**第一章实验报告**

班级：

姓名：

学号：

# 一、实验目的

掌握Visual Studio控制台工程项目的建立；温习C++代码逻辑错误调试；掌握在给定代码中进行适当修改的方法；设计简单的数值计算代码。

# 二、实验内容

1.运行《实验准备(线性表)》程序，读懂程序代码；

2.运行《实验准备(graph邻接矩阵)》程序，读懂程序代码；

3.运行《求多项式A(x) 》程序，完成代码填空。

# 三、实验环境

Windows 10, Visual Studio 2017。

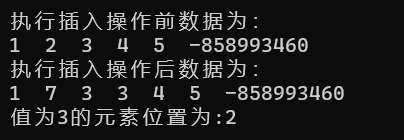
# 四、实验要求

1. 找出《实验准备(线性表)》中的逻辑错误，修改并调试。
2. 对《实验准备(graph邻接矩阵)》，修改输出边的功能：（1）在头文件.h中增加void PutArc();（2）在程序文件.cpp中实现void PutArc(); （3）在主程序中调用 PutArc()。
3. 对《求多项式A(x) 》，写出2种不同的求解代码。

# 五、报告内容

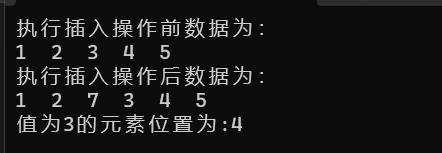
1. **修改语法错误，修改代码逻辑错误，调试并运行《实验准备(线性表)》代码，(a)给出错误结果截图; (b) 给出正确结果截图：具体说明对代码的修改**

