多项式与表达式求值计算器

编译启动命令

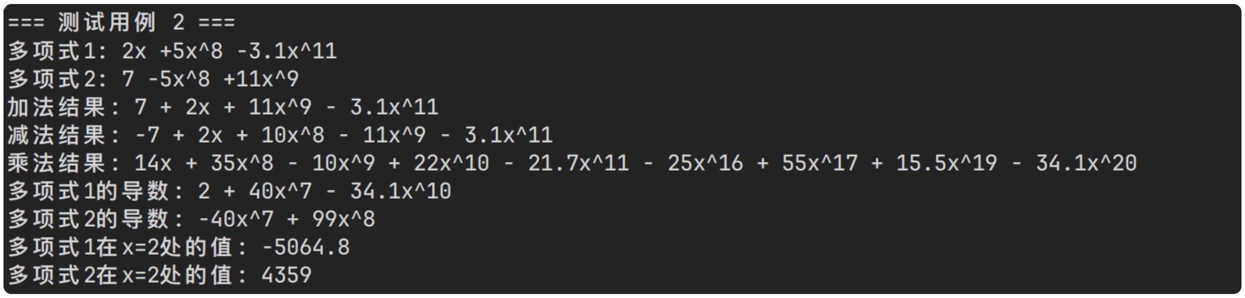
```
cd /home/Halllo/Projects/Homework-algorithm/Project1
g++ -std=c++17 -I./src -I. cmd/A/main.cpp -o build/main_A

cd /home/Halllo/Projects/Homework-algorithm/Project1
g++ -std=c++17 -I./src -I. cmd/B/main.cpp -o build/main_B
```

程序运行结果

多项式计算

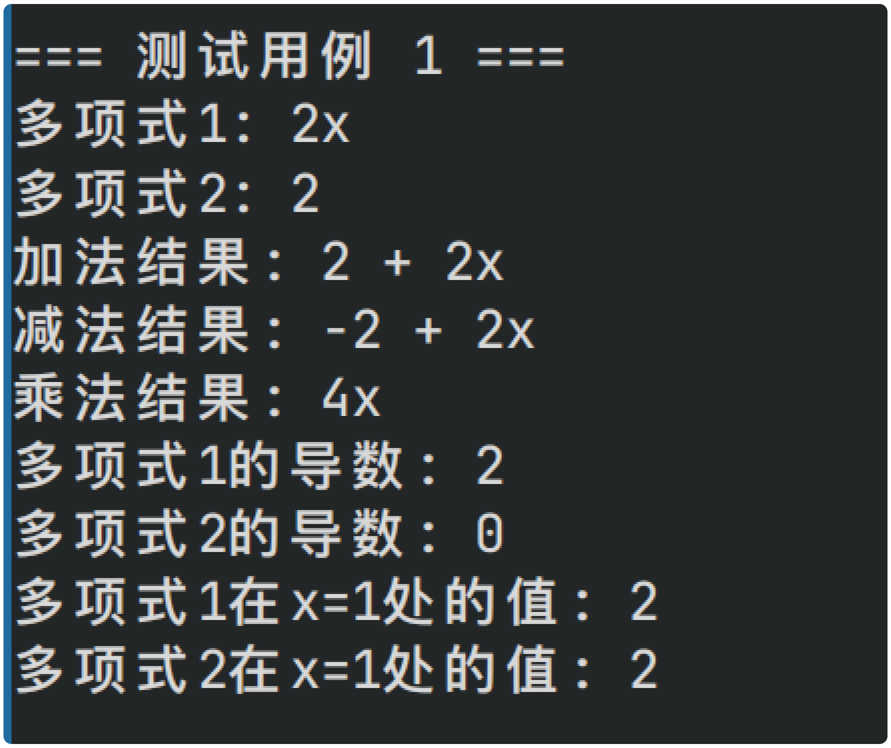
- **输入1**： `(2x + 5x^8 - 3.1x^11) & (7 - 5x^8 + 11x^9)`
- **输出1**：



```
=== 测试用例 2 ===
多项式1: 2x +5x^8 -3.1x^11
多项式2: 7 -5x^8 +11x^9
加法结果： 7 + 2x + 11x^9 - 3.1x^11
减法结果： -7 + 2x + 10x^8 - 11x^9 - 3.1x^11
乘法结果： 14x + 35x^8 - 10x^9 + 22x^10 - 21.7x^11 - 25x^16 + 55x^17 + 15.5x^19 - 34.1x^20
多项式1的导数： 2 + 40x^7 - 34.1x^10
多项式2的导数： -40x^7 + 99x^8
多项式1在x=2处的值： -5064.8
多项式2在x=2处的值： 4359
```

