武汉工程大学

计算机科学与工程学院

综合设计报告

**设计名称： 面向对象综合设计**

**设计题目： 一元多项式计算器的设计与实现**

**学生学号： 2207120406**

**专业班级： 2022软件工程04班**

**学生姓名： 谭轩钊**

**学生成绩：**

**指导教师（职称）：张俊（教授）**

**学业导师（职称）：何成万（教授）**

**完成时间： 2023.11.18 至 2023.12.9**

武汉工程大学计算机科学与工程学院 制

说明：

1、报告中的第一、二、三项由课程设计负责人在课程设计开始前填写并发给每个学生。

2、学业导师负责批改学生的设计报告，并给出相应的得分。同时，就设计报告质量撰写评语。

3、指导教师就学生在设计期间的表现及设计完成情况分别给出相应的得分。同时，就此两项情况撰写评语。

4、设计的总评成绩由上述各部分累加得出，由指导教师汇总，并填写于报告的封面。

5、设计报告正文字数一般应不少于3000字，也可由课程设计负责人根据本项课程设计的具体情况酌情增加字数或内容。

6、此表格式为武汉工程大学计算机科学与工程学院提供的基本格式（适用于学院各项课程设计），各专业也可根据本项课程设计的特点及内容做适当的调整，并上报学院批准。

成绩评定表

姓名：谭轩钊 学号：2207120406 班级：2022软件工程04班

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类别 | 考核内容及分值 | 评分标准 | 实际得分 | 评分人 |
| 设  计  报  告 | 解决方案  （20分） | 课题综述充分，能够运用面向对象的思想进行问题分析，建立对象模型，拟定的解决思路和方案合理，并对多种解决方案进行比较。 |  | 学业导师： |
| 设计与实现过程描述  （40分） | 能够运用面向对象程序设计的方法和技术并综合考虑各种因素。设计与实现过程描述完整，章节组织恰当、内容翔实，格式符合规范。 |  |
| 设  计  成  果 | 开发工具使用  （20分） | 能够使用相关开发工具及现代程序库进行程序开发与设计，并能按照要求完成设计任务。 |  | 指导教师： |
| 团队协作  （20分） | 在设计过程中各团队成员对项目任务进行分工和协作，设计工作量饱满。 |  |
| 总评成绩 | | |  | |

|  |
| --- |
| 一、课程设计目的、条件、任务和内容要求：  设计目的  《面向对象综合设计》是软件工程专业学生进入专业课学习后，综合运用所学习的面向对象程序设计思想和方法、C++等语言及工具环境、算法与数据结构等知识完成较为复杂的程序设计。通过该课程设计，锻炼学生的设计开发能力和文档撰写能力，为专业培养奠定知识和能力基础。  具备条件  已经学过面向对象程序设计等一系列专业相关课程，并能够熟练应用C++语言进行设计开发。  设计任务  1．用面向对象思想和方法完成复杂工程问题的求解，完成程序系统的设计、分析和实现等过程；  2．在设计分析过程中，用到算法与数据结构的相关知识；  3．根据设计成果，完成课程设计报告的撰写。注意报告格式规范、内容详实。  设计内容  1．学生可以根据自己的实际情况在教师指定的题目中选择一个进行设计实现。原则上应综合运用上述提及的知识。  2．在程序设计和撰写报告时，请注意按照软件工程的基本过程：需求分析，系统设计，过程及算法实现，功能测试与结果分析等。  指导教师签字： 张俊  2023 年 11 月 13 日 |
| 二、进度安排：  软件设计与实现：2.5个单元时间  撰写报告：1个单元时间  项目验收/报告提交：0.5个单元时间 |
| 三、应收集资料及主要参考文献：  应收集如下几方面资料：  1．C++与面向对象程序设计等相关书籍  2．算法与标准库等相关参考书籍  主要参考文献：  [1](美)Horton, I.著; 李予敏译.C++入门经典(第3版).北京:清华大学出版社, 2006.1  [2](德)Nicolai M. Josuttis著;候捷/孟岩译. C++标准程序库.武汉：华中科技大学出版社, 2002.9  [3]张俊.现代C++面向对象程序设计.北京：科学出版社，2022.3  [4]徐孝凯.C++语言程序设计.北京：北京师范大学出版社，2022.5  [5]张远龙.C++服务器开发精髓.北京：电子工业出版社，2021.5  [6]石鲁生,梁凤兰.C++程序设计案例驱动实用教程.南京：南京大学出版社，2020.7  [7]陈卫卫,王庆瑞.C/C++程序设计.北京：机械工业出版社，2019.1  [8]严悍,陆建峰,衷宜.新标准C++程序设计.南京：南京东南大学出版社，2018.8  [9]翁惠玉,俞勇.C++程序设计.北京：人民邮电出版社，2016.8  [10]宋春花,吕进来,王丽娟等.C++面向对象程序设计.北京：人民邮电出版社，2014.2  [11]赵烨.轻松学算法.北京：电子工业出版社，2016.7  [12]林少伟,唐林垚.驯服算法.上海：上海人民出版社，2020.7  [13]梁冰,冯林,刘胜蓝.算法基础.北京：电子工业出版社，2019.1  [14]许少华.算法设计与分析.哈尔滨:哈尔滨工业大学出社，2011.8  [15]麻新旗,王春红,张世民等.计算思维与算法设计.北京：人民邮电出版社，2015.9 |

# 目 录

[摘 要 III](#_Toc155360420)

[Abstract III](#_Toc155360421)

[第一章 绪论（概述） 1](#_Toc155360422)

[1.1 设计背景及目的 1](#_Toc155360423)

[1.2设计意义 1](#_Toc155360424)

