2017 年秋《C++程序设计》大作业

请从以下三道题目中**选择一道**完成。

注意事项：

1. 三道题目难度相当，评分时不存在因题目本身而产生的分数差距。
2. 每道题目至少包含一个`基础`题目和一个`进阶`题目，有的可能包含`困难`题目。`基础`题目**必须完成**，剩下题目选做，完成不同的难度可获得不同的分数。
3. 可以任意按照自己兴趣扩展每道题目。
4. 所有题目需要**考虑非法输入**或**传入参数错误**情况。
5. 对于第二题和第三题，需要加上`帮助`菜单，显示可以调用的功能。

一、计算器

编写程序，设计一款计算器，满足相应的需求。

基础

完成基本算式的计算，运算符包括`+`、`-`、`*`、`/`、幂运算`^`、阶乘`!`。可能的运行情况如下所示：

```
> 1.2+2*3/4+2^3+3! # 程序的输入
16.7 # 计算结果
> 1+ # 错误的表达式
Wrong
>
```

注意：

1. 程序应该一直处于等待输入状态，直到遇到预设退出字符（串）。
2. 程序需要对输入的合法性进行校验。
3. 幂运算需要支持小数。

进阶

在之前的基础上，支持更加复杂的运算（包括但不限于三角函数、对数）且支持括号。可能的运行情况如下所示：

```
> (1+2)*3+sin(30)+cos(60)+tan(45)
11
```

困难

在之前的基础上，支持简单一元函数的求解。可能的运行情况如下所示：

```
> x^2-1=0
1, -1
```

二、矩阵运算工具集

本题为编写相应**模版函数**，支持相应二维矩阵运算。本题一律使用结构 `Matrix` 来表示矩阵，`Matrix` 可能的定义如下：

```
template<typename T>
struct Matrix {
  size_t m, n; // 表示矩阵的行和列
  T** ptr; // 指向数据
};
```

基础

需要支持：

1. 矩阵 element-wise 加减乘除，运算规则即为两个矩阵对应位置元素进行加减乘除，如：

```
A = [
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
B = [
  [2,3,4],
  [5,6,7],
  [8,9,10]
]
则 A-B= [
  [-1,-1,-1],
  [-1,-1,-1],
  [-1,-1,-1]
]
```

2. 矩阵与字面值加减乘除运算，如：

```
A = [
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
A + 1 = [
  [2,3,4],
  [5,6,7],
  [8,9,10]
]
```

3. 矩阵乘法。如果你不了解规则，请参考这里。
4. 若矩阵为二阶或三阶方阵，求对应行列式。如果你不了解规则，请参考这里。

参考：可能的函数声明

如 element-wise 加法运算可声明为：

```cpp
// 该函数完成矩阵 a 和 b 对应位置元素相加。
template<typename T>
Matrix<T> elemendAdd(const Matrix<T>& a, const Matrix<T>& b);
```

又如矩阵乘可声明为：

```cpp
template<typename T>
Matrix<T> mul(const Matrix<T>& a, const Matrix<T>& b);
// 该函数完成矩阵 a 和 b 相乘并将结果返回。
```

再比如求解行列式可声明为：

```cpp
template<typename T>
T getValue(const Matrix<T>& a);
```

进阶

支持将矩阵保存到文件中和从文件中恢复，至少需要支持 `int` 和 `float` 两种数据格式。可能的使用方式为：

```cpp
template<typename T>
struct Matrix {
  size_t m, n;
  T** ptr;

  // 保存到文件中
  void dump(const char* filename);
};
// 你需要实现的两个函数
Matrix<float> load_f(const char* filePath); // For float
Matrix<int> load_i(const char* filePath); // For int

int main() {
  Matrix<float> a_f = load_f("array_float.dat");
  Matrix<int> a_i = load_i("array_int.dat");
  a_f.dump("dump.dat");
}
```

注意：

保存到文件中和从文件中恢复两个函数的参数中**不能包含矩阵的维度信息**，即*在文件中需要保存矩阵的维度信息*。

困难

支持更高阶行列式，并支持多元一次函数求解。

三、字符串工具集

学习 C++ 中 `std::vector` 和 `std::string` 的基本用法，编写相应函数，补充 C++ 字符串函数不支持的操作。

`vector` 学习可参考这里，如果你还想了解更多有关 `std::vector` 的信息，请参考这里。此外，`std::string` 支持的一些残做可以参考这里。

基础

支持以下操作：

1. 字符串分割函数 `split()`，可能的函数声明如下：

   ```cpp
   vector<string> split(const string str, const string sep);
   ```

   用法举例：

```
vector<string> res = split("1,2,3,4", ",");
// res 中应该存放 {"1", "2", "3", "4"}
vector<string> res2 = split("1,2,3,4", ".");
// res2 中应该存放 {"1,2,3,4"}
vector<string> res3 = split("1##2##3##4", "##");
// res3 中应该存放 {"1", "2", "3" , "4"}
```

2. 字符串替换函数 `replace()` 和 `replace_all()`。两者的区别是，`replace` 仅替换第一次出现的字符串，`replace_all()` 将所有出现的字符串都替换。可能的函数声明如下：

```
string replace(const string str, const string p, const string m);
string replace_all(const string str, const string p, const string m);
```

用法举例：

```
string str = "abcdefg";
cout << replace(str, "abc", "1") << endl; // 应该输出 1defg
string str2 = "abcabc";
cout << replace_all(str, "ab", "123") << endl; // 应该输出 123c123c
```

3. 判断字符串是否以某子串开头和结尾 `startWith()` 和 `endWith()` 函数。可能的函数声明如下：

```
bool isStartWith(const string str, const string p);
bool isEndWith(const string str, const string p);
```

用法举例：

```
string str = "abcdefg";
cout << isStartWith(str, "bcd") << endl; // False
cout << isEndWith(str, "efg") << endl; // True
```

进阶

支持 `format` 操作，用法举例：

```
string str = "Hi {1}, I'm {0}, from {2}";
cout << format(str, {"Mac", "Bob", "China"}) << endl; // 应该输出 "Hi Bob, I'm Mac, from China"
```

用 `format` 函数的第二个参数中对应的字符串替换第一个参数中的占位符。占位符定义为 `{INDEX}`，其中 `INDEX` 从 `0` 开始。

注意处理非法情况：

下面情况实际为合法情况：

```
string str = "Hi {1}, I'm {0}, from {2}, {a}";
cout << format(str, {"Mac", "Bob", "China"}) << endl; // 应该输出 "Hi Bob, I'm Mac, from China, {a}"
```

但下面情况为非法情况：

```
string str = "Hi {1}, I'm {0}, from {2}, {3}";
cout << format(str, {"Mac", "Bob", "China"}) << endl; // 这样调用非法，因为 3 超出传入参数个数。可以输出空字符串，也可以输出错误提示信息
```