

编号: _____

实习 成绩	一	二	三	四	五	六	七	八	九	十	总评	教师签名

武汉大学计算机学院

本科生实验报告

《解释器构造》实验

计算表达式分析器构造实验

编 号: _____

实习题目: _____

专业 (班): _____ 软件工程

团队成员一: _____ 任思远 (2016302580320)

团队成员二: _____ 黎冠延 (2016302580264)

团队成员三: _____ 刘瑞康 (2016302580242)

团队成员四: _____ 朱 申 (2016302580074)

指导教师: _____ 杜 卓 敏

2 0 1 8 年 1 2 月 2 9 日

郑 重 声 明

本团队呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

团队成员签名：_____

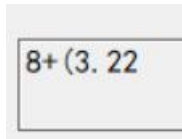
日期：_____

目 录

第一部分	即时解析的图形界面计算器	3
第二部分	允许单字符回退的图形界面计算器	4
2.1	图形界面计算器概述		
2.2	图形界面计算器特征		
2.3	图形界面计算器状态变化图		
第三部分	交互式计算器	5
第四部分	用 Rules Translator 实现的交互式计算器	6
4.1	交互式计算器概述		
4.2	交互式计算器原理		

第一部分 即时解析的图形界面计算器

这个计算器在输入时进行实时解析，并且限制下一步的错误输入，比如：“3+(2”，这是按下“(”符号就不会产生任何变化。另外，这个计算器还支持 token 级别的回退，即

A screenshot of a calculator's display area. It shows the expression "8+(3.22" in a monospaced font. The text is contained within a rectangular frame that represents the display's border.

按下回退键，将变为：

A screenshot of a calculator's display area after a backspace operation. It shows the expression "8+(" in a monospaced font. The text is contained within a rectangular frame that represents the display's border.

实现的主要思路：使用了一些变量特征来隐式地维护了整个计算式的状态变化。

如上一个符号的类型，左括号与右括号数量之差等，通过不同状态间的跳转来实现计算式的实时解析。

第二部分 允许单字符回退的图形界面计算器

2.1 图形界面计算器概述

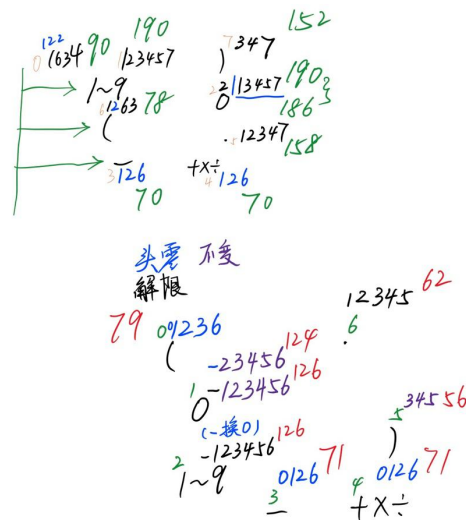
这个计算器允许单个字符回退，但是不是对每个输入进行计算的，它的计算部分是首先使用字符串把内存进行存储然后再统一进行计算，但是每个字符的输入，会让状态栈产生变化，以此激活或者是封闭某些按键的点击操作。即不允许一些情况下的按键的点击操作，通过这样的方式来防止输入过程中出现的不符合语法的问题。

除以零的问题会通过弹窗提示错误来解决。

2.2 图形计算器特征

- A. 不允许语法错误的输入，例如多个小数点的问题。
- B. 同时包括不允许多个开头 0，如果输入了第一个是 0 且后面跟着的不是小数点而是其他的数字则会将 0 取代。
- C. 支持括号运算，如果括号不对等将会不能点击等号进行运算。

2.3 图形界面计算器状态变化图



第三部分 交互式计算器

这个计算器是下一个计算器用到的 Rules Translator 实现之前的一个版本。

```
input format:
1. expr      -> like "a=(3+4.8)*-.3 + 9/4.5" or "b =a= (a + + 7.1 ) -.722"
2. print(expr) -> output the value of the expression.
3. dict      -> output the dictionary table.

-----

1 >>> a = (1+2)/2
2 >>> print(a)
1.5
3 >>> b=2*a+1.2
4 >>> dict
a          1.5
b          4.2
5 >>>
```

