**********

**实践教学**

**********

# 兰州理工大学

2018 年春季学期

## 算法与数据结构课程设计

题　　目：1.排序算法比较问题
2.递归替换问题
3.跳马问题
4.长整数运算问题

专业班级：计算机科学与技术 2 班

姓　　名：宋智勇

学　　号：1616240206

指导教师：包仲贤

成　　绩：

# 目　录

# 摘　要

本程序主要解决排序算法比较问题，递归替换问题，跳马问题，长整数运算问题。排序算法比较问题是针对实际数据的特点选择合适的排序算法可以使程序获得更高的效率，有时候排序的稳定性还是实际问题中必须考虑的。递归替换问题利用递归算法解决文件替换问题。跳马问题也称为马踏棋盘问题，是算法的设计中的经典问题。在 8*8 的方格棋盘中马的行走规则从棋盘的某一方格出发，开始在棋盘上周游，如果能不重复地走遍棋盘上的每一个方格，这样的一条周游路线在数学上被称为国际象棋棋盘上马的哈密尔顿链。长整数运算问题实现两个任意长的正负整数的加减乘除的计算问题。通过该题目的设计过程，可以加深理解线性表的逻辑结构、存储结构，掌握线性表上基本运算的实现。这些程序主要功能是加深我们对算法与数据结构中存储，线性表和栈的理解。让我们对算法与数据结构有个更深刻的认识。

关键词：排序算法比较；递归；跳马；长整数运算；数据结构

# 一．排序算法比较问题（2）

设计各类排序算法的程序，通过随机的数据测试，比较各算法的关键字比较次数和关键字移动次数。

## 1．采用类语言定义相关的数据类型

```
typedef struct node {
    int data3;
    int next;
} node;
//可排序表的长度
int Size;
int head;
//统计比较次数
long compCount;
//统计移动次数
long shiftCount;
```