[1.3应解决的主要问题及应达到的技术要求 1](#_Toc155360425)

[1.4基本理论依据和主要工作内容 1](#_Toc155360426)

[第二章 系统设计 3](#_Toc155360427)

[2.1 数据结构选择 3](#_Toc155360428)

[2.2类设计 3](#_Toc155360429)

[2.3运算符重载 3](#_Toc155360430)

[2.4微分和积分运算 3](#_Toc155360431)

[2.5求根算法与在某点的函数值 3](#_Toc155360432)

[2.6菜单选择系统及用户界面 3](#_Toc155360433)

[2.7测试和调试 3](#_Toc155360434)

[2.8注释 3](#_Toc155360435)

[2.9显示时间 3](#_Toc155360436)

[第三章 算法与过程实现 4](#_Toc155360437)

[3.1多项式类定义 4](#_Toc155360438)

[3.2 运算符重载 4](#_Toc155360439)

[3.3微分和积分运算 4](#_Toc155360440)

[3.4 求根算法 4](#_Toc155360441)

[3.5 菜单选择系统 4](#_Toc155360442)

[3.6算法实现 4](#_Toc155360443)

[3.7注释 11](#_Toc155360444)

[3.8显示时间 11](#_Toc155360445)

[第四章 功能测试与结果分析 12](#_Toc155360446)

[4.1查看当前时间 12](#_Toc155360447)

[4.2计算多项式在某点附近的根 12](#_Toc155360448)

[4.3多项式的积分或微分运算 12](#_Toc155360449)

[4.4计算多项式在某点的函数值 13](#_Toc155360450)

[4.5多项式算术运算 14](#_Toc155360451)

[总 结 17](#_Toc155360452)

[致 谢 18](#_Toc155360453)

[参考文献 19](#_Toc155360454)

摘 要

一元多项式计算器综合设计报告主要介绍了一个基于C++面向对象编程实现的一元多项式计算器的设计、实现和测试过程。项目旨在满足多项式的创建、显示、相加、相减、相乘、相除、取余、微分、积分、求某点附近的根和给定一个自变量值，计算多项式在该点的函数值，等操作需求。论文分为以下几个部分：

1.引言：简要介绍了一元多项式计算器的背景和意义，以及面向对象编程在解决实际问题中的优势。

2.需求分析：详细阐述了一元多项式计算器的功能需求，为后续设计和实现奠定基础。

3.设计：提出了一个面向对象的类结构，CPolynomial类。重点讨论了类的封装、继承和多态在设计中的应用。

4.实现：使用C++语言实现了设计的类结构，利用C++的特性（如构造函数、析构函数、运算符重载等）简化代码并提高性能。遵循良好的编程实践，确保代码可读性和可维护性。

5.测试：对一元多项式计算器进行了详细的测试，包括单元测试、集成测试和系统测试。测试结果验证了项目的功能性和可靠性。

6.总结：回顾了项目的设计、实现和测试过程，总结了项目的优点和不足。讨论了如何进一步优化项目，以提高软件质量。

关键词：一元多项式计算器；封装；继承；多态；测试

Abstract

The comprehensive design report of the univariate polynomial calculator mainly introduces the design, implementation and testing process of a univariate polynomial calculator based on C++ object-oriented programming. The project aims to meet the needs of creating polynomials, displaying, adding, subtracting, multiplying, dividing, remaining, differentiating, integrating, finding the root near a point, given the value of an independent variable, calculating the function value of a polynomial at that point, and so on. The dissertation is divided into the following sections:

1. Introduction: A brief introduction to the background and significance of univariate polynomial calculators, as well as the advantages of object-oriented programming in solving practical problems.

2. Requirements analysis: elaborate the functional requirements of the univariate polynomial calculator to lay the foundation for subsequent design and implementation.

3. Design: An object-oriented class structure, CPolynomial class , is proposed. The focus is on the encapsulation, inheritance, and polymorphism of classes in design.

4. Implementation: The class structure of the design is implemented in the C++ language, and the features of C++ (such as constructors, destructors, operator overloads, etc.) are used to simplify the code and improve performance. Follow good

programming practices to ensure code readability and maintainability.

5. Testing: The unary polynomial calculator was tested in detail, including unit tests, integration tests, and system tests. The test results verified the functionality and reliability of the project.

6. Summary: The design, implementation and testing process of the project is reviewed, and the strengths and weaknesses of the project are summarized. Discussed how to further optimize the project to improve the quality of the software.

**Keywords**:Unary polynomial calculators;encapsulation;inheritan;polymorphism;testing

第一章 绪论（概述）

1.1 设计背景及目的

1.1.1背景

在数学和工程领域，一元多项式在解决各种问题时具有广泛的应用，如信号处理、控制系统、数值分析等。因此，设计一个高效、易用的一元多项式计算器对于简化计算过程、提高工作效率具有重要意义。

1.1.2目的

设计一个一元多项式计算器，能够实现多项式的构造、基本运算（加、减、乘、除）、微分、积分、求根等操作，并提供用户友好的界面，以便用户能够方便地进行多项式相关的计算。