- **输入2**: `2x & 2`

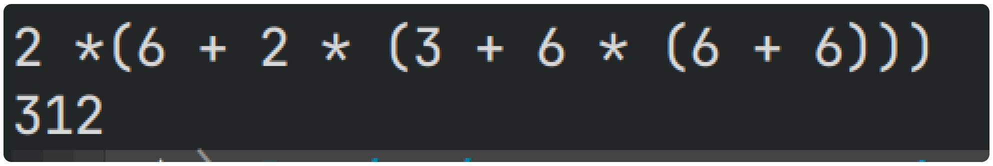
- **输出2**：



```
=== 测试用例 1 ===
多项式1: 2x
多项式2: 2
加法结果: 2 + 2x
减法结果: -2 + 2x
乘法结果: 4x
多项式1的导数: 2
多项式2的导数: 0
多项式1在x=1处的值: 2
多项式2在x=1处的值: 2
```

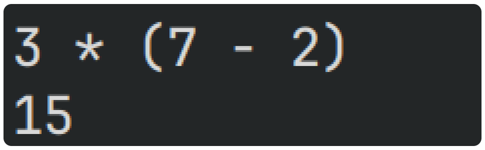
算术表达式求值

- **输入1**: `2 *(6 + 2 * (3 + 6 * (6 + 6)))`
- **输出1**：



```
2 *(6 + 2 * (3 + 6 * (6 + 6)))
312
```

- **输入2**： `3 * (7 - 2)`
- **输出2**：



```
3 * (7 - 2)
15
```

多项式计算关键代码

解析多项式输入

```
// 解析多项式输入

void ParsePolynomial(const std::string& input, SeqList<double>& coeffs,
SeqList<int>& powers) {
    std::istringstream iss(input);
    std::string token;
    while (iss >> token) {
        // 判断系数正负
        if (token == "+") continue;
        bool negative = false;
        if (token[0] == '-') {
        negative = true;
        token = token.substr(1);
    }

    size_t x_pos = token.find('x');
    double coeff;
    int power = 0;

    if (x_pos == std::string::npos) {
        // 常数项
        coeff = ToDouble(token);
        power = 0;
        } else {
        // 处理系数
        if (x_pos == 0) {
            coeff = 1.0; // 针对幂次为0的特殊判断
        } else {
            coeff = ToDouble(token.substr(0, x_pos));
        }

        // 处理指数
        if (x_pos == token.length() - 1) {
            power = 1;
        } else if (token[x_pos + 1] == '^') {
            power = std::stoi(token.substr(x_pos + 2));
        } else {
            power = 1;
        }
    }

    if (negative) coeff = -coeff; // 转换负系数

    coeffs.Push_back(coeff);
    powers.Push_back(power);
    }
}
```