## 2．算法设计

a、利用全局变量存储比较和移动次数，用文件存储源数据和比较的数据。

b、大小比较和数值移动使用统一的函数比较和数值移动并对所需要的次数进行叠加，用函数替代所有排序函数中的比较和移动操作。

c、比较之前所有次数清零，返回排序源数据。

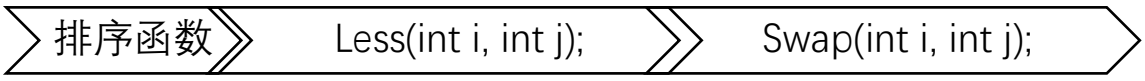


图 1-1 计数原理图

## 2．函数的调用关系图

由于四个程序合到了一起，所以在此只讲子主函数。

本子程序共使用了二十个函数，十个排序函数及排序相关的五个子函数，sort

（）函数为主函数。Less(),swap()等为计数函数。

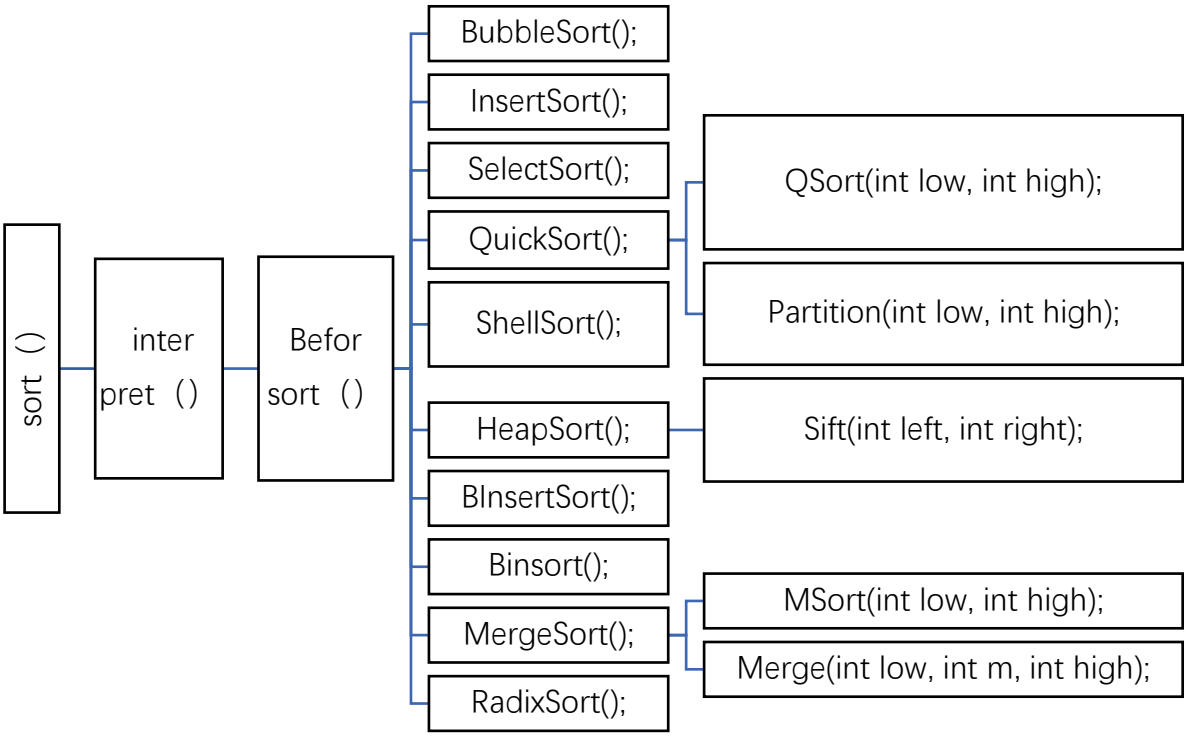


图 1-1 程序调用图

## 4. 调试分析

a、主要是文件操作时没能找到一个令人心满意足的方案；

b、生成随机数不理想，不支持太大的数；

c、由于数据量比较大，难作证明测试。

## 5. 测试结果

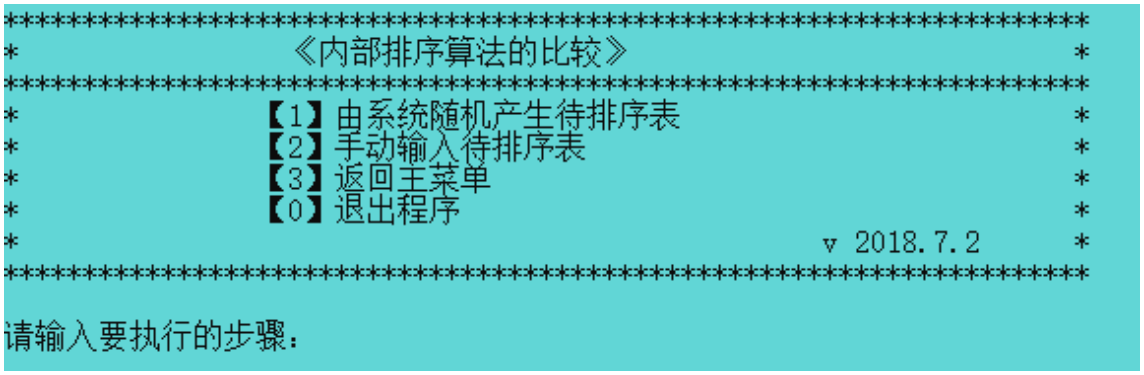


图 1-2 程序初始化结果

```
请输入要执行的步骤：1
请输入待排序表的长度:50
        43 42 44 28 10 17 39 28 32 32 23 37 49 31 33 3 39 1 0 8 17 30 1
2 49 6 36 25 8 4 28 34 37 17 16 49 39 13 12 45 29 27 11 40 50 32 29 17 4
1 19
由系统随机产生待排序表的各个算法的比较次数和移动次数如下：
        compCount       shiftCount
Bubbl:  1089            1815
Tnser:  654             703
Selec:  1225            141
Quick:  421             220
Shell:  284             354
Heap :  420             563
BInse:  215             703
Merge:  226             572
Bin :   568             570
Radix:  17              147

0 1 1 2 3 4 6 8 8 10 11 12 13 16 17 17 17 17 19 23 25 27 28 28 28 29 29
30 31 32 32 32 33 34 36 37 37 39 39 39 40 41 42 43 44 45 49 49 49 50
按任意键继续：
```

图 1-3 随机数排序测试结果

```
请输入要执行的步骤：2
请输入待排序表的长度：50
请输入50个数据:
 29 42 9 13 17 24 18 2 37 16 12 42 32 10 46 31 10 19 24 40 24 4 23 41 0
5 29 39 2 38 43 9 32 3 6 40 8 2 7 39 47 2 22 44 0 33 33 47 50 44
手动输入待排序表的各个算法的比较次数和移动次数如下：
        compCount       shiftCount
Bubbl:  1210            1641
Tnser:  596             645
Selec:  1225            141
Quick:  402             204
Shell:  286             360
Heap :  411             560
BInse:  215             645
Merge:  221             572
Bin :   313             315
Radix:  17              147

0 0 2 2 2 2 3 4 5 6 7 8 9 9 10 10 12 13 16 17 18 19 22 23 24 24 24 29 29
 31 32 32 33 33 37 38 39 39 40 40 41 42 42 43 44 44 46 47 47 50
按任意键继续：
```

