Project 2 技术报告

17341015 计一陈鸿峥

1 实验目的

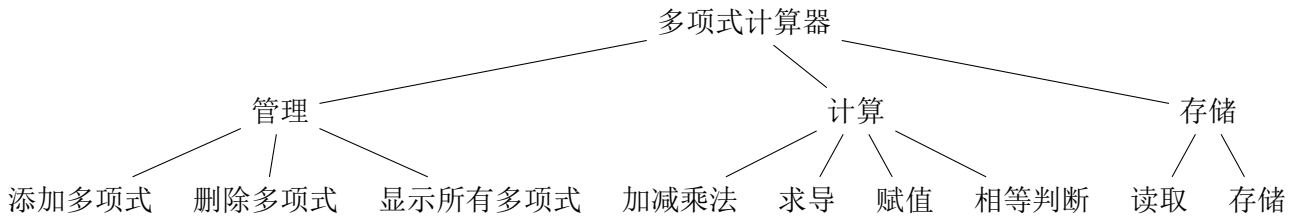
实现一个多项式计算器，可以进行简单的加减乘、求导等运算，并支持从文件读取和存储。

2 实验环境

采用 C++ 编写，在 Sublime Text 3 上进行开发，并用 gcc 6.3.0 编译，代码符合 C++11 标准。

3 实现思路

多项式计算器的功能如下：



本多项式计算器主要分为管理、计算、存储三大模块。

1. 管理模块
   包含添加多项式、删除多项式、显示所有多项式三个功能

2. 计算模块
   包含加法、减法、乘法、求导、判断多项式相等、赋值共6个功能

3. 存储模块
   包含读取数据和存储数据两个功能

4 设计细节

4.1 文件关系

本多项式计算器包含三个源程序文件：

1. polynomial_calculator.cpp 为主文件，包含上述三个模块及具体的交互界面

2. polynomial.hpp 包含一个多项式类，实现单个多项式的存储及运算操作

3. polynomials.hpp 包含一个多项式仓库类，实现所有多项式的存储读取及多项运算

4.2 头文件

1. <string>,<vector>,<map>为C++标准库的容器，用于存储数据
2. <iostream>,<fstream>用于命令行及文件的输入输出
3. <iomanip>用于控制输出格式
4. <regex>,<iterator>为正则表达式和迭代器，用于字符串的分割

4.3 类结构设计

下图为Polynomial类的实现，各个成员函数的含义均在命名中或是注释中表明，详情请见源码

```
10  class Polynomial
11  {
12  public:
13      // constructor
14      // default
15      Polynomial();
16      // construct from pairs like (1,2)
17      Polynomial(std::vector<std::pair<int,int>> poly);
18      // construct from coefficients
19      Polynomial(std::vector<int> poly);
20      // construct from other polynomials (=)
21      Polynomial(const Polynomial& other);
22      // destructor
23      ~Polynomial();
24      // clear all terms including 0
25      void clear();
26
27      // basic operations
28      // derivative
29      Polynomial D() const;
30      // overload
31      Polynomial operator+(const Polynomial& Poly) const;
32      Polynomial operator-(const Polynomial& Poly) const;
33      Polynomial operator*(const Polynomial& Poly) const;
34      Polynomial& operator=(const Polynomial& other);
35      Polynomial& operator+=(const Polynomial& Poly);
36      Polynomial& operator*=(const Polynomial& Poly);
37      bool operator==(const Polynomial& Poly) const;
38      // assignment
39      int operator()(int value) const;
40
41      // set and get data
42      void setCoefficient(int numTerm, int coeff);
43      int getTermCoeff(int numTerm) const; // get coefficient of specific term
44      int getHighestDegree() const;
45      std::vector<int> getCoefficients() const;
46
47      // output
48      std::string PolyToString() const;
49  private:
50      // a_0x_0^0+a_1x_1^1+\cdots+a_nx_n^n
51      // only record coefficients
52      // push term 0 into the vector while no corresponding terms were inputed
53      std::vector<int> terms;
54  };
```