1.2设计意义

1.提高教育和研究中多项式处理的效率。

2.为工程应用提供快速求解多项式问题的工具。

3.通过实践提高对面向对象编程和数据结构的理解。

1.3应解决的主要问题及应达到的技术要求

1.3.1应解决的主要问题

1.如何有效地存储和操作多项式，以支持快速的计算和修改。

2.如何实现多项式的高级运算，如微分、积分和求根。

3.如何设计用户界面，使得非专业用户也能够轻松地使用计算器。

1.3.2应达到的技术要求

1.使用动态数组（如std::vector）或链表来存储多项式的系数。

2.实现多项式的基本运算，包括加法、减法、乘法、除法（带余数）和取余。

3.实现多项式的微分和积分运算，支持指定阶数的微分和给定区间的定积分。

4.实现牛顿迭代法求根，用于求解多项式的根。

5.实现多项式的文件输入/输出，以便保存和加载多项式对象。

6.提供菜单选择系统，允许用户通过键盘输入选择不同的操作。

1.4基本理论依据和主要工作内容

1.4.1基本理论依据

多项式的基本性质、牛顿迭代法、数值微分和积分方法。

1.4.2主要工作内容

1.设计一元多项式类，包括构造函数、复制构造函数、赋值运算符、析构函数等。

2.实现多项式的加法、减法、乘法、除法（带余数）和取余运算。

3.实现微分和积分运算，包括微分的阶数和积分的区间。

4.实现牛顿迭代法求多项式的根。

5.实现文件输入/输出功能，以便保存和加载多项式对象。

6.设计用户界面，提供清晰的菜单选择，以便用户进行操作。

第二章 系统设计

2.1 数据结构选择

使用STL的std::vector或std::list来存储多项式的系数和指数。std::vector由于其动态数组的特性，更适合于这种应用，因为它可以方便地进行插入和删除操作，同时保持元素的连续性。

2.2类设计

创建一个名为Polynomial的类，它将包含一个std::vector来存储系数和指数。

类中应该包含构造函数、析构函数、复制构造函数、赋值运算符等基本成员函数。

提供成员函数来设置和获取多项式的系数和指数。

2.3运算符重载

1.实现+、-、\*、/运算符重载，用于多项式的加、减、乘、除运算。

2.实现()运算符重载，用于计算多项式在给定自变量值的函数值。

3.实现%运算符重载，用于带余数的除法。

4.实现<<运算符重载，用于输出。

2.4微分和积分运算

1.实现微分运算，根据给定的阶数n，计算多项式的微分。

2.利用辛普森规则实现积分运算，根据给定的区间，计算多项式的定积分。

2.5求根算法与在某点的函数值

1.实现牛顿迭代法，用于求解多项式的根。

2.提供成员函数来执行求根操作，并返回根的近似值。

3.计算多项式在某点附近的函数值。

2.6菜单选择系统及用户界面

1.设计一个简单的文本菜单，允许用户通过键盘输入选择不同的操作。

2.实现菜单循环，根据用户的选择执行相应的操作。

3.提供清晰的文本界面，展示当前的多项式、操作结果和菜单选项。

实现错误处理和用户反馈机制。

2.7测试和调试

1.编写测试用例来验证多项式的各种运算是否正确。

2.在开发过程中进行调试，确保没有内存泄漏和其他潜在的错误。

2.8注释

为代码编写详细的注释，解释每个函数和类的作用。

2.9显示时间

显示当前的年月日及其时分秒，为用户提供准确的时间。

第三章 算法与过程实现

3.1多项式类定义

1.创建一个名为Polynomial的类，它包含一个std::vector来存储系数（从低次项到高次项）。

2.定义构造函数、析构函数、复制构造函数、赋值运算符等。

3.2 运算符重载

1.实现加法、减法、乘法运算符重载，用于多项式的加、减、乘运算。

2.实现除法运算符重载，包括带余数的除法。

3.实现圆括号运算符重载，用于计算多项式在给定自变量值的函数值。

3.3微分和积分运算

1.实现微分运算，根据给定的阶数n，计算多项式的微分。

2.使用辛普森原则实现定积分运算，根据给定的区间，计算多项式的定积分。

3.4 求根算法

1.实现牛顿迭代法，用于求解多项式的根。

2.提供成员函数来执行求根操作，并返回根的近似值。

3.5 菜单选择系统

1.设计一个简单的文本菜单，允许用户通过键盘输入选择不同的操作。

2.实现菜单循环，根据用户的选择执行相应的操作。

3.6算法实现

3.6.1加法和减法

std::vector<std::pair<float, int>> add(std::vector<std::pair<float, int>>v1, std::vector<std::pair<float, int>>v2)

{std::vector<std::pair<float, int>> mergedVec = {};

std::vector<std::pair<float, int>> ::iterator it;

std::map<float, float> sumMap = {};

for (const auto& p : v1) {sumMap[p.second] = p.first;}

for (const auto& p : v2) {sumMap[p.second] += p.first;}

for (const auto& kv : sumMap) {

if (kv.second == 0 && kv.first != 0) {;}

else{mergedVec.push\_back(std::make\_pair(kv.second, kv.first));}}

return (mergedVec);}

std::vector<std::pair<float, int>> sub(std::vector<std::pair<float, int>>v1, std::vector<std::pair<float, int>>v2)

{std::vector<std::pair<float, int>> mergedVec;

std::vector<std::pair<float, int>> ::iterator it;

std::map<float, float> sumMap;

for (const auto& p : v1) {sumMap[p.second] = p.first;}

for (const auto& p : v2) {sumMap[p.second] -= p.first;}

for (const auto& kv : sumMap)

{if (kv.second == 0 && kv.first != 0) {;}

else {mergedVec.push\_back(std::make\_pair(kv.second, kv.first));}}

return (mergedVec);}

如上述代码所示，这两段代码分别定义了两个函数add和sub，它们用于实现两个多项式的加法和减法运算。这些多项式以std::vector<std::pair<float, int>>的形式表示，其中pair<float, int>的first成员是系数，second成员是指数。