图 1-4 人工输入排序测试结果

## 6. 源程序（带注释）

```
#pragma once
#include<iostream>
#include<ctime>
#include<fstream>
#define MAXSIZE 1000    //可排序表的最大长度
```

```
#define SORTNUM 10     //测试 10 中排序方法
#define max 100       //基数排序时数据的最大位数不超过百位；
typedef int DataType[MAXSIZE + 2];
typedef struct node {
      int data3;
      int next;
} node;

//对比较次数和移动次数清零
void BeforeSort();
//若表中第 i 个元素小于第 j 个元素，则返回 True，否则返回 False
bool Less(int i, int j);
//交换表中第 i 个和第 j 个元素
void Swap(int i, int j);
//将 R2[j]赋给 R[i]
void Shift(DataType &R, DataType &R2, int i, int j);
void CopyData(DataType list1, DataType list2);
//将可排序表置为逆序
void InverseOrder();
//由系统随机一组数
void RandomizeList();
//输出函数
void output();
//冒泡排序
void BubbleSort();
//插入排序
void InsertSort();
//选择排序
void SelectSort();
int Partition(int low, int high);
//QuickSort 的辅助函数
void QSort(int low, int high);
```

```
//快速排序
void QuickSort();
//希尔排序
void ShellSort();
//堆排序的调堆函数
void Sift(int left, int right);
//堆排序
void HeapSort();
//折半插入排序
void BInsertSort();
//2-路插入排序
void Binsort();
void Merge(int low, int m, int high);
void MSort(int low, int high);
//归并排序
void MergeSort();
void Distribute(node *a, int w);
void Collect(node *a);
//基数排序算法
void RadixSort();
//系统初始化
void Initialization();
//调用各个算法
void Interpret(int cmd);
void sort()

#include"sort.h"
using namespace std;

DataType data1;
DataType data2;
DataType R1;
```

```
int Size;//可排序表的长度
int head;
int fr[10];
int re[10];
long compCount;//统计比较次数
long shiftCount;//统计移动次数

void BeforeSort()//对比较次数和移动次数清零
{
    compCount = 0;
    shiftCount = 0;
}

bool Less(int i, int j)//若表中第 i 个元素小于第 j 个元素，则返回 True，否则返回 False
{
    compCount++;
    return data1[i] < data1[j];
}

void Swap(int i, int j)//交换表中第 i 个和第 j 个元素
{
    int a;
    a = data1[i];
    data1[i] = data1[j];
    data1[j] = a;
    shiftCount = shiftCount + 3;
}

void Shift(DataType &R, DataType &R2, int i, int j)//将 R2[j]赋给 R[i]
{
    R[i] = R2[j];
    shiftCount++;
```

```
}

void CopyData(DataType list1, DataType list2)
{
    int i;
    for (i = 1; i <= Size; i++) list2[i] = list1[i];

}

void InverseOrder()//将可排序表置为逆序
{
    int i, j;
    for (i = 1, j = Size; i <= Size / 2; i++, j--){
        int a;
        a = data1[i];
        data1[i] = data1[j];
        data1[j] = a;
    }
    CopyData(data1, data2);
}

void RandomizeList()//由系统随机一组数
{
    int i;
    srand(time(0));
    for (i = 1; i <= Size; i++)
        data1[i] = rand() % (Size + 1);
    CopyData(data1, data2);
    ofstream out_stream;
    out_stream.open("input.txt", ios::app);
    if (out_stream.fail()){
        cout << "input file opening failed.\n";
        exit(1);
```

```
    }
    for (i = 1; i <= Size; i++) out_stream << data1[i] << " ";
    out_stream << "\n";
    out_stream.close();
}

void output()//输出函数
{
    ofstream out_stream;
    cout << "\t" << compCount << "\t\t" << shiftCount << "\n";
    out_stream.open("output.txt", ios::app);
    if (out_stream.fail()){
        cout << "Output file opening failed.\n";
        exit(1);
    }
    out_stream << "\t" << compCount << "\t\t" << shiftCount << "\n";
    out_stream.close();
}

void BubbleSort()//冒泡排序
{
    BeforeSort();
    int swapped, i, m;
    m = Size - 1;
    do {
        swapped = 0;
        for (i = 1; i <= m; i++)
            if (Less(i + 1, i)){
                Swap(i + 1, i);
                swapped = 1;
            }
        m--;
    } while (swapped);
```

```
    output();
}

void InsertSort() //插入排序
{
    BeforeSort();
    int i, j;
    for (i = 2; i <= Size; i++){
        Shift(data1, data1, 0, i);
        j = i - 1;
        while (Less(0, j)){
            Shift(data1, data1, j + 1, j);
            j--;
        }
        Shift(data1, data1, j + 1, 0);
    }
    output();
}

void SelectSort()//选择排序
{
    BeforeSort();
    int i, j, min;
    for (i = 1; i <= Size - 1; i++){
        min = i;
        for (j = i + 1; j <= Size; j++)
            if (Less(j, min)) min = j;
        if (i != min) Swap(i, min);
    }
    output();
}

int Partition(int low, int high)
```

```
{
    int pivotkey;
    Shift(data1, data1, 0, low);
    pivotkey = data1[low];
    while (low < high) {
        compCount++;
        while (low < high&&data1[high] >= pivotkey) { compCount++; --high; }
        Shift(data1, data1, low, high);
        compCount++;
        while (low < high&&data1[low] <= pivotkey) { compCount++; ++low; }
        Shift(data1, data1, high, low);
    }
    Shift(data1, data1, low, 0);
    return low;

}

void QSort(int low, int high)//QuickSort 的辅助函数
{
    int pivotloc;
    if (low < high) {
        pivotloc = Partition(low, high);
        QSort(low, pivotloc - 1);
        QSort(pivotloc + 1, high);
    }
}

void QuickSort()//快速排序
{
    BeforeSort();
    QSort(1, Size);
    output();
}
```

```
void ShellSort()//希尔排序
{
    BeforeSort();
    int i, j, h;
    i = 4;
    h = 1;
    while (i <= Size) {
        i = i * 2;
        h = 2 * h + 1;
    }
    while (h != 0) {
        i = h;
        while (i <= Size) {
            j = i - h;
            while (j > 0 && Less(j + h, j)) {
                Swap(j, j + h);
                j = j - h;
            }
            i++;
        }
        h = (h - 1) / 2;
    }
    output();
}

void Sift(int left, int right)//堆排序的调堆函数
{
    int i, j, finished = 0;
    i = left;
    j = 2 * i;
    Shift(data1, data1, 0, left);
    Shift(data1, data1, MAXSIZE + 1, left);
```

```
    while (j <= right && !finished) {
        if (j < right&&Less(j, j + 1)) j = j + 1;
        if (!Less(0, j)) finished = 1;
        else {
            Shift(data1, data1, i, j);
            i = j;
            j = 2 * i;
        }
    }
    Shift(data1, data1, i, MAXSIZE + 1);
}

void HeapSort()//堆排序
{
    int left, right;
    BeforeSort();
    for (left = Size / 2; left >= 1; left--) Sift(left, Size);
    for (right = Size; right >= 2; right--) {
        Swap(1, right);
        Sift(1, right - 1);
    }
    output();
}

void BInsertSort()//折半插入排序
{
    BeforeSort();
    int i, low, high, m, j;
    for (i = 2; i <= Size; i++) {
        Shift(data1, data1, 0, i);
        low = 1;
        high = i - 1;
        while (low <= high)
```

```
        {
            m = (low + high) / 2;
            if (Less(0, m)) high = m - 1;
            else low = m + 1;
        }
        for (j = i - 1; j >= high + 1; j--) Shift(data1, data1, j + 1, j);
        Shift(data1, data1, high + 1, 0);
    }
    output();
}

void Binsort()//2-路插入排序
{
    BeforeSort();
    int i, k, j;
    int first, last;
    first = last = 1;
    Shift(R1, data1, 1, 1);
    for (i = 2; i <= Size; i++) {
        compCount++;
        if (data1[i] >= R1[1]) {
            compCount++;
            j = last;
            while (j >= 1 && R1[j] > data1[i]){
                Shift(R1, R1, j + 1, j);
                j--;
                compCount++;
            }
            Shift(R1, data1, j + 1, i);
            last++;
        }
        else {
            first--;
            if (first == 0) first = Size;
```

```
            j = first + 1;
            compCount++;
            while (j <= Size && R1[j] <= data1[i]){
                Shift(R1, R1, j - 1, j);
                j++;
                compCount++;
            }
            Shift(R1, data1, j - 1, i);