合并同类项

```
// 合并同类项

void Combine(SeqList<double>& coeffs, SeqList<int>& powers) {
    // 转换为数组
    PolyTerm terms[100];
    int termCount = coeffs.Length();

    for (int i = 0; i < termCount; ++i) {
        terms[i].coeff = coeffs.Get(i);
        terms[i].power = powers.Get(i);
    }

    // 按幂次排序
    std::sort(terms, terms + termCount, compareTerms);

    // 合并同类项
    int newTermCount = 0;
    for (int i = 0; i < termCount; ) {
        double sum = terms[i].coeff;
        int j = i + 1;

        while (j < termCount && terms[j].power == terms[i].power) {
            sum += terms[j].coeff;
            j++;
        }

        // 防止 double 计算存在误差
        if (fabs(sum) > 1e-10) { // 忽略极小系数
            terms[newTermCount].coeff = sum;
            terms[newTermCount].power = terms[i].power;
            newTermCount++;
        }

        i = j;
    }

    // 更新回SeqList

    coeffs.Clear();
    powers.Clear();

    for (int i = 0; i < newTermCount; ++i) {
        coeffs.Push_back(terms[i].coeff);
        powers.Push_back(terms[i].power);
    }
}
```

加法

```
// 多项式加法

void PolyAdd(const SeqList<double>& coeffs1, const SeqList<int>&
powers1,
    const SeqList<double>& coeffs2, const SeqList<int>& powers2,
    SeqList<double>& resCoeffs, SeqList<int>& resPowers) {
    // 合并两个多项式
    for (int i = 0; i < coeffs1.Length(); ++i) {
        resCoeffs.Push_back(coeffs1.Get(i));
        resPowers.Push_back(powers1.Get(i));
    }

    for (int i = 0; i < coeffs2.Length(); ++i) {
        resCoeffs.Push_back(coeffs2.Get(i));
        resPowers.Push_back(powers2.Get(i));
    }
    // 合并同类项
    Combine(resCoeffs, resPowers);
}
```

乘法

```
// 多项式乘法
void PolyMul(const SeqList<double>& coeffs1, const SeqList<int>&
powers1,
    const SeqList<double>& coeffs2, const SeqList<int>& powers2,
    SeqList<double>& resCoeffs, SeqList<int>& resPowers) {
    // 执行乘法
    for (int i = 0; i < coeffs1.Length(); ++i) {
        for (int j = 0; j < coeffs2.Length(); ++j) {
            double coeff = coeffs1.Get(i) * coeffs2.Get(j);
            int power = powers1.Get(i) + powers2.Get(j);
            resCoeffs.Push_back(coeff);
            resPowers.Push_back(power);
            }
        }
        // 合并同类项
        Combine(resCoeffs, resPowers);
}
```

求导

```
void PolyDerivative(const SeqList<double>& coeffs, const SeqList<int>&
powers, SeqList<double>& derivCoeffs, SeqList<int>& derivPowers) {
    for (int i = 0; i < coeffs.Length(); ++i) {
        double coeff = coeffs.Get(i);
        int power = powers.Get(i);
        if (power > 0) { derivCoeffs.Push_back(coeff * power);
            derivPowers.Push_back(power - 1);
        }
    }
}
```

赋值

```
double PolyEvaluate(const SeqList<double>& coeffs, const SeqList<int>&
powers, double x) {
    double result = 0.0;
    for (int i = 0; i < coeffs.Length(); ++i) {
        double coeff = coeffs.Get(i);
        int power = powers.Get(i);
        result += coeff * pow(x, power);
    }
    return result;
}
```

数值计算器关键代码

分割 token

```
int Spilt(std::string &s, std::string tokens[], int maxTokens) {
    int cnt = 0;
    int i = 0;

    while (i < s.length() && cnt < maxTokens) {
        if (isspace(s[i])) {
            i++;
            continue;
        }

        if (isdigit(s[i]) || (s[i] == '.' && i + 1 < s.length() && isdigit(s[i + 1]))) {
            std::string num;
            bool isFloat = false;
            while (i < s.length() && (isdigit(s[i]) || (s[i] == '.' && !isFloat))) {
                if (s[i] == '.') isFloat = true;
                    num += s[i];
                    i++;
                }
            tokens[cnt++] = num;
            continue;
        }

        if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' ||
        s[i] == '(' || s[i] == ')') {
            tokens[cnt++] = std::string(1, s[i]);
            i++;
        }
    }

    return cnt;

}
```