加法函数add的分析：

1.初始化一个空的合并向量mergedVec和一个空的std::map<float, float>sumMap，用于存储和计算每个指数对应的系数和。

2.遍历第一个多项式v1，将每个项的系数添加到sumMap中，键是指数，值是系数。

3.遍历第二个多项式v2，对于每个项，将其系数从sumMap中对应的键值中减去（因为减法实际上是加法的逆操作）。

4.遍历sumMap，如果系数为0且指数不为0，说明这个项在减法中被完全抵消，不需要添加到结果向量中。

5.如果系数不为0，将这个项作为一对std::pair<float, int>添加到mergedVec中。

6.返回合并后的多项式mergedVec。

减法函数sub的分析：

1.初始化一个空的向量mergedVec和一个空的std::map<float, float>sumMap。

2.遍历第一个多项式v1，将每个项的系数添加到sumMap中。

3.遍历第二个多项式v2，对于每个项，将其系数从sumMap中对应的键值中减去。

4.遍历sumMap，如果系数为0且指数不为0，说明这个项在减法中被完全抵消，不需要添加到结果向量中。

5.如果系数不为0，将这个项作为一对std::pair<float, int>添加到mergedVec中。

6.返回合并后的多项式mergedVec。

3.6.2乘法

std::vector<std::pair<float, int>> mul(std::vector<std::pair<float, int>>v1, std::vector<std::pair<float, int>>v2)

{std::vector<std::pair<float, int>> ret{};

std::vector<std::pair<float, int>>::iterator it1, it2;

for (it1 = v1.begin(); it1 != v1.end(); ++it1)

{std::pair<float, int> p = {};

for (it2 = v2.begin(); it2 != v2.end(); ++it2)

{float xsf = it1->first;

int xmc = it1->second;

float new\_xsf = xsf \* it2->first;

int new\_xmc = xmc + it2->second;

p.first = new\_xsf;

p.second = new\_xmc;

ret.push\_back(p);

}}

if (ret.size() <= 0) return ret;std::sort(ret.begin(), ret.end(), Merge());

std::vector<std::pair<float, int>> result;

for (auto it = ret.begin(); it != ret.end(); ++it) {

if (result.empty() || result.back().second != it->second) {result.push\_back(\*it);}

else {result.back().first += it->first;}}

return (result);}

如上述代码所示，这一段代码定义了一个名字为mul的乘法函数，它可以接受两个std::vector<std::pair<float, int>>类型的参数v1和v2，分别代表两个多项式。函数的目的是对这两个多项式进行乘法运算，并返回乘积多项式。

函数的工作原理如下：

1.初始化结果向量：创建一个空的std::vector<std::pair<float, int>>类型的向量ret用于存储乘法运算的结果。

2.多项式乘法：

（1）使用两个迭代器it1和it2分别遍历两个输入多项式v1和v2。

（2）对于v1中的每个项，与v2中的每个项相乘，得到新的系数new\_xsf和新的指数new\_xmc。

（3）将乘法结果作为一对std::pair<float, int>添加到ret中。

3.合并相同指数的项：

（1）如果ret不为空，对其进行排序，排序依据是多项式的指数。

（2）创建一个空的std::vector<std::pair<float, int>>类型的向量result用于存储最终的乘积多项式。

（3）遍历排序后的ret，如果result为空或者当前项的指数与result中最后一个项的指数不同，则将当前项添加到result中。

（4）如果当前项的指数与result中最后一个项的指数相同，则将当前项的系数加到result中最后一个项的系数上。

4. 返回结果：函数返回合并后的乘积多项式result。

3.6.3除法

std::vector<std::pair<float, int>> div(std::vector<std::pair<float, int>>v1, std::vector<std::pair<float, int>>v2)

{std::sort(v1.begin(), v1.end(), MergeDec());

std::sort(v2.begin(), v2.end(), MergeDec());

std::vector<std::pair<float, int>> yushu ,ans ,di ;

float shang = 0;

int mic = 0;

yushu = v1;

std::pair<float, int >pp;

int iMaxMic = v1.begin()->second;

std::vector<std::pair<float, int>> ve;

LLOO:

std::vector<std::pair<float, int>>::iterator it1, it2, it22;

for (it1 = v1.begin(); it1 != v1.end(); ++it1) {

for (it2 = v2.begin(); it2 != v2.end(); ++it2) {

std::pair<float, int> p;

if (it1->second < it2->second) {

goto Ext;}

p.first = it1->first / it2->first;

p.second = it1->second - it2->second;

ans.push\_back(p);

ve.clear();

for (it22 = v2.begin(); it22 != v2.end(); ++it22) {

std::pair<float, int> pp;

pp.first = it22->first \* p.first;

pp.second = it22->second + p.second;

ve.push\_back(pp);}

yushu.clear();

yushu = sub(v1, ve);

v1.clear();

v1 = yushu;

std::sort(v1.begin(), v1.end(), MergeDec());

goto LLOO;}}

Ext:return yushu;}

如上述代码所示，这一段代码定义了一个名为div的除法函数，它可以接受两个std::vector<std::pair<float, int>>类型的参数v1和v2，分别代表两个多项式。函数的目的是对这两个多项式进行除法运算，返回余数多项式。

函数的工作原理如下：

1.排序多项式：首先对两个多项式v1和v2进行排序，排序依据是多项式的指数，从高到低。

2.初始化变量：

（1）yushu：用于存储余数多项式。

（2）ans：用于存储商多项式。

（3）di：未使用，用于存储除数多项式。

（4）shang：用于存储当前商的系数。

（5）mic：用于存储当前商的指数。

（6）pp：用于存储临时的多项式项。

（7）iMaxMic：用于存储v1中最高次项的指数。

3. 除法循环：

（1）使用两个迭代器it1和it2分别遍历v1和v2。

（2）如果v1中的当前项的指数小于v2中的当前项的指数，跳转到Ext标签，表示无法继续除。

（3）计算商的系数p.first和指数p.second，并将这个商项添加到ans中。

（4）计算v2中每一项乘以商项p的结果，并将这些结果添加到ve中。

（5）从v1中减去ve，得到新的余数多项式yushu。

（6）清空v1，并将yushu赋值给它。

（7）对新的v1进行排序，然后继续循环，直到v1为空或者无法继续除。

4.返回余数：函数返回余数多项式yushu。

3.6.4微分

std::vector < std::pair<float, int>> getWF(std::vector < std::pair<float, int>> &v)