        }
    }
    k = 1;
    j = first;
    while (k <= Size) {
        Shift(data1, R1, k, j);
        k++;
        j = (j + 1) % (Size + 1);
        if (j == 0) j = j + 1;
    }
    output();
}

void Merge(int low, int m, int high)
{
    int i = low, j = m + 1, p = 1;
    while (i <= m && j <= high) {
        if (Less(i, j)) Shift(R1, data1, p++, i++);
        else Shift(R1, data1, p++, j++);
    }
    while (i <= m)
        Shift(R1, data1, p++, i++);
    while (j <= high)
        Shift(R1, data1, p++, j++);
    for (p = 1, i = low; i <= high; p++, i++)
        Shift(data1, R1, i, p);
```

```
}

void MSort(int low, int high)
{
	int mid;
	if (low < high) {
		mid = (low + high) / 2;
		MSort(low, mid);
		MSort(mid + 1, high);
		Merge(low, mid, high);
	}
}

void MergeSort()//归并排序
{
	BeforeSort();
	MSort(1, Size);
	output();
}

void Distribute(node *a, int w)
{
	int i;
	for (i = 0; i < 10; i++) fr[i] = -1;
	for (i = head; i != -1; i = a[i].next) {
		int x = a[i].data3 / w % 10;
		if (fr[x] == -1) {
			fr[x] = re[x] = i;
			compCount++;
		}else {
			a[re[x]].next = i;
			re[x] = i;
			shiftCount++;
		}
```

```
    }
    for (i = 0; i < 10; i++)
        if (fr[i] != -1)a[re[i]].next = -1;
}

void Collect(node *a)
{
    int i, last = -1;
    for (i = 0; i < 10; i++)
        if (fr[i] != -1) {
            if (last == -1) {
                head = fr[i];
                last = re[i];
            }else {
                a[last].next = fr[i];
                last = re[i];
                shiftCount++;
            }
        }
    a[last].next = -1;
}

void RadixSort()//基数排序算法。
{
    BeforeSort();
    ofstream out_stream;
    node* a;
    a = new node[Size];
    int i, j = 1;
    for (i = 0; i < Size; i++) {
        a[i].data3 = data1[i + 1];
        a[i].next = i + 1;
    }
    head = 0;
```

```
    a[Size - 1].next = -1;
    for (i = 1; i <= max; i *= 10) {
        Distribute(a, i);
        Collect(a);
    }
    cout << "\t" << compCount << "\t\t" << shiftCount << "\n";
    while (head != -1){
        data1[j++] = a[head].data3;
        head = a[head].next;
    }
    CopyData(data1, data2);
    cout << "\n";
    out_stream.open("output.txt", ios::app);
    out_stream << "\t" << compCount << "\t\t" << shiftCount << "\n\n";
    out_stream.close();
}

void Initialization()//系统初始化
{
    system("cls");//清屏
    cout
        << "************************************************************\n"
        << "*            《内部排序算法的比较》                        *\n"
        << "************************************************************\n"
        << "*        【1】由系统随机产生待排序表                       *\n"
        << "*        【2】手动输入待排序表                             *\n"
        << "*        【3】返回主菜单                                   *\n"
        << "*        【0】退出程序                                     *\n"
        << "*                                          v 2018.7.2      *\n"
        << "************************************************************\n"
        << endl << "请输入要执行的步骤：";
}
```

```
void Interpret(int cmd)//调用各个算法
{
    int i, j, x = 2;
    ofstream out_stream;
    out_stream.open("output.txt", ios::app);
    if (out_stream.fail())
    {
        cout << "Output file opening failed.\n";
        exit(1);
    }
    switch (cmd) {
    case 1:
        out_stream << "由系统随机产生待排序表的各个算法的比较次数和移动次数如下:\n";
        out_stream << "\tcompCount\tshiftCount\n";
        out_stream.close();
        cout << "请输入待排序表的长度:";
        cin >> Size;
        RandomizeList();
        cout << "\t"; for (i = 1; i <= Size; i++) cout << data1[i] << " "; cout << "\n";
        cout << "由系统随机产生待排序表的各个算法的比较次数和移动次数如下:\n";
        cout << "\tcompCount\tshiftCount\n";
        for (j = 0; j < SORTNUM; j++) {
            CopyData(data2, data1);
            out_stream.open("output.txt", ios::app);
            if (j == 0) { cout << "Bubbl: "; out_stream << "Bubbl: "; out_stream.close(); BubbleSort(); }
            if (j == 1) { cout << "Tnser: "; out_stream << "Tnser: "; out_stream.close(); InsertSort(); }
            if (j == 2) { cout << "Selec: "; out_stream << "Selec: "; out_stream.close(); SelectSort(); }
            if (j == 3) { cout << "Quick: "; out_stream << "Quick: "; out_stream.close();
```

```
QuickSort(); }
                if (j == 4) { cout << "Shell: "; out_stream << "Shell: "; out_stream.close();
ShellSort(); }
                if (j == 5) { cout << "Heap : "; out_stream << "Heap : "; out_stream.close();
HeapSort(); }
                if (j == 6) { cout << "BInse: "; out_stream << "BInse: "; out_stream.close();
BInsertSort(); }
                if (j == 7) { cout << "Merge: "; out_stream << "Merge: "; out_stream.close();
MergeSort(); }
                if (j == 8) { cout << "Bin : "; out_stream << "Bin : "; out_stream.close();
Binsort(); }
                if (j == 9) { cout << "Radix: "; out_stream << "Radix: "; out_stream.close();
RadixSort(); }
            }
            for (i = 1; i <= Size; i++) cout << data1[i] << " ";
            cout << "\n 按任意键继续： ";    while (x--)cin.get();
            Initialization();
            break;
        case 2:
            cout << "请输入待排序表的长度： ";
            cin >> Size;
            cout << "请输入" << Size << "个数据:\n";
            for (i = 1; i <= Size; i++) cin >> data1[i];
            CopyData(data1, data2);
            out_stream << "手动输入待排序表的各个算法的比较次数和移动次数如
下:\n";
            out_stream << "\tcompCount\tshiftCount\n";
            out_stream.close();
            cout << "手动输入待排序表的各个算法的比较次数和移动次数如下:\n";
            cout << "\tcompCount\tshiftCount\n";
            for (j = 0; j < SORTNUM; j++) {
                CopyData(data2, data1);
                out_stream.open("output.txt", ios::app);
```

```
                if (j == 0) { cout << "Bubbl: "; out_stream << "Bubbl: "; out_stream.close();
BubbleSort(); }
                if (j == 1) { cout << "Tnser: "; out_stream << "Tnser: "; out_stream.close();
InsertSort(); }
                if (j == 2) { cout << "Selec: "; out_stream << "Selec: "; out_stream.close();
SelectSort(); }
                if (j == 3) { cout << "Quick: "; out_stream << "Quick: "; out_stream.close();
QuickSort(); }
                if (j == 4) { cout << "Shell: "; out_stream << "Shell: "; out_stream.close();
ShellSort(); }
                if (j == 5) { cout << "Heap : "; out_stream << "Heap : "; out_stream.close();
HeapSort(); }
                if (j == 6) { cout << "BInse: "; out_stream << "BInse: "; out_stream.close();
BInsertSort(); }
                if (j == 7) { cout << "Merge: "; out_stream << "Merge: "; out_stream.close();
MergeSort(); }
                if (j == 8) { cout << "Bin : "; out_stream << "Bin : "; out_stream.close();
Binsort(); }
                if (j == 9) { cout << "Radix: "; out_stream << "Radix: "; out_stream.close();
RadixSort(); }
            }
            for (i = 1; i <= Size; i++) cout << data1[i] << " ";
            cout << "\n 按任意键继续： ";    while (x--)cin.get();
            Initialization();
            break;
        case 0: break;
        default:Initialization(); break;
        }
}

void sort()
{
    Initialization();
    int cmd;
```

```
    do {
        cin >> cmd;
        Interpret(cmd);
    } while (cmd != 0);
}
```

