面向对象程序设计
（08305121，5 学分）实验安排

第 1 周 实验一

1.1 一元二次方程求解函数设计

对于一元二次方程$ax^2+bx+c=0$（不考虑$a=0$的退化情形，即$a\neq 0$且无须判断），要求设计函数实现方程求解。有如下设计原则。

① 认真分析问题，将其数字化（例如：一元二次方程与变元无关，由 3 个系数决定方程的类型，……），确定求解算法（利用求根公式，采取直接法）。

② 采用多文件结构，合理使用头文件和源程序文件；不使用全局变量。

③ 计算与 I/O 分开。

即设计专门的函数用于方程求解。所需数据全部从参数传入，计算结果可利用函数返回类型返回，或者利用有传出功能的形参“返回”。方程求解函数中不出现 I/O 语句。

另设计专门的函数输出结果。测试数据可从键盘输入，也可事先在程序中用数组或多维数组存放。例如：

```
double a[][3] = { {1, 2, -3}, {1, 2, 1}, {1, 2, 3} };
int n = sizeof(a)/sizeof(*a);
```

便可以表示 n（n 为 3）个方程$x^2+2x-3=0$、$x^2+2x+1=0$和$x^2+2x+3=0$。

④ 要求用尽可能多的方法设计一个个函数，它们都分别能实现一元二次方程求解（至少三种）。最后编写测试函数对它们分别进行测试。建议充分利用已有的函数（例如：用求根公式编写一个函数，然后被其他函数调用）。

1.2 C-字符串处理

阅读 C-String 程序，回答其中的思考题。请先阅读其中的“说明”文档。

第 2 周 实验二（单向链表设计）

2.1 函数设计基本原则（供参考）

C/C++程序是由函数组织的，程序的运行时函数之间相互调用驱动的。编写 C/C++程序主要是编写各种各样的函数，包括主函数。

我们认为，学习了 C 语言程序设计基础后，“微观程序设计”能力应该基本具备。即：能熟练处理顺序结构、分支结构、循环结构语句。从现在起，需要将注意力集中到函数的首部（函数返回类型、函数名、形式参数）设计上来，特别是函数的参数设计。

2.1.1 函数名

在符合标识符命名规则的前提下，函数名要在一定程度上反映函数的功能。同一程序中，函数名命名风格一致，且函数名不宜过长。

2.1.2 函数返回类型

(1) 如果函数确实没有需要返回的量，则返回类型设计为 **void**。

(2) 一般地，有些函数原本并不需要返回值或返回变量，在不增加程序设计负担、同时还能增加函数应用的灵活性的情况下，可以设计返回类型，返回一个数值、或返回一个变量（引用返回）。返回类型的设计应尽量保持已有函数、或运算符的相似特性。

(3) 如果可能，返回类型应**尽可能地采用引用返回**，以避免值返回时创建、销毁临时无名变量。

2.1.3 函数形式参数

(1) 形参的名称应尽可能表意，形参个数不宜过多，参数之间应相互独立。精心安排参数的顺序，并考虑是否设置默认参数。

(2) 函数所需的数据最好都从形参传入，尽量减少对全局变量的依赖（仅允许类似 cin、cout 这样的全局变量）；尽量减少对某个具体题目所设定数值的依赖，使函数具有灵活性、适应性、可移植性（例如：将数组传递给函数时，需要传递数组的“三要素”，而不要默认元素个数为某个具体值）。

(3) 根据参数是单纯地“传入”给函数，还是“传入并传回”的需求，考虑采用值传递或引用传递。运用指针、多级指针亦可以间接地“传回”函数计算的结果；运用变量（包括指针变量）的引用可以直接地“传回”函数的计算结果。

(4) 如果形参是指针，且仅用于将目标数据“传入”给函数，则应尽可能用常量指针；如果形参是引用，且仅用于将所绑定的变量或常量“传入”给函数，则应尽可能采用常量的引用。其好处有二：① 增加了数据的安全性；② 扩大了函数的适用面（能够处理真正的常量）。

