i);   
 }   
 cout<<"多项式2项数:";   
 cin>>n;   
 for(int i=0;i<n;i++){   
 cout<<"多项式2系数";   
 cin>>s;   
 Insert(L2,s,i);   
 }   
 PrintList(L1);   
 PrintList(L2);   
 Add(L1,L2,L3);   
 PrintList(L3);   
 return 0;   
}