## 二．递归替换问题（3）

编写程序，扩展 C/C++源文件中的#include 指令（以递归的方式）。请以 文件名的内容替换形如下面的代码行。

#include “filename”

### 1．采用类语言定义相关的数据类型

```
typedef struct
{
    char name[rMAX][rMAX];              //存文件名
    char str[rMAX][rMAX][rMAX];         //存入数据
}tmp;
```

### 2．算法设计

模拟 C/C++预处理程序的功能，打开源文件后，先检查每一行的第一个字符是否是“#”，如果不是，则原样输出这一行；如果是，则处理预处理指令，先把”#“后紧跟的预处理指令解析出来，如果是 include，则提取后面的文件名，打开文件，再递归调用这个预处理函数。

### 3．函数的调用关系图

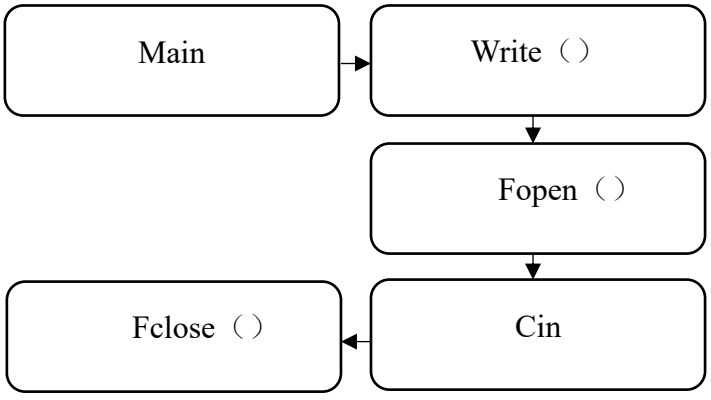


图 2-1 函数关系

### 3．调试分析

a）调试中遇到的问题及对问题的解决方法

i． 问题理解不透彻，和舍友研究了几天才搞出的方案。

ii. VS 弃用旧函数，在 C 语言的相关网站自学新函数。

b) 算法的时间复杂度和空间复杂度

i. 该算法的复杂度为 O(n³)，空间复杂度为 O(n)。

## 5. 测试结果

```
# 递归替换程序
text1.h
输入文件内容:
#include<text>
head1

text2.h
输入文件内容:
#include<text>
head2

text3.h
输入文件内容:
head3

已完成递归替换问题按0退出:
```