(5) 如果形参是以值传递的方式传递对象，可改用传递常引用“const &”方式来传递，这样可以省去值传递对象时的拷贝构造形参对象和析构该形参对象的过程，从而提高时间效率和空间效率。

2.1.4 函数体

(1) 功能单一，不要将众多可以独立的计算、处理过程放在同一个函数中。

(2) 除专门处理 I/O 的函数外，将计算与 I/O 相分离。尽量不在执行计算的函数体内执行 I/O 操作，函数所需的数据应从参数传入给函数；函数的计算结果应从返回类型或有“传回”功能的形参“返回”。因为，不同的应用场景中 I/O 的方式方法有很大的差别（如：字符模式、图形模式）。

(3) 函数体代码规模不宜过大，要有足够的注解。

2.2 单向链表基本操作

给定构成单向链表结点的结构体 Node。编写若干函数对链表进行操作。

```
struct Node
{
   int data;          // 数据域(虽然这里仅有一个数据，但还是用数据 datum 的复数形式)
```

```
    Node *next;       // 指针域
};
```

程序 Link_int 编写了主控函数，可以边设计函数边进行测试。建议按表 2.1 中的顺序逐个设计并逐个测试函数如下 6 个函数。请将所设计的函数原型写入头文件 Link_int.h 中。在 Link_int.cpp 文件中编写函数定义。对每一个函数，给出了设计与实现中的要点，特别需要注意的问题。

表 2.1 单向链表函数设计

序号	设计说明（请考虑形参传入 in□，传回 out□中是否应该☑或☒）	
1	函 数 名	Create
	形式参数	① 链表(in□ out□) ② 存放数据域具体数据的数组三要素，两个参数:元素个数(in□ out□)，首地址(in□ out□，带默认值 NULL)
	返回类型	Node *& （引用返回链表首地址）或 **void**
	功 能	根据连续存放的数据域数组创建链表
	思考问题	① 用什么表示一条链表？即形参的数据类型是什么？ ② 对入口参数传入的实参链表有何要求？入口参数传入的链表必须完整，也可以为空链表（最好为空）。如果非空，则可以清空原来的所有结点，也可以添加到已有结点前面。 ③ 如何简便地完成操作？（提示：创建动态结点，采用头插法。让所有结点皆为动态结点，以利于 FreeList 操作的统一性） ④ 第 3 个参数带默认值有和作用？如何测试？
	函数原型	
2	函 数 名	NumNodes
	形式参数	链表(in□加以保护？ out□)
	返回类型	**int** （返回结点的个数）
	功 能	计算并返回链表的结点个数
	注意事项	遍历链表
	函数原型	
3	函 数 名	ShowList
	形式参数	链表(in□，加以保护？ out□)
	返回类型	**int**（返回输出的结点个数） 或 **void**
	功 能	输出链表各结点的数据域数据值，要求用 head -> 数据 -> 数据 -> NULL 的形式。
	注意事项	遍历链表
	函数原型	
4	函 数 名	Insert