**(a)**



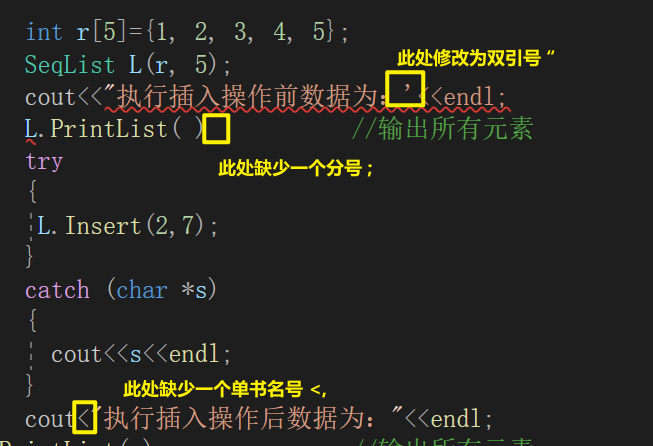
（仅修改main文件语法错误后的输出结果）

**(b)**

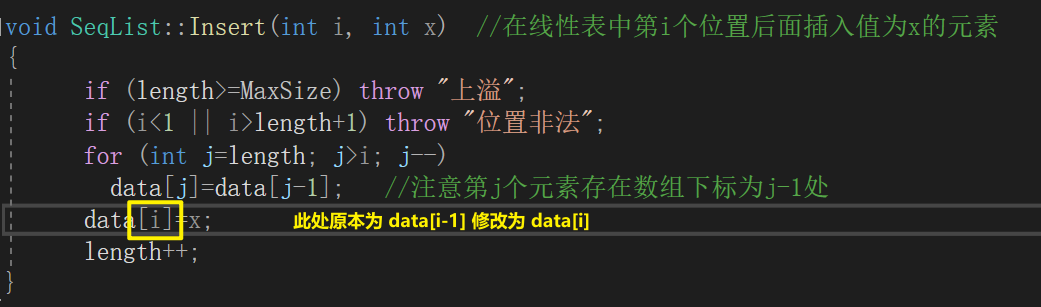


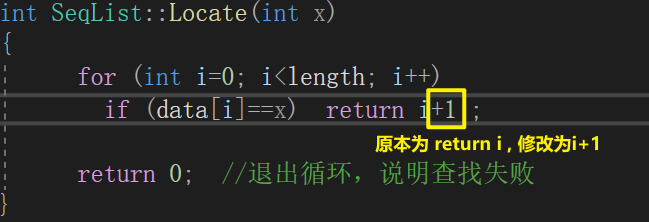
**代码的修改说明：**

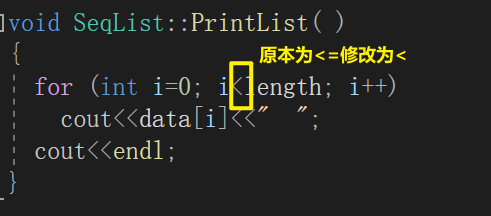
1.main.cpp修改部分



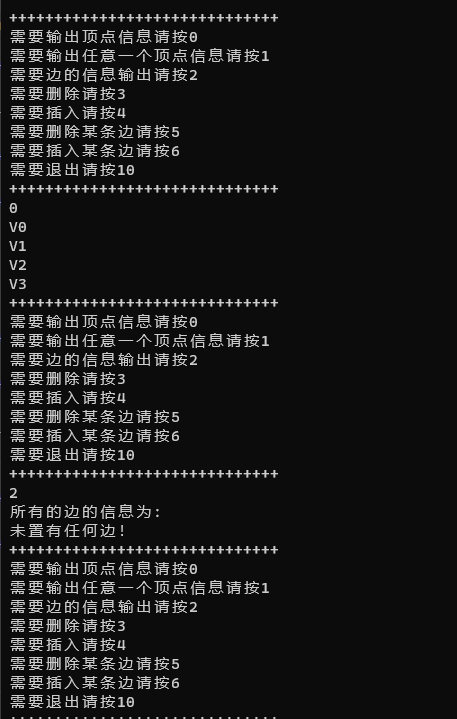
2.Seqlist.cpp修改部分



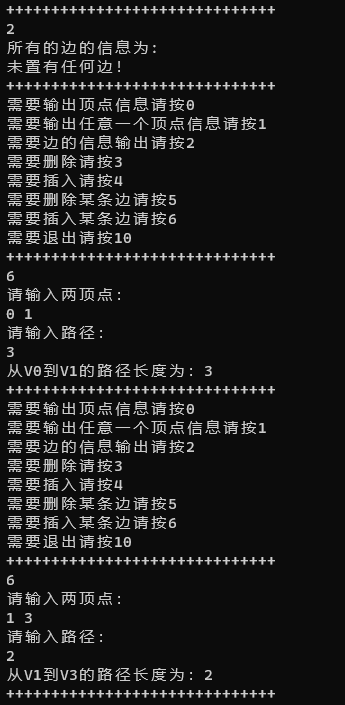
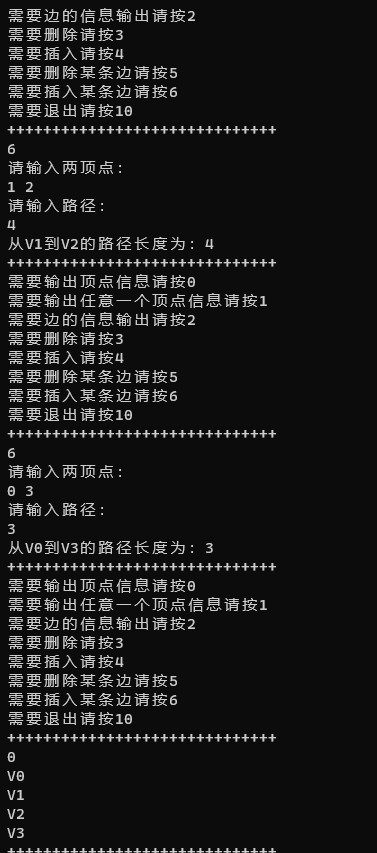