图 2-2 递归替换示例


图 2-3 递归替换结果

## 6. 源程序（带注释）

```
#pragma once
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<string>
#define rMAX 100
typedef struct
{
    char name[rMAX][rMAX];              //存文件名
```

```
    char str[rMAX][rMAX][rMAX];         //存入数据
}tmp;

void creatfile(tmp *s, int j);       //文件创立函数
void change(tmp *s, char *ch, int x);    //递归函数
void replace();

#include"replace.h"
using namespace std;
int h = 0;
int lens[rMAX][rMAX];
void creatfile(tmp *s, int j)       //文件创立函数
{
    int i = -1;
    errno_t err;
    FILE *fp;
    switch (j)
    {
    case 0:strcpy_s(s->name[j], "text1.h"); break;
    case 1:strcpy_s(s->name[j], "text2.h"); break;
    case 2:strcpy_s(s->name[j], "text3.h"); break;
    }
    err = fopen_s(&fp, s->name[j], "w");
    if (err)
    {
        cout << "不能打开文件!!!" << endl;
        exit(1);
    }
    cout << s->name[j] << endl;
    cout << "输入文件内容:" << endl;
    while (s->str[j + 1][i][0] != '\n')
    {
        i++;
```

```
            gets_s(s->str[j + 1][i]);
            lens[j + 1][i] = strlen(s->str[j + 1][i]);
            strcat_s(s->str[j + 1][i], "\n");
            fputs(s->str[j + 1][i], fp);
        }
        fclose(fp);
}
void change(tmp *s, char *ch, int x)    //递归函数
{
        FILE *fp1, *fp;
        int i = 0, j = 0;
        errno_t err;
        err = fopen_s(&fp1, ch, "r");
        if (err)
        {
            cout << "文件打开失败！！！ " << endl;
            exit(1);
        }

        while (s->str[x][i][0] != '\n')
        {
            fgets(s->str[x][i], lens[x][i] + 2, fp1);
            if (strcmp(s->str[x][i], "#include<text>\n") == 0){
                h++;
                if (h == 1)change(s, s->name[1], 2);
                if (h == 2)change(s, s->name[2], 3);
            }
            else
            {
                strcpy_s(s->name[3], "text4.h");//文件四的建立
                err = fopen_s(&fp, s->name[3], "a+");
                if (err){
                    cout << "文件打开失败！！！ " << endl;
```

```
                    exit(1);
                }
                fputs(s->str[x][i], fp);
                fclose(fp);
            }
        }
        fclose(fp1);
}
void replace()
{
        system("cls");
        cin.clear();
        while (cin.get() != '\n') {}
        cout << "#  递归替换程序" << endl;
        tmp s;
        int i, x;
        for (i = 0; i < 3; i++)creatfile(&s, i);
        change(&s, s.name[0], 1);
        cout<<"已完成递归替换问题按 0 退出：";
        cin >> x;
}
```

# 三．跳马问题（3）

要求在 64 个国际象棋格子，任意位置放一个马，如何不重复地把格子走完。

## 1．采用类语言定义相关的数据类型

```
typedef struct
{
    int board[8][8]; //定义一个 int 的二位数组 board
    int step;/*走的步数*/
}SeqList;
int deltar[] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int deltac[] = { 1,   2, 2, 1,-1,-2, -2, -1 };
```

## 2．算法设计

对于跳马问题，一般情况下，可以采用回溯法找解，但对于该问题来说，只要找到一种解法即可。

本程序使用了回溯法，对下一步进行判断后取最优位置，提高程序的效率。

## 3．函数的调用关系图

本子程序共有四个函数，其中 Horse（）为主函数，

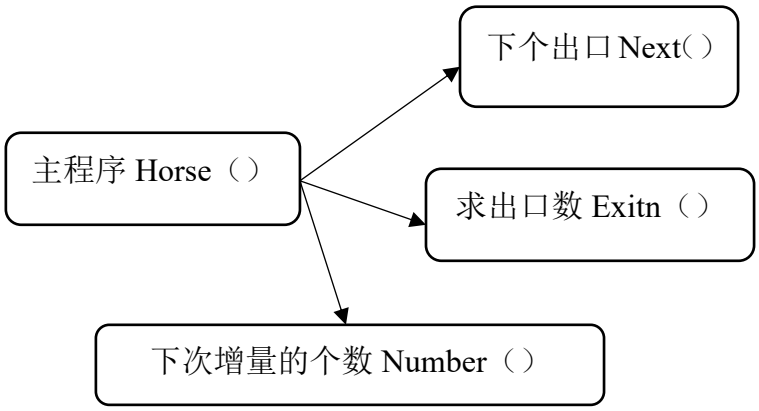


图 3-1 函数调用关系图

## 4．调试分析

a、跳马问题理解不透彻，上维基百科了解了一下这个马踏棋盘；

b、试了好多种方案，有些时间效率太低，完全跑不出来；

c、算法的时间复杂度和空间复杂度

时间复杂度：$O(n^2)$；空间复杂度：O（n）。

## 5．测试结果

```
请输入马的位置(x y):2 4
5       8       3       24      47      10      51      64
2       23      6       9       52      63      48      11
7       4       25      46      27      50      59      62
22      1       28      53      58      61      12      49
29      18      35      26      45      54      57      60
34      21      32      41      56      39      44      13
17      30      19      36      15      42      55      38
20      33      16      31      40      37      14      43

按零键退出，其他键继续
```

图 3-2 跳马运行示例

## 6．源程序（带注释）

```
#pragma once
#include<iostream>
#include<stdio.h>
typedef struct
{
    int board[8][8]; //定义一个 int 的二位数组 board
    int step;/*走的步数*/
}SeqList;
int exitn(int r, int c, int nexta[]);
int number(int r, int c);
int next(int r, int c);
void horse();

#include"horse.h"
SeqList L;
int board[8][8];
int start;
```

```
//马的可能走法对当前位置的横纵增量
int deltar[] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int deltac[] = { 1,    2, 2, 1,-1,-2, -2, -1 };
int exitn(int r, int c, int nexta[])
{
    //求马所在位置(r,c)的出口数,nexta[]为出口号
    int i, k, a, b;
    int num = 0;
    for (i = 0; i <= 7; i++) {
        k = (i + start) % 8;
        a = r + deltar[k];
        b = c + deltac[k];
        if (a <= 7 && a >= 0 && b >= 0 && b <= 7 && L.board[a][b] == 0) {
            nexta[num] = k;
            num++;
        }
    }
    return num;
}

int number(int r, int c)
{
    //返回下次可以找到的增量的个数
    int i, k, a, b;
    int num = 0;
    for (i = 0; i <= 7; i++) {
        k = (i + start) % 8;
        a = r + deltar[k];
        b = c + deltac[k];
        if (a <= 7 && a >= 0 && b >= 0 && b <= 7 && L.board[a][b] == 0) {
            num++;
        }
    }
```

```
    return num;
}

//选择下一个出口
int next(int r, int c)
{
    //min 用来确定最少出口数
    int nexta[8], num, num1 = 0, min, i, k;
    min = 9;
    num = exitn(r, c, nexta);//num 为(r,c)的出口个数
    if (num == 0)return -1;    //没有出口
    for (i = 0; i <= num - 1; i++) {
        //  找出口最少的点
        num1 = number(r + deltar[nexta[i]], c + deltac[nexta[i]]);
        if (num1 <= min){
            min = num1;
            k = nexta[i];
        }
    }
    return k;//返回出口最少的点
}

void horse()
{
    system("cls");
    int ber, bec;//用于保存输入初始的行列
    int r, c;//坐标
    std::cout << "请输入马的位置(x y):";
    std::cin >> bec >> ber;
    if (ber > 0 && ber-- < 9 && bec>0 && bec-- < 9) {
        start = 0;
        while (start <= 7)//进行测试的次数
```

```
        {
            do {
                for (int i = 0; i <= 7; i++)
                    //初始化 board 数组
                    for (int j = 0; j <= 7; j++)L.board[i][j] = 0;
                r = ber;//将当前值赋给 r
                c = bec;//将当前值赋给 c
                L.board[r][c] = 1;//将初始位置赋值为 1
                for (L.step = 2; L.step <= 64; L.step++) {
                    //此路不通
                    int k = next(r, c);
                    if (k == -1) {
                        start++;
                        break;//进行下一次测试
                    }
                    r += deltar[k];
                    c += deltac[k];
                    L.board[r][c] = L.step;
                }
            } while (L.step <= 64);
            //如果步数大于棋盘的格子数，则表明遍历成功
            if (L.step > 64){
                std::cout << std::endl;
                for (int i = 0; i <= 7; i++){
                    for (int j = 0; j <= 7; j++)std::cout << L.board[i][j] << "\t";//
printf("%4d ", L.board[i][j]);
                    std::cout<<"\n"<<std::endl;
                }
                start++;
                break;
            }
        }
    }
```

```
}
```