值得一提的特性：

1. 类私有成员采用单个向量vector<int>的形式存储，如x^3+2x+1则用$[1,0,2,1]$表示，没有项的系数用0补齐
2. 提供了多种构造函数，支持多种构造方式
3. 对几个运算符进行了重载，使得编写代码更加简便
4. 由于存储某一多项式后terms向量将不会清空，故经过运算后存在向量最高几维为0的情况，故特意引入getHighestDegree()函数获取正确的最高次项
5. PolyToString()函数看似简单，但需要考虑的细节有很多

下图为Polynomials_类的实现，各个成员函数的含义均在命名中或是注释中表明，详情请见源码

```
class Polynomials_
{
public:
    //default constructor
    Polynomials_(){};
    //append polynomials
    bool append(const std::string name, Polynomial poly);
    //delete polynomials
    bool erase(const std::string name);

    //basic calculation
    Polynomial add(const std::vector<Polynomial> Polys) const;//enable multi-polynomials input
    Polynomial subtract(const Polynomial& PolyA, const Polynomial& PolyB) const;
    Polynomial multipy(const std::vector<Polynomial> Polys) const;//enable multi-polynomials input
    Polynomial derivative(const Polynomial& Poly) const;
    int assign(const Polynomial& Poly, int value) const;
    bool equalQ(const std::vector<Polynomial> Polys) const;

    //get data
    size_t getSize() const;
    bool findName(const std::string name) const;
    std::vector<std::string> getNames() const;//get all names of polynomials
    std::vector<Polynomial> getPolys(std::vector<std::string> names);//get specific polynomials from names
    std::map<std::string,Polynomial> getAllData() const;

    //count num of polynomials without names(temporary polynomials when calculation)
    static int cntVir;
private:
    //map names to polynomials
    std::map<std::string,Polynomial> data;
};
```

值得一提的特性：

1. 类私有成员采用从名字到多项式的映射map<string,Polynomial>存储
2. 加法和乘法支持多个多项式同时运算，故函数输入是一个Polynomial类型的vector
3. 减法支持一行内输入多个多项式，但只会对**前两个合法**的多项式进行操作
4. 求导、等价判断同样支持一行内输入多个多项式，但只会对最前面一个合法多项式进行操作
5. 引入了一个虚拟多项式(virtual polynomial)，用于存储计算时直接输入的无名多项式，名字就定为Polynomials_::cntVir的值，每次用完就删除；由于正常多项式的名字不允许出现数字(在validname_test中保证)，故名字为某一整数值不会与已有多项式冲突