{LO:std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*请输入微分运算的阶数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::string input;

while (true) {std::cin >> input;

if (input.empty()) {std::cin.clear();continue;}

else if (std::all\_of(input.begin(), input.end(), ::isdigit)) {break;}

else {std::cin.clear();continue;}}

if (!isFloat(input)) {goto LO;}

float yy = atof(input.c\_str());

std::vector < std::pair<float, int>> ::iterator it;

std::vector <std::pair<float, int>> ret;

std::sort(v.begin(), v.end(), MergeDec());

for (it = v.begin(); it != v.end(); ++it) {

if (it->second == 0) {v.erase(it);break;}}

for (it = v.begin(); it != v.end(); ++it) {

std::pair<float, int>p;

p.first = it->first \* it->second;

p.second = it->second - 1;

ret.push\_back(p);}

return ret;}

如上述代码所示，这一段代码定义了一个名为getWF的微分函数，它可以接受一个std::vector<std::pair<float, int>>类型的引用v作为参数，该向量包含了多项式的系数和指数对。函数的目的是对多项式进行微分运算，并返回微分后的多项式。

函数的工作原理如下：

1.用户输入：首先，函数提示用户输入微分运算的阶数。用户通过标准输入（std::cin）输入一个字符串input。

2.输入验证：

（1）如果输入为空，清除输入流并继续循环。

（2）如果输入不是纯数字，清除输入流并继续循环。

（3）如果输入是纯数字，转换为浮点数yy。

3.常数项处理：在微分多项式时，常数项的导数为0，所以函数会删除向量v中的常数项（即指数为0的项）。

4.微分运算：

（1）对于向量v中的每个非常数项，计算其导数。导数的系数是原系数乘以原指数，指数减1。

（2）将计算出的导数项添加到新的向量ret中。

5.返回结果：函数返回微分后的多项式ret。

3.6.5积分

float simpson\_rule(float a, float b, int n, std::vector<float> xs, int d) {float h = (b - a) / n;

float x = a;

float fintegral = 0.0;

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

double temp = (i == 0 || i == n - 1) ? evepoly(x, xs, d) : (i % 2 == 0) ? 2 \* evepoly(x, xs, d) : 4 \* evepoly(x, xs, d);fintegral += temp;x += h;}

fintegral \*= h / (3 \* n);return fintegral;}

如上述代码所示，这段代码定义了一个名为simpson\_rule的函数，它使用辛普森规则（Simpson's rule）来近似计算一元多项式在区间[a, b]上的定积分。辛普森规则是一种数值积分方法，它通过在积分区间上选择奇数个点来近似积分值。

函数的参数和工作原理如下：

1.a：积分区间的下限。

2.b：积分区间的上限。

3.n：辛普森规则中使用的子区间数量。n必须是奇数，因为辛普森规则在每个子区间上使用3个点。

4.xs：一个std::vector<float>，包含了多项式的系数，用于计算多项式在特定点的值。

5.d：多项式的最高次项的指数，即多项式的次数。

函数的执行步骤：

1. 计算子区间的长度h，即(b - a) / n。

2. 初始化积分近似值fintegral为0.0。

3. 使用一个for循环，从0到n（不包括n），步长为1，遍历每个子区间。

4. 在循环中，首先根据当前的i值和x计算多项式在该点的值。这里使用了evepoly函数，它之前已经定义好了，用于计算多项式在给定自变量x处的函数值。

5. 对于每个子区间，根据i的奇偶性，选择不同的系数加到fintegral上。这是辛普森规则的一部分，其中奇数子区间的系数是4，偶数子区间的系数是2，而第一个和最后一个子区间的系数是1。

6. 更新x的值，使其移动到下一个子区间的起始点。

7. 循环结束后，将fintegral乘以h / (3 \* n)，这是辛普森规则的最终修正系数，用于将近似值转换为积分的准确值。

8. 返回fintegral，即多项式在区间[a, b]上的定积分近似值。

这段代码假设xs向量至少有d + 1个元素，并且d是一个非负整数。此外，evepoly函数应该正确地计算多项式在给定自变量x处的值。

3.6.6牛顿迭代法求根

float CPolynomial::getPolynomialValueArea(float x)//在某点附近的根

{float yy = x;

std::vector <std::pair<float, int>> ::iterator it;

bool bFind = false;

std::sort(m\_vecData.begin(), m\_vecData.end(), Merge());

for (it = m\_vecData.begin(); it != m\_vecData.end(); ++it) {

if (it->second == 0) {it->first -= yy;bFind = true;break;}}

if (bFind == false) {

std::pair<float, int>p;

p.first = 0 - yy;

p.second = 0;

m\_vecData.insert(m\_vecData.begin(), p);}

std::vector <std::pair<float, int>> vdx1;

for (it = m\_vecData.begin(); it != m\_vecData.end(); ++it) {std::pair<float, int>p;

if (it->second == 0) {continue;}

p.first = it->first \* it->second;

p.second = it->second - 1;

vdx1.push\_back(p);}

float x0, x1 = 1.5, f, fdx;

do {x0 = x1;

f = getValue(m\_vecData, x0);

fdx = getValue(vdx1, x0);

x1 = x0 - f / fdx;} while (fabsf(x1 - x0) >= 1e-5);

return x1;}

如上述代码所示，这段代码是CPolynomial类的一个成员函数getPolynomialValueArea，它的目的在于找到一个多项式在给定自变量值x附近的根。这个函数使用了牛顿迭代法（Newton's method）来求解多项式的根。