# 四．长整数运算问题（4）

设计程序，实现两个任意长的整数的加、减、乘运算问题。

## 1．采用类语言定义相关的数据类型

```
//定义长整数类
class LongInt
{
private:
    //存储正负
    int sign = 1;
    //存储数值
    std::string number;
};
```

## 2．算法设计

长整数运算的符号运算法则和整数符号的四则运算相同，所以选用整型保存大数的符号，并进行运算。

字符串是特殊的字符数组，可以存储很长的数据。不考虑内部返回的 size()大小，利用 C++ 中的 auto 自动推断 int，long 等整数存储类型，保证了空间的有效利用。长整数表示可根据生产需求适当修改相关的流操作，此处统一使用连续表示。

所有的相关运算符一律重载，包括加减乘除、大小比较、输入输出流。

加、减、乘运算采用了小学课本的计算法则，为便于运算先对字符串进行了逆置，运算完成后再逆置回。

如果要提高计算效率可进行分组计算，考虑到代码量和时间的问题，该程序不作分组处理。

由于除法未作要求，所以除法使用了减法统计次数的方法获取相关值——商、余数、模，大多电脑每秒减 1000 次，所以当在 1000 倍以上时此法不可用于生产环境。

## 3．函数的调用关系图

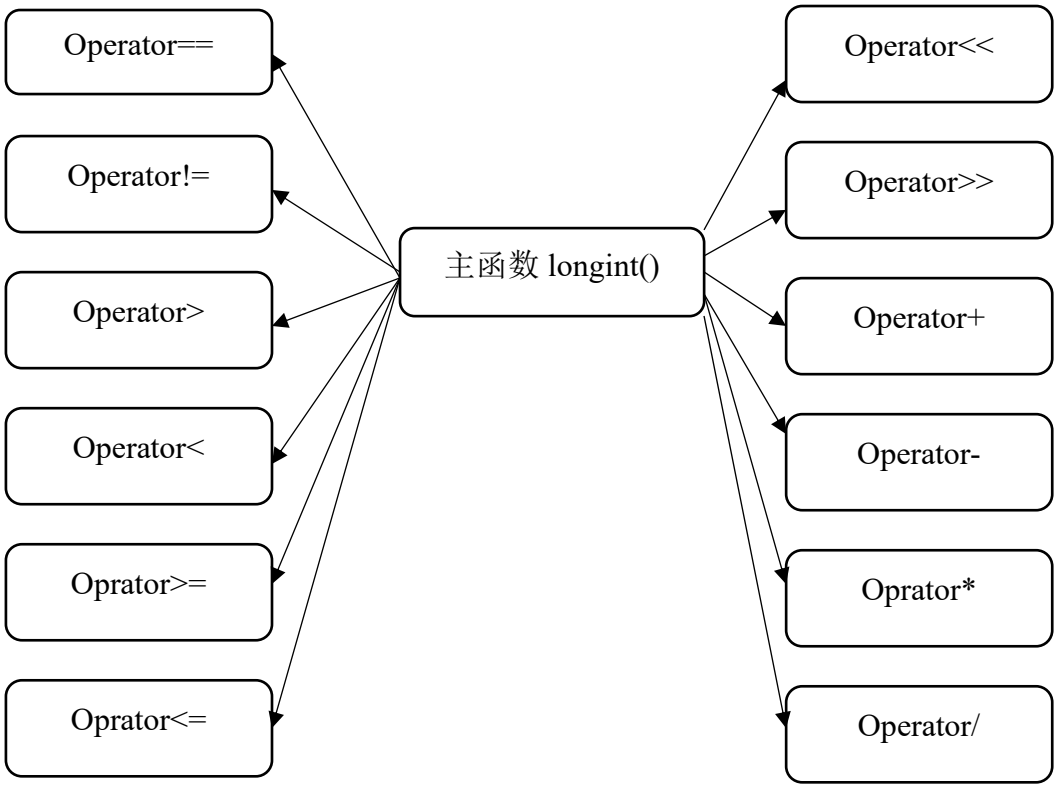


图 4-1 函数调用关系图

## 4．调试分析

c) 字符和数字混用，调试程序时经常崩溃；

d) 输入的数据未作校验处理，导致容易发生崩溃；

e) 除法程序由于使用了减法叠加统计，导致运算效率极低；

f) 时间复杂度 $O(n^2)$，空间复杂度 $O(n)$。

## 5．测试结果

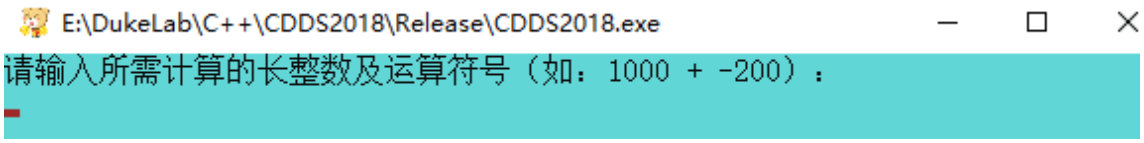


图 4-2 程序初始界面

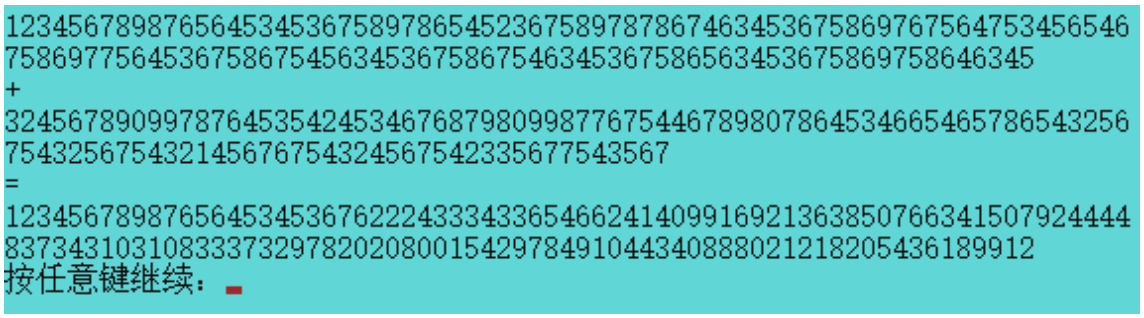


图 4-3 长整数加法

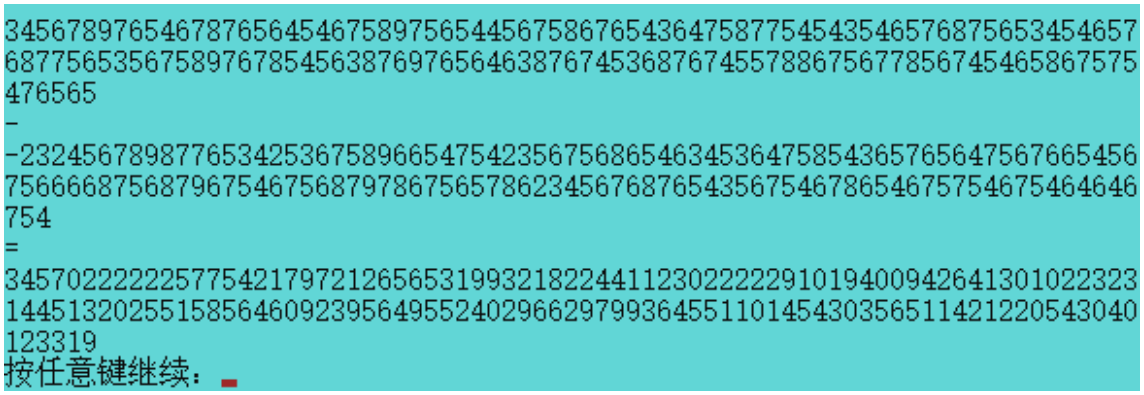


图 4-4 长整数减法

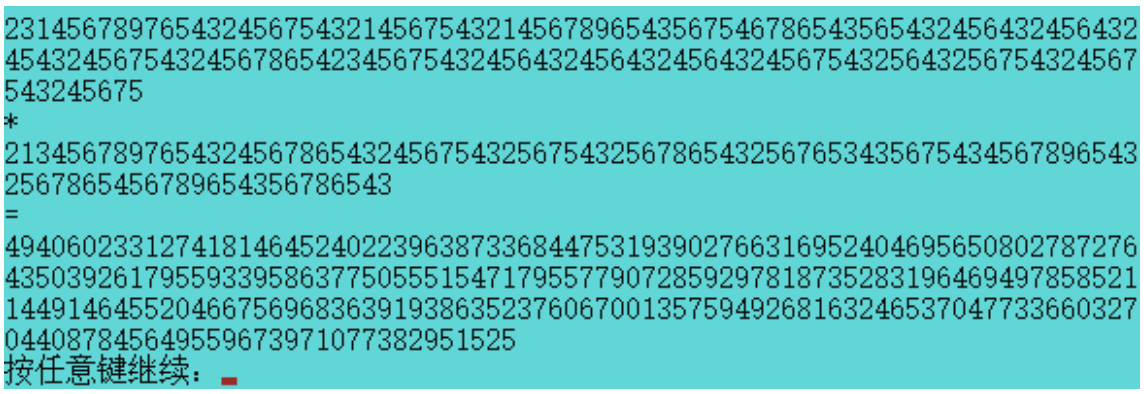


图 4-5 长整数乘法

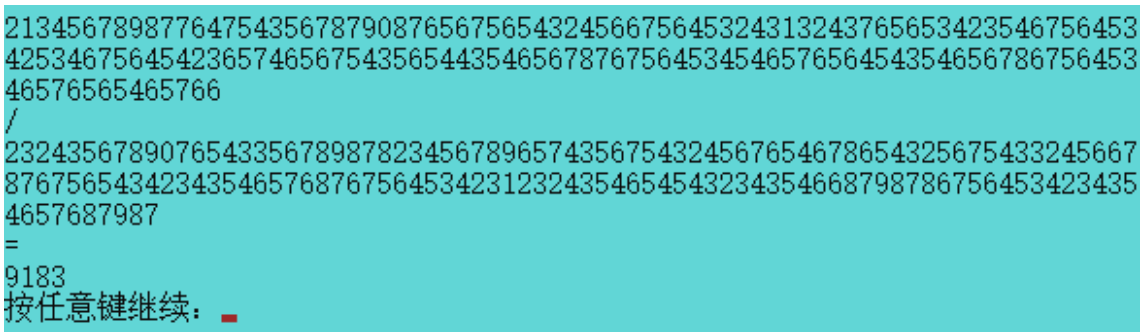


图 4-6 长整数除法

## 6. 源程序（带注释）

```
#pragma once
```

```
#include<iostream>
#include<string>
//长整数
class LongInt
{
private:
    int sign = 1;//正负
    std::string number;//数值
    bool isbig(std::string& l, std::string& r);
    bool compare(LongInt lnum, LongInt rnum);
public:
    LongInt(std::string str="0");
    auto size() { return number.size(); }
    LongInt operator+(LongInt num);
    LongInt operator-(LongInt num);
    LongInt operator*(LongInt num);
    LongInt operator/(LongInt num);
    //LongInt operator%(LongInt num);
    const bool& operator==(const LongInt &num);
    const bool& operator!=(const LongInt &num);
    const bool& operator >(const LongInt &num);
    const bool& operator>=(const LongInt &num);
    const bool& operator <(const LongInt &num);
    const bool& operator<=(const LongInt &num);
    //输出长整数
    friend std::ostream& operator<<(std::ostream& os, const LongInt &num);
    //输入长整数
    friend std::istream& operator>>(std::istream& is, LongInt &num);
    ~LongInt();
};

void longint();
```

```
#include "LongInt.h"
//判断左字符串是否大于右字符串
bool LongInt::isbig(std::string& l, std::string& r)
{
	if (l.size() == r.size())
		return l > r;
	return l.size() > r.size() ? true : false;
}

//左值大 Ture，相等、小 false
bool LongInt::compare(LongInt lnum, LongInt rnum)
{
	if (lnum.sign == rnum.sign) {
		if (lnum.sign > 0) {
			if (isbig(lnum.number, rnum.number))return true;
		}
		else if (isbig(rnum.number, lnum.number))return true;
	}
	else if (lnum.sign > rnum.sign)return true;
	return false;
}

LongInt::LongInt(std::string str)
{
	if (str.front() == '-') {
		sign = -1;
		str.erase(0, 1);
	}
	else if (str.front() == '+')str.erase(0, 1);
	number = str;
}

LongInt LongInt::operator+(LongInt num)
```

```
{
    LongInt lint;
    std::string l, r, small;
    auto lsize = (*this).size(), rsize = num.size();
    auto max = lsize > rsize ? lsize : rsize;
    auto temp = 0;
    unsigned i = 0;
    if (sign == num.sign) {
        std::copy(number.crbegin(), number.crend(), std::back_inserter(l));
        std::copy(num.number.crbegin(), num.number.crend(), std::back_inserter(r));
        lint.sign = sign;
        while (temp || i < max) {
            temp += i < lsize ? l[i] - '0' : 0;
            temp += i < rsize ? r[i] - '0' : 0;
            lint.number += '0' + temp % 10;
            temp /= 10;
            i++;
        }
        l = lint.number.erase(0, 1);
        lint.number.clear();
        std::copy(l.crbegin(), l.crend(), std::back_inserter(lint.number));
    }
    else {
        if (num.number == number)return lint;
        lint = isbig(number, num.number) ? *this : num;
        small = lint.sign == sign ? num.number : number;
        auto min = small.size();
        while (max > i) {
            if (min < ++i)break;
            temp = max >= i ? lint.number[max - i] - '0' : 0;
            while (temp < small[min - i] - '0') {
                int count = 1;
                while (max - i >= count) {
                    if (lint.number[max - i - count] > '0') {
```

```
                        lint.number[max - i - count]--;
                        temp += std::pow(10, count);
                        break;
                    }