	形式参数	① 链表(in□ out□) ② 数据域数据(in□ out□)
	返回类型	Node *（返回新插入结点的地址）
	功　　能	插入一个新结点至链表首结点前，成为新的链首结点
	注意事项	创建一个动态结点（堆结点），保证链表中的所有结点均为堆结点，便于 FreeList 统一操作。考虑原链表为空、非空两种情形的程序代码是否相同。
	函数原型	
5	函 数 名	Append
	形式参数	① 链表(in□ out□) ② 数据域数据(in□ out□)
	返回类型	Node *（返回新插入结点的地址）
	功　　能	追加一个新结点至链表尾结点后，成为新的尾结点
	注意事项	创建一个动态结点（堆结点），保证链表中的所有结点均为堆结点，便于 FreeList 统一操作。考虑原链表为空、非空两种情形的程序代码是否相同。
	函数原型	
6	函 数 名	FreeList
	形式参数	链表(in□ out□)
	返回类型	**void**
	功　　能	释放链表中的所有结点
	注意事项	本函数要求所有结点皆为动态结点（堆结点）。采用引用型参数（Node *&head）函数返回后，链表为空链表。如果形参仅仅设计成链首结点指针值传递（Node *head），虽可以成功释放所有结点，但 head 的指向依然指向原链首结点处。
	函数原型	
已经提供了如下函数原型设计、函数定义的源代码（参见程序文件）		
7	函 数 名	**operator**<<
	形式参数	① 输出设备(in☑，out☑) ② 链表（in☑加以保护，out☒）
	返回类型	ostream & （返回输出设备）
	功　　能	重载运算<<，使之能够进行链表输出（如：cout << head << endl）。
	注意事项	调用本函数时，最常用的实参是 cout。cout 是在名字空间 std 中定义的 ostream 类型的全局对象。本函数的形参 out 多数情况下是引用（绑定）cout 对象，即 out 可以是 cout 的别名。当然，out 还可以绑定文件对象，从而可以直接输出到文件。
	函数原型	ostream & **operator**<<(ostream &out, **const** Node *head);
8	函 数 名	Locate

	形式参数	① 链表（in☑，out☒） ② 数据域数值（用作查找的依据）（in☑，out☒） ③ 找到的结点序号（整型数据）（in☐传入的数据不重要，out☑） ④ 重新开始的查找/继续查找标志（bool 型，待默认值）
	返回类型	Node * （返回找到的结点的地址值，或 NULL）
	功　　能	根据给定的数值，在链表中查找结点数据域成员的数值与给定的值相等的结点。支持继续查找。
	思 考 题	① 形式参数 newsearch 的作用是什么？ ② 形式参数 num 有何作用？ ③ 为什么要将 p，k，data 设计成静态局部变量？ ④ 局部自动指针变量 temp 的作用是什么？ ⑤ 函数的返回值是什么？如何区别是否找到满足条件的结点 ⑥ 若形参 newsearch 为 false，此时形参 x 有无作用？为什么要这样设计？ 答：对于一种条件的搜索，不能穿插另外一种条件的继续搜索。因此，此时需要故意废掉 x。
	函数原型	Node *Locate(Node *head, **int** x, **int** &num, **bool** newsearch=**false**);
9	函 数 名	Save
	形式参数	① 文件名（C-字符串）（in☑加以保护，out☒） ② 链表（in☑，out☒）
	返回类型	**int**　（返回写入文件的结点数据域数据的个数）
	功　　能	将链表所有结点数据域数据写入指定文件名的文件，每个结点的数据占一行。
	提　　示	① 用到文件流类及其对象（参见教材第 14 章第 2 节 文件 I/O 流）。 ② 此处打开文件采用默认的文本文件方式。 ③ 对于链表结点数据域数据类型(int)，要求能进行<<和>>操作，这个要求是满足的。 ④ 输出到文件的格式要能够使 Load 函数正确地读取。
	函数原型	**int** Save(**const char** *filename, **const** Node *head);
10	函 数 名	Load
	形式参数	① 文件名（C-字符串）（in☑加以保护，out☒） ② 链表（in☑，out☑）
	返回类型	**int**　（返回从文件中读取到的结点数据域数据的个数）
	功　　能	从指定的文本文件中读取数据，作为各结点数据域数值，创建链表。
	提　　示	同 Save 函数
	函数原型	**int** Load(**const char** *filename, Node *&head);

2.3 设计任务

主控函数以及部分函数已提供了源代码，请完成剩下的 6 个函数的定义。

第 3 周 实验三（阅读程序）

3.1 链表类的层次结构

表 3.1 链表类的层次性

层次	名称	特点	说明
底	数据域	抽象化、统一化	暂不设计，用形式数据类型 T。编写代码时想象成 **int**
中	结点类	简单化	**数据成员：** T data; Node<T> *next; **成员函数：** 默认的构造函数、转换构造函数、拷贝构造函数（next 指针初始化为 NULL）； 赋值运算符函数（next 指针） **其他：**将链表类作为友类
高	链表类	泛化 (尽量拥有一定的普适性)	**数据成员：** Node<T> *head, *cur_node; **int** num; **成员函数：** 四大函数（构造函数、拷贝构造函数、析构函数、赋值运算符函数）； 标准 I/O、文件 I/O； 增、删、改、查； 当前结点定位（绝对定位、相对定位、依条件定位）。