函数的工作原理如下：

1. 初始化：将自变量x赋值给yy。

2. 寻找多项式中的常数项：遍历多项式m\_vecData，寻找系数为0的项。如果找到，将该项的系数减去yy，并标记bFind为true。

3.添加常数项：如果多项式中没有常数项，创建一个新的std::pair<float, int>，其系数为yy，指数为0，并将其插入到m\_vecData的开始位置。

4.计算导数：遍历m\_vecData，对于每个非零指数的项，计算其导数（系数乘以指数，指数减1），并将结果存储在vdx1向量中。

5.牛顿迭代：

（1）使用牛顿迭代法来求解根。这个循环会反复执行，直到新的x1与旧的x0之间的差小于给定的容差（1e-5）。

（2）在每次迭代中，计算当前点x0处的函数值f和导数值fdx。

（3）使用牛顿迭代公式x1 = x0 - f / fdx来更新x1。

（4）如果新的x1与旧的x0之间的差小于容差，迭代结束。

6.返回结果：返回迭代得到的根x1。

3.6.7某点的函数值

float CPolynomial::getPolynomialValueAtPoint(int a)

{int x = a;

std::sort(this->m\_vecData.begin(), m\_vecData.end(), MergeDec());

float sum = 0;

std::vector <std::pair<float, int>>::iterator it;

for (it = m\_vecData.begin(); it != m\_vecData.end(); ++it) {

if (it->second == 0) {sum += it->first;}

else {float fmc = 1;

for (int index = 0; index < it->second; ++index) {fmc \*= x;}

fmc = it->first \* fmc;

sum += fmc;}}

return sum;}

如上述代码所示，这段代码是CPolynomial类的一个成员函数，它的目的在于计算多项式在给定自变量值a处的函数值。

函数的工作原理如下：

1.初始化：将自变量a赋值给x。

2.排序多项式：使用std::sort对m\_vecData进行排序，排序依据是多项式的指数，从高到低。

3.计算函数值：

（1）初始化一个变量sum，用于累加多项式的各项系数。

（2）使用迭代器it遍历排序后的多项式m\_vecData。

（3）对于每个项，如果其指数为0（即常数项），直接将其系数加到sum中。

（4）如果指数不为0，计算x的指数次幂（fmc），然后将该值乘以该项的系数，并将结果加到sum中。

4.返回结果：返回累加后的sum，即多项式在点a处的函数值。

3.7注释

为代码编写详细的注释，解释每个函数和类的作用。

3.8显示时间

调用C.time显示当前的年月日及其时分秒，为用户提供实时的时间查看。

第四章 功能测试与结果分析

4.1查看当前时间

如图4-1所示，当在主菜单输入“5”时，输出当前的年月日及其时分秒

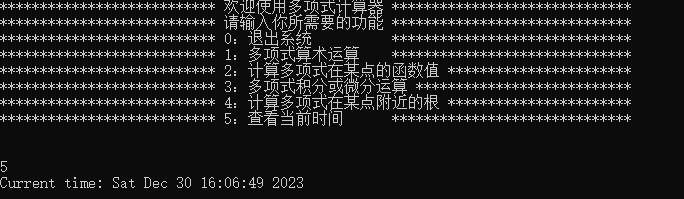


图4-1 查看当前时间功能测试图

4.2计算多项式在某点附近的根

如图4-2所示，当在主菜单输入“4”时，输出“请输入多项式”；输入多项式：“6+5\*x^2+4\*x^3+7\*x^4”，输入enter，输出“请输入y的值”；输入y的值为“0”时，输出“该多项式的根为-0.693549”。



图4-2 计算多项式在某点附近的根功能测试图

4.3多项式的积分或微分运算

4.3.1多项式的微分运算

如图4-3所示，当在主菜单输入“3”时，输出“请输入多项式”；输入多项式：“3+x^2+3\*x^4+7\*x^8-8\*x^8+3\*x^9”，输入“0”，输出“请输入微分运算的阶数”；输入阶数为“2”时，输出“2.00\*x+12.00\*x^3+56.00\*x^7-64.00\*x^7+27.00\*x^8”。

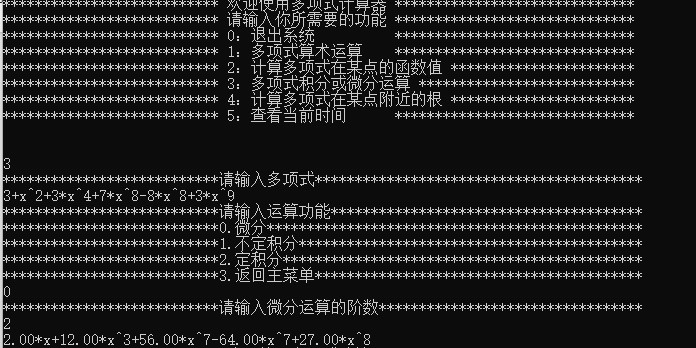


图4-3 多项式的微分运算功能测试图

4.3.2多项式的不定积分运算

如图4-4所示，当在主菜单输入“3”时，输出“请输入多项式”；输入多项式“4+x^2+3\*x^3+5\*x^4-2\*x^5+8\*x^6”并输入“1”，输出不定积分的运算结果“4.00\*x+0.33\*x^3+0.75\*x^4+x^5-0.33\*x^6+1.14\*x^7+C”。