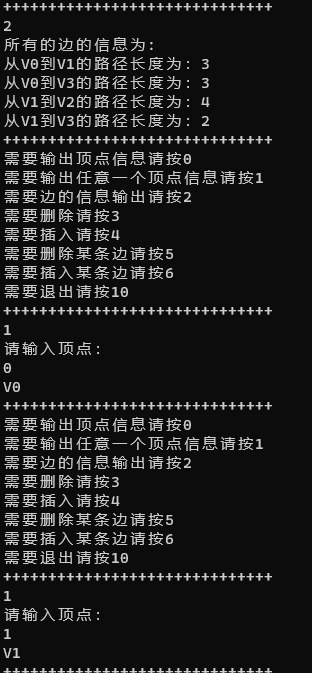
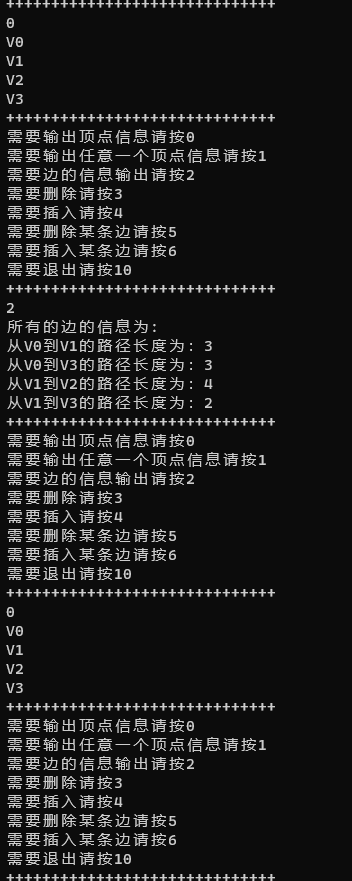
1. **《实验准备(graph邻接矩阵)》，修改输出边的功能，在已有顶点之间新增一条边，新增一个顶点V5，在新顶点和已有顶点间插入一条边，给出运行截图。**



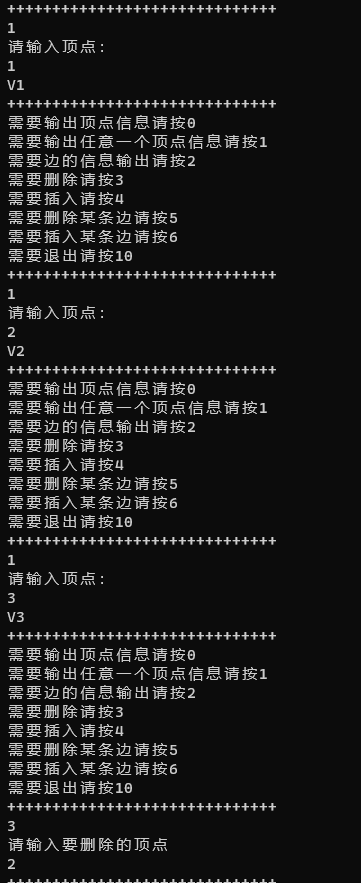
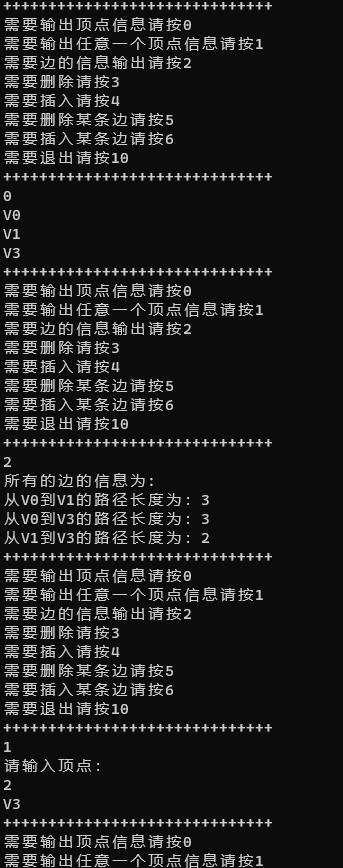
(1)

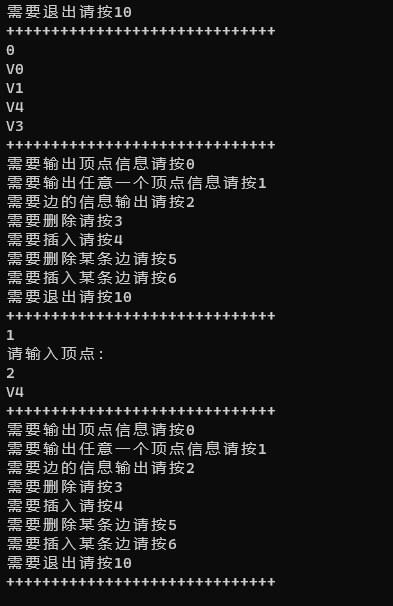
(2) (3)