计算

```
double EvaluateExpression(const std::string &input) {
    Stack<std::string> op_stack;
    Stack<double> num_stack;
    int maxTokens = input.length();
    std::string tokens[MAXTOKEN];

    int cnt = Spilt(const_cast<std::string&>(input), tokens, maxTokens);

    for (int i = 0; i < cnt; i++) {
        std::string c = tokens[i];
        if (c == "+" || c == "-" || c == "*" || c == "/") {
            while (!op_stack.isEmpty() && level(op_stack.Top()) >= level(c) && op_stack.Top() != "(") {
                std::string c1 = op_stack.Top(); op_stack.Pop();
                double n1 = num_stack.Top(); num_stack.Pop();
                double n2 = num_stack.Top(); num_stack.Pop();
                double n3 = cal(c1, n2, n1);
                num_stack.Push(n3);
            }
            op_stack.Push(c);
        } else if (c == "(") {
            op_stack.Push(c);
        } else if (c == ")") {
            while (!op_stack.isEmpty() && op_stack.Top() != "(") {
                std::string c1 = op_stack.Top(); op_stack.Pop();
                double n1 = num_stack.Top(); num_stack.Pop();
                double n2 = num_stack.Top(); num_stack.Pop();

                double n3 = cal(c1, n2, n1);
                num_stack.Push(n3);
            }

            if (!op_stack.isEmpty()) op_stack.Pop();
        } else {
            num_stack.Push(std::stod(c));
        }
    }

        while (!op_stack.isEmpty()) {
            std::string c1 = op_stack.Top(); op_stack.Pop();
            double n1 = num_stack.Top(); num_stack.Pop();
            double n2 = num_stack.Top(); num_stack.Pop();

            double n3 = cal(c1, n2, n1);
            num_stack.Push(n3);
        }
    return num_stack.Top();
}
```

错误处理

除0错误

```
if (c == "/") {
    if (b == 0) {
        throw std::runtime_error("除零错误");
    }
    return a / b;
}
```

检查无效字符

```
if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' ||
    s[i] == '(' || s[i] == ')') {
    tokens[cnt++] = std::string(1, s[i]);
    i++;
} else {
    throw std::runtime_error("无效字符： " + std::string(1, s[i]));
}
```

检查操作数

```
if (num_stack.Length() < 2) {
    throw std::runtime_error("表达式语法错误：操作数不足");
}
```