图4-4 多项式的不定积分运算功能测试图

4.3.3多项式的定积分运算

如图4-5所示，在主菜单输入“3”时，输出“请输入多项式”；输入多项式“6+3\*x^2+4\*x^3-2\*x^5+7\*x^7”并输入“2”；输出“请输入定积分的区间”“请输入形如1，2的数据来表示区间”；继续输入“2，4”，输出多项式定积分的运算结果“56561.34”

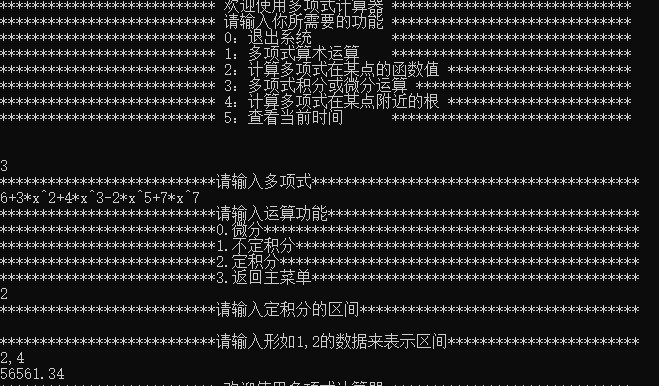


图4-5 多项式的定积分运算功能测试图

4.4计算多项式在某点的函数值

如图4-6所示，在主菜单输入“2”时，输出“请输入多项式”；输入多项式“1+2\*x-4\*x^2+5\*x^3”,输出“请输入x的值”，继续输入“2”，输出“该点y的值”为“29”。

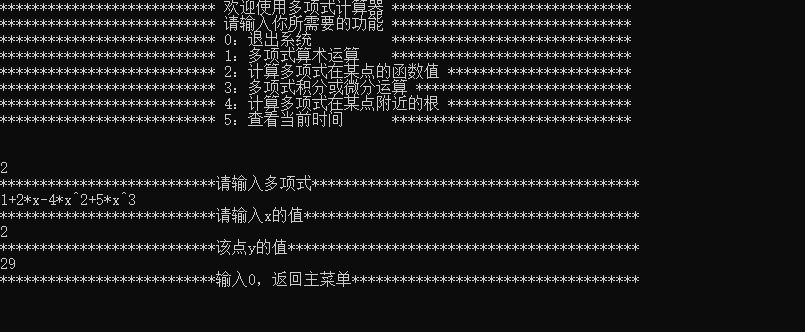


图4-6 计算多项式在某点的函数值功能测试图

4.5多项式算术运算

4.5.1多项式加法运算

如图4-7所示，在主菜单输入“1”时，分别输出“请输入第一个多项式”“请输入第二个多项式”；分别输入第一个多项式“5+2\*x^2+3\*x^4-5\*x^5”，第二个多项式“8+1\*x^3+9\*x^4-7\*x^5+3\*x^7”；输出“请输入你所需要的功能”，当输入“0”时，输出两个多项式的加法结果“13.00+2.00\*x^2+x^3+12.00\*x^4-12.00\*x^5+3.00\*x^7”。



图4-7 多项式加法运算功能测试图

4.5.2多项式减法运算

如图4-8所示，在主菜单输入“1”时，分别输出“请输入第一个多项式”“请输入第二个多项式”；分别输入第一个多项式“6+3\*x^2-6\*x^4+14\*x^8”，第二个多项式“7+5\*x^3-6\*x^7+9\*x^9”；输出“请输入你所需要的功能”，当输入“1”时，输出多项式减法结果“-1.00+3.00\*x^2-5.00\*x^3-6.00\*x^4+6.00\*x^7+14.00\*x^8-9.00\*x^9”。



图4-8 多项式减法运算功能测试图

4.5.3多项式乘法运算

如图4-9所示，在主菜单输入“1”时，分别输出“请输入第一个多项式”“请输入第二个多项式”；分别输入第一个多项式“6+2\*x^1+3\*x^2-71\*x^4+32\*x^7”，第二个多项式“8+6\*x^4+5\*x^5-6\*x^8”；输出“请输入你所需要的功能”，当输入“2”时输出 “48.00+16.00\*x+24.00x^2+532.00\*x^4+42.00\*x^5+28.00\*x^6+271.00\*x^7-462.00\*x^8-367.00\*x^9-18.00\*x^10+192.00\*x^11+586.00\*x^12-192.00\*x^15”。



图4-9 多项式乘法运算功能测试图

4.5.4多项式除法运算

如图4-10所示，在主菜单输入“1”时，分别输出“请输入第一个多项式”“请输入第二个多项式”；分别输入第一个多项式“18+4\*x^7-6\*x^9+7\*x^10”，第二个多项式“9+4\*x^2+8\*x^4-4\*x^9”；输出“请输入你所需要的功能”，输入“3”时，输出多项式除法结果：“4.50+15.75\*x-6.00x^2+7.00\*x^3-12.00\*x^4+14.00\*x^5+4.00\*x^7”。



图4-10 多项式除法运算功能测试图

4.5.5多项式取余运算

如图4-11所示，在主菜单输入“1”时，分别输出“请输入第一个多项式”“请输入第二个多项式”；分别输入第一个多项式“6+4\*x^2-7\*x^4+8\*x^6+3\*x^8”，第二个多项式“9+x^3+6\*x^4-7\*x^5+9\*x^6”；输出“请输入你所需要的功能”，当输入“4”时，输出两个多项式除法结果：“0.87+0.26\*x+0.33x^2”。



图4-11 多项式取余运算功能测试图

总 结

在本篇C++面向对象程序设计一元多项式计算器综合设计报告中，我们详细介绍了一个基于C++面向对象编程实现的一元多项式计算器的设计、实现和测试过程。通过这个项目，我们展示了如何运用面向对象编程原则解决实际问题，提高代码的可读性、可维护性和可扩展性。以下是对本报告的总结。

