计算器

Calculator.c

程序效果演示

得分

必做部分：70分

选做部分：

- 连续变量赋值 10分
- 浮点数 10分
- 负数 10分

总览

1. 词法分析
2. 语法分析 & 表达式求值
3. 变量赋值

例子：4 + 3 * (2 - 1)

词法分析

按照分隔符（一个空格）将每个字符串分段，并求出每一段（称为 token）的类型

3.1 字符集

定义本项目的所要支持的表达式求值语言定义在以下字符集上，也就是说，除去用于区分 TOKEN 之间的空格 `' '(ASCII = 32)` ，在输入中仅可能出现以下字符。

$$\Sigma = \{=, +, -, *, /, (,), \_, [0-9], [a-z], [A-Z]\}$$

如果你选做了 Bonus #2 浮点数支持，那么 $\Sigma' = \Sigma \cup \{.\}$ 。

3.2 token

本项目中会出现的所有 token 类型和其合法性定义如下。

1. 变量（variable）：由字母、数字、下划线组成，但不能以数字开头。
2. 整数（integer）：由数字组成，不能含有前导 0（比如 `002`）。
3. 运算符（operator）：`+, -, *, /, (, ), =` 。
4. * 浮点数（float）：由数字、小数点 `'.'` 和数字连接而成，其中前后的数字必须出现且合法，例如 `3.2` 和 `0.03` 都是合法的浮点数，而 `00.1` 和 `.233333` 和 `22.` 均不是合法的浮点数。**请注意这个定义，这和 C 语言中不同。**

tokens

```
typedef struct token {
    enum {
        VAR, NUM, OP // 变量、数（整数、浮点数）、运算符
    } type;
    char str[32];
} Token;

Token tokens[1024]; // 将所有的token储存在一个连续的数组中
```

NUM	OP	NUM	OP	OP	NUM	OP	NUM	OP
4	+	3	*	(	2	-	1	)

tips

读输入的一种方法：

```
int n = 0;
while (scanf("%s", tokens[n++].str) != EOF) {
    char c = getchar();
    if (c == '\n') {
        // do someting
        n = 0;
    }
}
```

获取token中存储的整数、浮点数：

```
int iVal;
sscanf(token.str, "%d", &iVal);
double fVal;
sscanf(token.str, "%lf", &fVal);
```

语法分析 & 表达式求值

语法分析：分析表达式逻辑的正确性

表达式求值：递归求值

语法分析步骤通常可以和表达式求值一起完成

求值函数

使用两个整数 l 和 r 来指示表达式的开始位置和结束位置

```
eval(l, r) {
    if (l > r) {
        // 错误的表达式，返回error
    } else if (l == r) {
        // 单个token，返回数字或变量的值
    } else if (checkParentheses(l, r)) {
        return eval(l + 1, r - 1); // 去掉两侧的括号
    } else if (checkMinus(l, r)) {
        return minus(eval(l + 1, r)); // 解引用负号
    } else {
        op = 表达式中主运算符的索引;
        val1 = eval(l, op - 1);
        val2 = eval(op + 1, r);
        return meetValue(val1, val2, op);
    }
}
```

checkParentheses()

判断是否需要去掉表达式两侧的括号

1. 表达式是最左边/最右边不是括号，返回false
   1 + 1
2. 去掉最左边和最右边的括号后，表达式的左右括号不匹配，返回false
   (1 + 1) + (1 + 1)

checkMinus()

判断是否需要解引用负号

- 负号的优先级非常低，和表达式最外层包裹的配对括号类似
- 只有当找不到主运算符的时候，才会考虑“解引用”这个负号

思路：

1. 不是以`-`开头，返回false
2. 扫描所有token，尝试寻找主运算符，找不到返回false

寻找主运算符思路

for循环遍历所有**token**

- 非运算符的**token**不是主运算符，continue
- 出现在一对括号中的**token**不是主运算符，continue
  如何判断是否在一对括号中？
  【栈顶指针top】
- 主运算符的优先级在表达式中是最低的
  当有多个运算符的优先级都是最低时，根据结合性，最后被结合的运算符才是主运算符
  【从左到右扫描，遇到优先级更低或相同的，替换】

减号还是负号？

如何判断遇到的`-`是减号还是负号？

根据前一个token判断，不是减号的都是负号

- 前一个token是变量或数字，下一个`-`是减号
  例子：a - 1， 4 - 1
- 前一个token是右括号，下一个`-`是减号
  例子：(1 + 2) - 3

变量赋值

推荐用一个结构体来记录变量和对应的值

```
typedef struct assignment {
    char name[32];
    Value val;
} Assignment;

Assignment vars[128];
int varsLen = 0;
```

```
typedef struct value {
    enum {
        INT,
        FLOAT,
        ERROR
    } type;
    union {
        int iVal;
        double fVal;
    } val;
} Value;
```

变量赋值

```
Value evalAssign(int l, int r) {
    if (there exists "=") { // 赋值语句
        Variable var = ...
        Value val = evalAssign(...) // 递归处理连续赋值
        save(var, val)
    } else { // 表达式语句
        return eval(...)
    }
}
```

```
a = b = c = 1
```

类型提升

可以在meetValue()中处理类型提升的过程

```
Value meetValue(Value v1, Value v2, int op) {
    ...
    if (v1.type != v2.type) {
        // 类型提升
    }
    ...
}
```

错误处理

词法错误：不合法的token，无法与变量、整数、运算符、浮点数任一类型匹配
词法错误的分析应该在 `词法分析` 阶段完成

语法错误：字符串无法与BNF范式的任何一种展开相匹配，在表达式求值的过程中调用了当前未定义的变量
语法错误的分析应该在 `语法分析 & 表达式求值` 阶段完成，仅借助eval()函数的返回值已经足够处理语法错误的所有情况

- l > r，错误的表达式
- l == r，剩下的单个token是运算符
- 括号不匹配
- 找不到主运算符