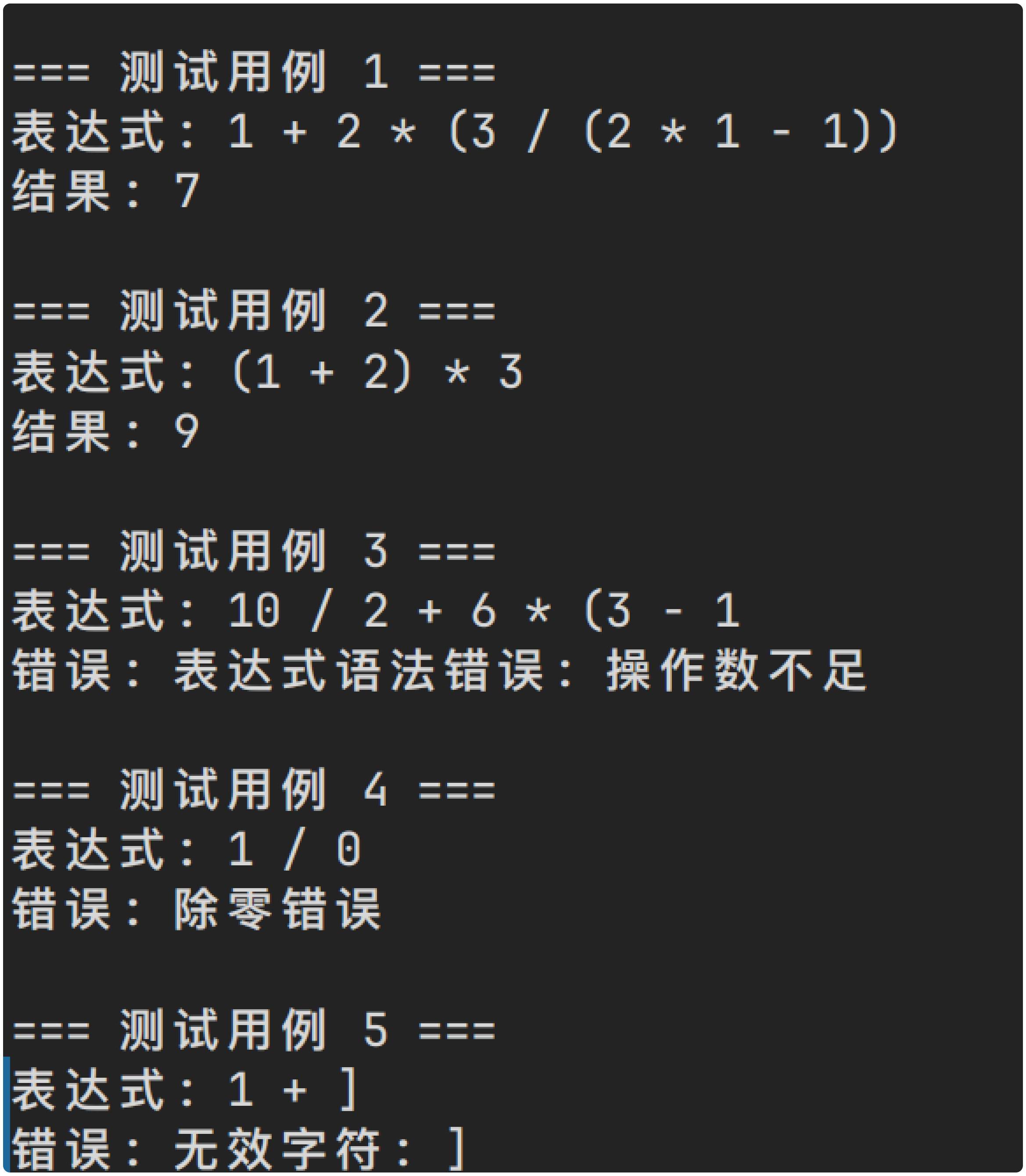
检查括号数

```
else if (c == ")") {

    while (!op_stack.isEmpty() && op_stack.Top() != "(") {
        std::string c1 = op_stack.Top(); op_stack.Pop();
        if (num_stack.Length() < 2) {
            throw std::runtime_error("表达式语法错误：操作数不足");
        }
        double n1 = num_stack.Top(); num_stack.Pop();
        double n2 = num_stack.Top(); num_stack.Pop();
        double n3 = cal(c1, n2, n1);
        num_stack.Push(n3);
    }

    if (op_stack.isEmpty()) {
        throw std::runtime_error("括号不匹配：缺少左括号");
    }
    op_stack.Pop();
}
```

输出结果



```
=== 测试用例 1 ===
表达式：1 + 2 * (3 / (2 * 1 - 1))
结果：7

=== 测试用例 2 ===
表达式：(1 + 2) * 3
结果：9

=== 测试用例 3 ===
表达式：10 / 2 + 6 * (3 - 1
错误：表达式语法错误：操作数不足

=== 测试用例 4 ===
表达式：1 / 0
错误：除零错误

=== 测试用例 5 ===
表达式：1 + ]
错误：无效字符：]
```