首先，我们对项目进行了需求分析，明确了一元多项式计算器的主要功能，包括多项式的创建、显示、相加、相减、相乘和求导等。基于这些需求，我们设计了一个面向对象的类结构，包括表示表示一元多项式的CPolynomial类。

在设计阶段，我们重点考虑了类的封装、继承和多态。CPolynomial类重载圆括号运算符，重载运算符实现多项式的算术运算，同时封装了多项式的多项式项列表，实现了多项式的各种操作，如添加，加减乘除，取余，模的计算，求定积分，不定积分，微分等运算，利用牛顿迭代法计算在某点附近的根，以及计算结果。为了实现多态，我们Polynomial类定义了相同的接口，以便在不同情况下使用不同的实现。

接下来，我们进入了编码阶段。我们使用C++语言实现了Polynomial类，利用构造函数、析构函数、运算符重载等特性简化了代码，提高了性能。在实现过程中，我们遵循了良好的编程实践，编写了清晰的注释和遵循一致的编码风格。

完成编码后，我对项目进行了详细的测试，包括单元测试、集成测试和系统测试。我们使用了合适的测试方法和工具，确保项目的各个部分都能正常工作，并满足项目的需求。在测试过程中，我发现了一些潜在的问题，并及时进行了修复和优化。

在项目总结阶段，我分析了项目的优点和不足，以及在项目过程中遇到的问题和挑战。我们讨论了如何运用面向对象编程原则和C++特性来改进项目的设计和实现，以提高项目的质量。

该项目仍然存在些许不足，比如由于时间问题，我未能编写出文件的存储与读取功能，以及计算器计算精度不够，只能精确到小数点后两位，今后仍需继续努力，力求做出更加完整的项目。

总之，本篇C++面向对象程序设计一元多项式计算器综合设计报告详细展示了一个实际项目的设计和实现过程，强调了面向对象编程原则在解决实际问题中的应用。通过这个报告，能够更好地理解和掌握C++面向对象编程的方法和技巧，并将这些知识应用到实际项目中，提高软件开发的效率和质量。

致 谢

在这篇C++综合设计报告即将完成之际，我想借此机会向那些在报告写作过程中给予我支持和帮助的人们表示衷心的感谢。

首先，我要感谢我的导师，他不仅为我提供了宝贵的学术指导，还在我遇到困难时给予了鼓励和支持。他的专业知识、严谨的治学态度以及对学术研究的热情深深地影响了我。在论文的选题、构思和撰写过程中，导师的耐心指导和建议使我受益匪浅。

其次，我要感谢同学们。在这段时间里，我们共同探讨问题、交流想法，互相帮助和鼓励。他们的友谊和支持让我在学术研究的道路上不再孤单。同时，实验室的学术氛围也激发了我对C++编程的热情，使我在报告写作过程中不断进步。

此外，我要感谢学校和相关院系提供的优质教育资源。在这里，我有机会接触到前沿的学术研究，参加各种学术活动，拓宽了我的视野。同时，学校提供的实验设施和图书馆资源也为我的报告写作提供了极大的便利。

在报告写作过程中，我还得到了软件工程学长的帮助。他们在C++的学习领域具有丰富的经验和独到的见解，为我提供了许多宝贵的意见和建议。在此，我要向他们表示诚挚的谢意。

此外，我还要感谢我的家人。他们在我求学的道路上一直给予我关爱和支持，鼓励我追求卓越。在我遇到困难和挫折时，他们始终相信我，给我信心和力量。没有他们的陪伴和支持，我无法完成这篇报告。

最后，我要感谢我的朋友们。他们在我生活中的点点滴滴都给予了我关心和帮助。在我疲惫时，他们给我鼓励；在我迷茫时，他们给我指引。他们的陪伴让我在学术研究的道路上充满信心和动力。

在此，我再次向所有给予我帮助和支持的人们表示衷心的感谢。这篇报告的完成离不开你们的关爱和鼓励。我会继续努力，以更好的学术成果回报你们的期望。

参考文献

[1](美)Horton, I.著; 李予敏译.C++入门经典(第3版).北京:清华大学出版社, 2006.1

[2](德)Nicolai M. Josuttis著;候捷/孟岩译. C++标准程序库.武汉：华中科技大学出版社, 2002.9

[3]张俊.现代C++面向对象程序设计.北京：科学出版社，2022.3

[4]徐孝凯.C++语言程序设计.北京：北京师范大学出版社，2022.5

[5]张远龙.C++服务器开发精髓.北京：电子工业出版社，2021.5

[6]石鲁生,梁凤兰.C++程序设计案例驱动实用教程.南京：南京大学出版社，2020.7

[7]陈卫卫,王庆瑞.C/C++程序设计.北京：机械工业出版社，2019.1

[8]严悍,陆建峰,衷宜.新标准C++程序设计.南京：南京东南大学出版社，2018.8

[9]翁惠玉,俞勇.C++程序设计.北京：人民邮电出版社，2016.8

[10]宋春花,吕进来,王丽娟等.C++面向对象程序设计.北京：人民邮电出版社，2014.2

[11]赵烨.轻松学算法.北京：电子工业出版社，2016.7

[12]林少伟,唐林垚.驯服算法.上海：上海人民出版社，2020.7

[13]梁冰,冯林,刘胜蓝.算法基础.北京：电子工业出版社，2019.1

[14]许少华.算法设计与分析.哈尔滨:哈尔滨工业大学出社，2011.8

[15]麻新旗,王春红,张世民等.计算思维与算法设计.北京：人民邮电出版社，2015.9