                    count++;
                }
            }
            temp -= small[min - i] - '0';
            for (int count = 0; temp > 0 || count == 0; count++) {
                lint.number[max - i - count] = temp % 10 + '0';
                temp = temp / 10;
            }
        }
    }
    while (lint.number[0] == '0')lint.number.erase(0, 1);
    return lint;
}

LongInt LongInt::operator-(LongInt num)
{
    num.sign = num.sign > 0 ? -1 : 1;
    return *this + num;
}

LongInt LongInt::operator*(LongInt num)
{
    LongInt lint;
    lint.sign = sign * num.sign;
    auto lsize = (*this).size();
    auto rsize = num.size();
    int temp = 0, k = 1;
    for (int i = 0; i < lsize; i++) {
        for (int j = 0; (j < rsize || temp); j++) {
            temp += j < rsize ? (number[lsize - i - 1] - '0')*(num.number[rsize - j - 1] -
```

```
'0') : 0;
                if (j + i < k) {
                    temp += lint.number[j + i] - '0';
                    lint.number[j + i] = '0' + temp % 10;
                }
                else {
                    lint.number += '0' + temp % 10;
                    k++;
                }
                temp /= 10;
            }
        }
        std::string str=lint.number;
        lint.number.clear();
        std::copy(str.crbegin(), str.crend(), std::back_inserter(lint.number));
        if(lint.number=="0")lint.sign = 1;
        return lint;
}

//num 为零报错
LongInt LongInt::operator/(LongInt num)
{
        LongInt li;
        if (*this == li)return li;
        LongInt    max(number), min(num.number), add("1");
        while (max >= min)
        {
            max = max - min;
            li = li + add;
        }
        li.sign = sign * num.sign;
        return li;
}
```

```
const bool & LongInt::operator==(const LongInt & num) {
	return number == num.number&&sign == num.sign ? true : false;
}

const bool & LongInt::operator!=(const LongInt & num) {
	return number == num.number&&sign == num.sign ? false : true;
}

const bool & LongInt::operator>(const LongInt & num) {
	return compare(*this, num);
}

const bool & LongInt::operator>=(const LongInt & num) {
	return !compare(num, *this);
}

const bool & LongInt::operator<(const LongInt & num) {
	return compare(num, *this);
}

const bool & LongInt::operator<=(const LongInt & num) {
	return !compare(*this, num);
}

LongInt::~LongInt()
{
}

std::ostream & operator<<(std::ostream & os, const LongInt & num)
{
	if (num.sign < 0)os << '-';
	os << num.number;
	return os;
}
```

```
std::istream & operator>>(std::istream & is, LongInt & num)
{
	num.sign = 1;
	std::string str;
	is >> str;
	if (str.front() == '-') {
		num.sign = -1;
		str.erase(0, 1);
	}
	else if (str.front() == '+')str.erase(0, 1);
	num.number = str;
	return is;
}

void longint()
{
	system("cls");
	LongInt lnum, rnum, num;
	char ch;
	int x = 2;
	std::cout << "请输入所需计算的长整数及运算符号（如：1000 + -200）：" <<
std::endl;
	std::cin >> lnum >> ch >> rnum;
	switch (ch)
	{
	case '+':num = lnum + rnum;break;
	case '-':num = lnum - rnum;break;
	case '*':num = lnum * rnum;break;
	case '/':num = lnum / rnum;break;
	default:
		break;
	}
	std::cout << std::endl << lnum
```

```
            << std::endl << ch
            << std::endl << rnum
            << std::endl << '='
            << std::endl << num;
    std::cout << "\n 按任意键继续：";    while (x--)std::cin.get();
}
```

# 总　结

就编写的程序而言，虽然能达到预期的结果，但在运行时所需的时间比较长，而且总体结构还不够简洁，不太容易去理解。许多问题还需要继续研究，许多技术还需要更多的改进。去图书馆借了不少书，也去网上看了些资料，只是对大概的知识有了点了解，但还是很难着手于写代码，后来就按照老师说的，先搞清楚原理，再考虑如何去实现！后来又去上网查看相关资料，又到图书馆借了很多书看，总算有头绪了。但在调试过程中，还是遇到了很多困难，后来通过了很多同学的帮助才把问题解决了。

通过这次的课程设计，让我更好地掌握了递归思想、链表、拓扑排序以及一维数组等知识，这对我以后的学习生涯以及将来步入社会起到很大的帮助。这次课程设计虽然花了我很多时间和精力，但很值得，因为它对我能力提高起到很大帮助。这次课程设计也提醒我以前知识的匮乏，它给我敲响了警钟，让我意识到自己基础的不扎实。同时在这次课设中，我还学会了如何修改错误。

总的来说，这次课设让我学到了很多很多的东西，让我对未来的工作有了信心，也有了一定的了解。

# 参考文献

[1]严蔚敏，吴伟民.《数据结构（C 语言版）》. 清华大学出版社.

[2]严蔚敏，吴伟民.《数据结构题集（C 语言版）》. 清华大学出版社.

[3]《DATA STRUCTURE WITH C++》. William Ford，William Topp . 清华大学出版社（影印版）.

[4]谭浩强.《c 语言程序设计》. 清华大学出版社.

# 致　谢

课程设计终于告一段落了，努力过后，颇有收获，很多以前不清楚、不熟悉的内容都在这一周的努力中得到了锻炼，感谢老师给予的大量帮助及指导，感谢同学们的帮助，才让我顺利完成了这次的课程设计！通过他们的帮助，我深刻体会到，做程序设计需要大量的查阅相关的资料，并与他人合作才能编写出一段好的程序。

在此，衷心的感谢包老师的辛勤指导，让我认识这个课题、熟悉这个课题并且最后完成这个课题；感谢同学们的互帮互助，提供那么多经典程序供我参考，并且指出我编程过程中出现的许多问题；感谢每个给过我帮助的人员！ 谢谢你们的支持，谢谢你们！