4.4 主函数设计

主程序里的函数如下，具体实现请见代码

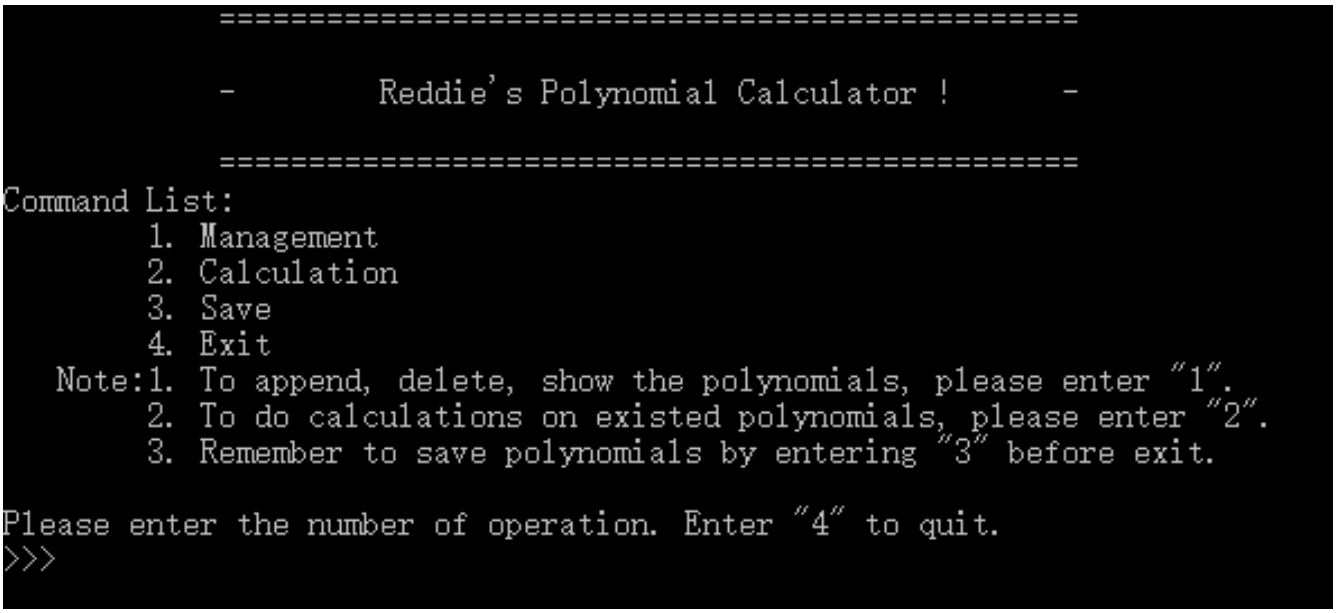
```
// Module
void managePolynomials(Polynomials_& PolyAll);
void calPolynomials(Polynomials_& PolyAll);
void readPolynomials(Polynomials_& PolyAll);
void savePolynomials(Polynomials_& PolyAll);
// Magagement
void appendPolynomials(Polynomials_& PolyAll);
void deletePolynomials(Polynomials_& PolyAll);
void showPolynomials(Polynomials_& PolyAll);
// Calculation
void addPolynomials(Polynomials_& PolyAll);
void subtractPolynomials(Polynomials_& PolyAll);
void multipyPolynomials(Polynomials_& PolyAll);
void derivativeOfPolynomials(Polynomials_& PolyAll);
void equalQPolynomials(Polynomials_& PolyAll);
void assignPolynomials(Polynomials_& PolyAll);
// Data process
void setPolynomial(Polynomial& polyin, string str);
void getNameOfPolynomials(Polynomials_& PolyAll);
// Deal with the input polynomials without names
int Polynomials_::cntVir = 0;
void appendVirtualPoly(Polynomials_& PolyAll, vector<string>& name);
void deleteVirtualPoly(Polynomials_& PolyAll);
// Ask whether to save the result of calculation
void askToSave(Polynomials_& PolyAll, vector<Polynomial>& vecPoly);
// Others
void show_main_manual();
void calculator_interface();
bool validname_test(const string& name);
vector<string> split(const string& input, const string& regex);
```

值得一提的细节：

1. 每个模块均允许输入多个样例，并以END结束输入
2. 在每个计算之前均会给出说明及合法的例子，并且列出存储的多项式的名字
3. 6种计算操作均支持一行内多个输入，并且支持有名函数间运算、直接输入数对运算、或者交叉运算
4. 结束本次计算都会进行询问是否需要保存计算结果
5. 保存时考虑保存的编码是否在范围内、名字是否合法、名字是否冲突等
6. `appendPolynomial` 函数：实现多项式的读入到存储，数对**不一定**得降序输入
7. `setPolynomial` 函数：读入一个字符串（有序对）并对其进行分割（这里采用了正则表达式匹配），并存为pair的形式，以便传入Polynomial类的构造函数
8. `validname_test` 函数：判断名字是否合法，为避免冲突，多项式名只能为字母、短划线、下划线
9. `split` 函数（详情见下面代码）：用C++的正则表达式库编写，实现了与Python的字符串分割函数split类似的功能

```cpp
vector<string> split(const string& input, const string& regex) // split the string
{
    // passing -1 as the submatch index parameter performs splitting
    try{
        std::regex re(regex);
        std::sregex_token_iterator
            first{input.begin(), input.end(), re, -1},
            last;
        return {first, last};
    }
    catch (...)
    {
        cout << "Unknown Error!!!" << endl;
        vector<string> temp;
        return temp;
    }
}
```