阅读 LinkList 程序。理解单向链表类模板的总体设计。对于每个成员函数、友元函数，重点在其首部设计，理解各个参数的作用。

3.2 LinkList程序中的文件

表 3.2 文件说明

序号	文件名	说　　明
1	**LinkList.h**	**单向链表类模板全部代码。该文件可移植到其他程序使用。** **设计了结点类、链表类，数据域类由应用程序设计。数据域类的设计需要满足一些要求：** **① 四大函数；② 重载 I/O 运算符函数；③ 能够实现关系运算（因为查找、排序等的需要）。参见下面的通信录程序、银行账户流水记录程序。**
以下都是测试单向链表类模板的代码文件		
2	Main.cpp	主函数（主控函数，调用其他测试函数）
3	Test.cpp	基本数据类型测试、及链表的链表测试
4	AddressBook.cpp	用户自定义的数据类型测试（通信录）

5	Banking.h Banking.cpp BankingTest.cpp	用户自定义的数据类型，测试链表的链表。有账户链表（每个结点为一个账户的信息），而每个结点中又包含一个链表（记录账户的流水单）。
文本文件		
6	AddressBook.txt	通信录数据文件。可被程序读取，也可将链表各结点数据存盘。
7	dLink.txt	测试函数 test1 产生的数据文件（double 型数据）
8	LLink.txt	整型数据链表的链表所产生的数据文件
工程文件：MinGW Developer Studio，Code::Blocks，Dev-C++，VS2019 集成开发环境适用		

【说明】从第 4 周起，上机实验以课程小项目为主，不提供参考代码。

第 4~5 周 实验四~五（课程小项目之一：单向链表类模板的应用）

各小组自行选题，利用 LinkList 类模板设计一个链表综合应用程序。要求使用**链表的链表**。参阅第 3 周实验程序 Banking 及 AddressBook。

第 6~7 周 实验六~七（课程小项目之二、三同步进行）

小组成员角色：总体设计并组织课外研究、详细设计人员、测试人员。不同的小项目中，成员角色轮换。

1. 课程小项目之二：字符串类。具体要求如下。

(1) 可以封装 C-字符串（即含串结束标志字符'\0'），也可以不含串结束标志；

(2) 对象构造、析构、赋值相关的四大函数；

(3) 其他成员函数（如：length、c_str 等）；

(4) 重载适当的运算符（如：<<、>>、+、+=、<、<=、>、>=、==、!=、[ ]等）；

(5) 有异常处理功能（如：方括号运算时下标越界等）；

(6) 测试用例设计（含异常处理）。

2. 课程小项目之三：向量类模板。具体要求如下。

(1) 向量的维数可缩放；

(2) 对象构造、析构、赋值相关的四大函数；

(3) 其他成员函数（如：返回维数等）；

(4) 重载适当的运算符（如：<<、>>、+、+=、==、!=、[ ]等）；

(5) 有异常处理功能（如：维数不同时无法进行两个向量的和；方括号运算时下标越界）；

(6) 测试用例设计（含异常处理）。

第 8~9 周 实验八~九（课程小项目之四：抽象向量类模板及其派生类）

1. 抽象向量类模板

(1) 数据成员设计；
(2) 成员函数（包括虚函数、纯虚函数）设计等。

2. 派生向量类模板

(1) 定义纯虚函数；
(2) 设计测试用例单独测试派生类（含异常处理）。

3. 派生字符串类

(1) 定义纯虚函数；
(2) 设计测试用例单独测试派生类（含异常处理）。

4. 联合测试

设计测试用例，对两个派生类进行联合测试，以展示动态多态性。

第 10 周 实验十（机动，或 I/O、异常处理）