Report for Project 3: Calculator

Demand Analysis

支持多种复合运算

- 基本的加减乘除、取相反数
- 三角函数(sin, cos, tan)
- 括号带来的优先级变化
- 考虑采用逆波兰表达式的思想加以实现

支持表达式的合法性检查和错误展示

- 对输入的表达式先检查，再计算
- 错误部分高亮显示
- 显式指出错误类型

支持历史记录的查看

- 上下键查看最近的5条计算记录
- 需要加入文件来支持存储记录

基础的界面支持

- 用户应能连续计算和查看记录
- 在各种模式下有自由的切换
- 明确退出后才会离开整个程序

Module Design

`FYI：具体设计细节在注释中给出`

Calculator

由于此次项目的整体逻辑相对单一，难点主要在表达式的计算方法上，所以仅用一个Calculator类来实现计算器所需的所有逻辑操作，包括：

- 计算器的初始化，从文件中读取计算记录
- 单次计算过程：
    - 检查输入表达式的正确性，并将原来的中缀表达式转换为后缀表达式
        - 若表达式非法，进一步告知用户错误的位置和错误的类型，即退出当前计算
        - 否则，继续计算转换后得到的后缀表达式
    - 计算正常的情形下，输出结果并加入记录

- 输入`'r'`查看最近计算记录，可用上下键滚动查看
- 输入`'q'`退出程序，此时需将最新记录保存至文件

```
class Calculator
{
private:
    vector<pair<string, double>> records;   //最近计算记录，最多5条
    string infixExpr;   //当前输入的中缀表达式
    vector<exprElem> postfixExpr;   //转换后得到的后缀表达式
    char prev, cur; //前一个、当前扫描的字符
    double result;  //运算结果
    bool checkSyntax(); //检查表达式合法性
    bool in2postExpr(); //中缀转后缀表达式
    bool checkParen();  //检查括号匹配，最先进行
    double computePostfixExpr();    //计算后缀表达式
    unsigned int findErr(); //查找已发现的错误在原始表达式中的位置
    void displayErr(unsigned int pos);  //高亮展示错误位置
public:
    Calculator();   //导入计算记录
    ~Calculator();  //导出计算记录
    void compute(string expr);  //供外界调用的计算接口
    void displayRecords();  //供外界调用的查看记录接口
};
```

Novelty & Extra

实现了额外的**查看记录功能**（上下键翻阅）和**错误展示功能**（高亮显示、错误类型）。

Problems & Solutions

P：项目的核心难点在于实现后缀表达式的计算和合法性的检查，如何合理地统筹这两点是一个值得认真思考的问题。

S：我采取了**逆波兰表达式**的思想，这也是计算后缀表达式的经典算法。首先对不同运算符赋以**不同的优先级**，通过**栈**的帮助将后缀表达式转化为中缀表达式。**与传统思路不同的是，我在这一阶段就完成了数字的解析，并以统一的形式存储起来**（见下方代码）。而合法性检查也是在转换过程中**同时进行**的，主要思路是**判断prev、cur的类型组合是否满足要求**。

```
//后缀表达式中的元素可能是运算符或操作数
//故建立一个统一的元素类型
struct exprElem
{
    elemType et;    //标识类型
    char opt;
    double opd;
    exprElem(elemType et, char c, double d) :
        et(et), opt(c), opd(d) {}
};
```

P：由于在进行合法性检查时，此时的表达式**已经经过一些预处理**，所以发现错误后正确地定位其在原字符串中的位置比较麻烦。

S：通过`findErr()`再次查找原字符串中prev和cur的组合，进而展示错误位置。

P：使用`_getch()`获取上下键字符时出现令人困惑的问题，即一次按键会引发两次捕获，且仅有第二次是符合预定逻辑的。

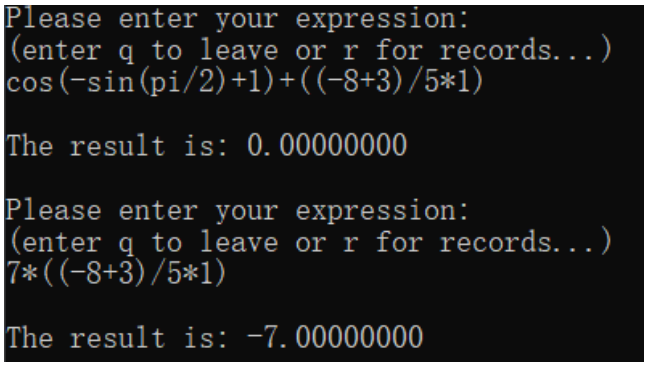
S：在上网查阅资料后发现，一些特殊按键的敲击（包括方位键），往往需要两字节的存储，而`_getch()`仅能捕获一字节，从而引发错误。解决方案就是**仅处理第二次捕获的字符**。

P：由于表达式中的非数字类型往往被视作运算符，所以`pi`的处理比较麻烦。

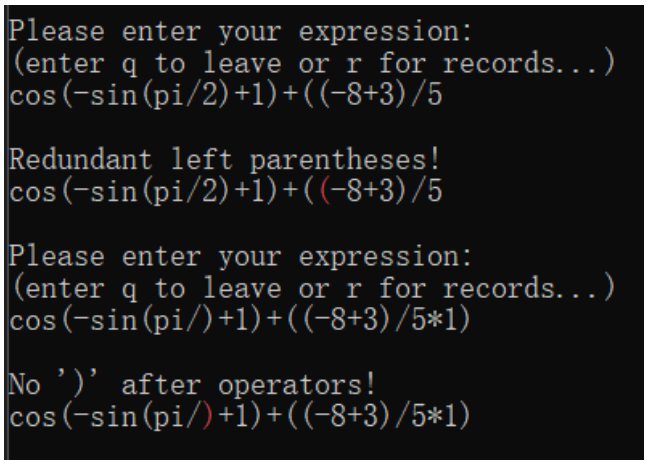
S：在**预处理**阶段即将其转化为具体的数值，方便后续操作。

Testing Snapshots

输入表达式正确计算：



输入非法表达式：



查看最近记录：

```
Please enter your expression:
(enter q to leave or r for records...)
r

Please check your computation records using UP/DOWN Arrow:
(Press ESC to leave...)

expression: 5+3/6
result:5.50000000

expression: -sin(pi/2)
result:-1.00000000

expression: (-8+3)/5
result:-1.00000000
```