4.5 交互界面



- 用<iomanip>控制输出格式确保美观
- 用cin.clear()和cin.sync()清空缓存区，确保每次输入被程序读入后不会有残留，避免干扰下一次操作
- 当字符串长度为0或大于1时循环读入字符串，避免读入空行或者错误输入
- 多重输入判定，如在appendPolynomials模块中（见下），会对输入名字的合法性、名字是否与已有多项式冲突等进行判定，确保输入准确合法

```
367  void addPolynomials(Polynomials_& PolyAll)
368  {
369      cout << endl;
370      cout << "Please enter the names of polynomials or terms of unnamed polynomials you want to add.\n"
371              "You can enter several cases. Every case a line. End by typing \"END\".\n"
372              "The examples listed below are all valid.\n"
373              ">>> polyA polyB polyC\n>>> polyA (1,0)(0,1)\n>>> (1,1)(2,3) (3,1)" << endl;
374      getNameOfPolynomials(PolyAll);
375      cin.clear();cin.sync(); // clear buffer
376      string str;
377      int flag = 0;
378      vector<Polynomial> results;
379      while (1)
380      {
381          cout << ">>> ";
382          getline(cin,str);
383          if (str.size() == 0)
384              continue;
385          if (str.substr(0,3) == "END")
386              break;
387          vector<string> names = split(str," ");
388          appendVirtualPoly(PolyAll,names);
389          vector<Polynomial> polys = PolyAll.getPolys(names);
390          if (polys.empty())
391          {
392              cout << "Error: Match no polynomials." << endl;
393              continue;
394          }
395          if (polys.size() < names.size())
396              cout << "Warning: Some names match no polynomials. Sum of the rest is shown below." << endl;
397          Polynomial resAdd = PolyAll.add(polys);
398          results.push_back(resAdd);
399          cout << resAdd.PolyToString() << endl;
400          deleteVirtualPoly(PolyAll);
401      }
402      askToSave(PolyAll,results);
403      cin.clear();cin.sync();
404  }
```

```
Command List:
        1. Add
        2. Subtraction
        3. Multiplication
        4. Derivative
        5. Equal0
        6. Assignment
        7. Back

Please enter the number of operation. Enter "7" to return.
>>> 1

Please enter the names of polynomials or terms of unnamed polynomials you want to add.
You can enter several cases. Every case a line. End by typing "END".
The examples listed below are all valid.
>>> polyA polyB polyC
>>> polyA (1,0)(0,1)
>>> (1,1)(2,3) (3,1)

The names of polynomials which are stored are listed below:
          polyA          polyB
>>>
```

4.6 输入输出

1. 输出可在命令行中手动输入，或者从文件流中直接读入
2. 输出同时有命令行的输出和文件流的输出（但要选择保存），输出格式同输入，即数对的列举，如$p = (1,2)(3,4)$
3. 具体操作在交互界面均会提示，按照提示要求进行即可

5 程序测试及实验结果

实现了项目要求的所有功能，并且自行添加了multicases、multipolynomials等特性。

写了1000+行代码考虑了各种极端情况，确保程序足够鲁棒。以下是调试时出现的一些问题及解决方案，以及用到的C++11特性：

1. `split`函数在进行正则匹配时常常会导致程序崩溃：
   加上`try,catch`特性捕获并避免发生异常
2. 重载等号运算符后赋值老是会多出一项：
   在等号赋值前一定要将数据全部清空，包括第0项
3. 赋值导致程序崩溃：
   赋值函数引用记得要返回`*this`
4. 必要的地方都加上const限定，防止误操作修改类的私有数据
5. `term.size()`是`unsigned int`类型，对它-1会变为一个超大正数，故要先进行类型转换
6. 使用`auto`自动推断类型
7. 使用`for`的范围语句简化代码
8. 使用`rbegin`和`rend`实现反向容器索引

6 心得体会

1. 第一次写项目写出千行代码，很开心!!!
2. 调bug太难受了!!!