(4) (5)

(6) (7)



(8)

**void MGraph<T>::PutArc()实验代码：**

template<class T>

void MGraph<T>::PutArc()

{

int i, j;

int flag = 0;

for (i = 0; i < vertexNum; i++) {

for (j = i + 1; j < vertexNum; j++){

if (arc[i][j] != 0) {

cout << "从" << vertex[i] << "到" << vertex[j] << "的路径长度为：" << arc[i][j] << "\n";

flag=1;

}

}

}

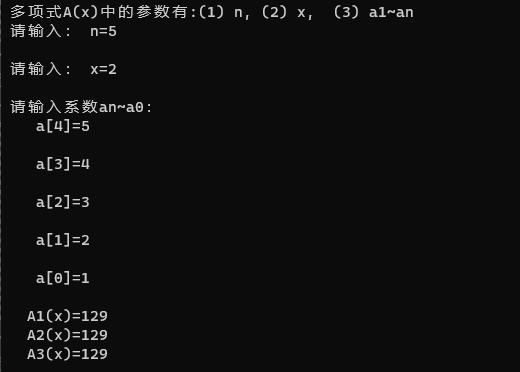
if (flag == 0) {

cout << "未置有任何边！"<< "\n";

}

}

1. **对《求多项式A(x) 》，设置n=5，x=2，并设系数，给出修改后的运行截图。**



**float Ax1(float A[],int n, float x)代码：**

float Ax1(float A[],int n, float x)

{

float result=0.0;

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

{

result+= A[i] \* pow(x, i);

}

return result;

}

**float Ax2(float A[],int n, float x)代码：**

float Ax2(float A[],int n, float x)

{

float result = 0;

float X = 1;

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

{

result += A[i] \* X;

X = X \* x;

}

return result;

}

**float Ax3(float A[],int n, float x)代码：**

float Ax3(float A[],int n, float x)

{

float result = A[n - 1];

for (int i = n - 2; i >= 0; i--) {

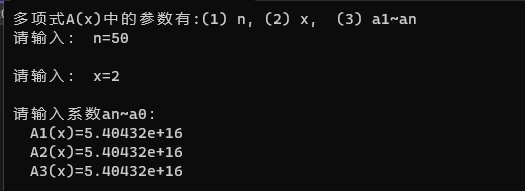
result = result \* x + A[i];

}

return result;

}

可选（加分）：设置n=50，x=2，自动设置系数，分析两种求解方法的运行时间，给出运行截图和相应代码。



（超出了float 类型的最大表示范围所以运行结果并不正确）

理论上讲第一个函数使用了幂函数的操作，所以应该是第一个函数需要的时间最长 ，第三个应该最短。

添加了一个记录运行时长的函数记录函数运行结束所需的总时长

#include <chrono> // 引入时间记录相关的头文件

#include <functional>

using namespace std::chrono;

long long measureExecutionTime(function<void()> func)

{

auto start = high\_resolution\_clock::now();

func(); // 执行传入的函数

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(stop - start);

return duration.count();

}

Main函数部分输出修改为

long long time1 = measureExecutionTime([&]()

{

cout << " A1(x)=" << Ax1(B, n1, x1) << endl;

});

long long time2 = measureExecutionTime([&]()

{

cout << " A2(x)=" << Ax2(B, n1, x1) << endl;

});

long long time3 = measureExecutionTime([&]()

{

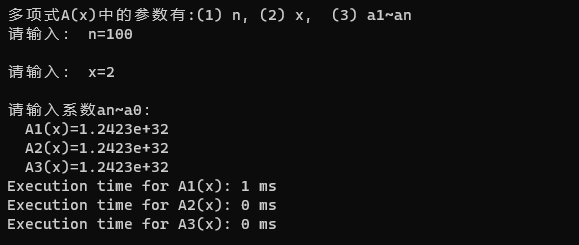
cout << " A3(x)=" << Ax3(B, n1, x1) << endl;

});

cout << "Execution time for A1(x): " << time1 << " ms" << endl;

cout << "Execution time for A2(x): " << time2 << " ms" << endl;

cout << "Execution time for A3(x): " << time3 << " ms" << endl;



（50的最高次幂无法准确看出各个函数之间的区别，把n换成了100）