如图所示，这个计算器支持算术表达式的计算，变量的赋值，并且还支持打印和维护一个变量字典。

具体思路是先做词法分析，然后使用两个栈，直接计算中缀表达式。

第四部分 用 Rules Translator 实现的交互式计算器

4.1 交互式计算器概述

本计算器是开发了 Rules Translator 以后进行测试用的一个计算器，一开始只允许进行单字符的数字运算，后面经过完善可以支持完整的计算器操作。其实现是通过语法制导翻译进行的。不需要借助后缀表达式即可执行计算。在每次规约的情况下就可以执行单步的计算。

因为它的实现方式是 Rules Translator，所以它有明确的整体产生式，而且它是直接通过语法产生式来进行翻译，所以每一步的语义过程也是有明确的规定。

4.2 交互式计算器原理

这个文件是 Rules Translator 的格式实例文件，但是因为内容不多，就也粘贴在下文。

```
#include <stdio.h>
#include <iostream>
#include "../extern/rules_translator.h"

// tmn 只是一个名称，这个名称可以任意自定义，也就是这个类名可以变换为任意字符串，
// 不过请不要替换为 terminate，因为是关键字。
enum class tmn {
    add, sub, mul, div, lbkt, rbkt, singlenum, doc
};

// 必须要给一个类型获取函数，用以映射从 token 到终结符类型
tmn getType(const char &c) {
    switch (c) {
        case '+':
            return tmn::add;
        case '-':
            return tmn::sub;
        case '.':
            return tmn::doc;
        case '*':
            return tmn::mul;
        case '/':
```

```

        return tmn::div;
    case '(':
        return tmn::lbkt;
    case ')':
        return tmn::rbkt;
    default:
        break;
}
if (c >= '0' && c <= '9')
    return tmn::singlenum;
throw std::string("Not a valid token");
}
// 如果$<num>指代的内容是一个终结符，则会将 token_t 本身替换它。

```

```

using ll = long long;
// 这里必须要声明下面会用到的 getValue 函数，但是可以不实现它。
ll getValue(const char &c);
double unionDouble(ll &k1, double &&k2) {
    while (int(k2)) k2 /= 10;
    return k1 + k2;
}

```

```

struct expr {
    double val;
};

```

```

using sub_count = size_t;

```

// 最好放在一个命名空间中，就不会发生命名冲突

```

namespace calculator_stuff {
    ``tsl
    terminate = enum class tmn {
        add, sub, mul, div, lbkt, rbkt, singlenum, doc
    };
    token_type = char; // 这里标记传进来的 token 的类型
    get_type = getType; // 这个函数的参数类型乱写行为未定义，函数原型必须是： terminate
    (*)(const token_type &);
    using e = expr;
    using t = expr;
    using f = expr;
    using n = double;
    using k = ll;
    using sub_seq = sub_count;
}

```



```

s := e                                { printf("%lf\n", $1.val); }
e := e add t                          { $$val = $1.val + $3.val; }
  | e sub t                          { $$val = $1.val - $3.val; }
  | t                                { $$val = $1.val; }
t := t mul f                          { $$val = $1.val * $3.val; }
  | t div f                          { $$val = $1.val / $3.val; }
  | f                                { $$val = $1.val; }
f := lbkt e rbkt                      { $$val = $2.val; }
  | sub_seq lbkt e rbkt { $$val = $1 % 2 == 0 ? $3.val : -$3.val; }
  | plus_seq lbkt e rbkt { $$val = $3.val; }
  | n                                { $$val = $1; }
n := k doc k                          { $$ = unionDouble($1, $3); }
  | k                                { $$ = $1; }
k := k singlenum                      { $$ = $1 * 10 + ($1 > 0 ? getValue($2) : -getValue($2)); }
  | singlenum                      { $$ = getValue($1); }
  | sub_seq singlenum              { $$ = $1 % 2 == 0 ? getValue($2) : -getValue($2); }
  | plus_seq singlenum             { $$ = getValue($2); }
sub_seq := sub_seq sub                { $$ = $1 + 1; }
  | sub                            { $$ = 1; }
plus_seq := plus_seq add
  | add;
...
}

```

// 这里实现 getValue()也是 OK 的。

```

ll getValue(const char &c) {
    return c - '0';
}

```

它是通过结合 C++ 的内容，这也展现了 Rules Translator 内容里面的 analyze 函数可以接受任意重载了下标运算符和 length() 函数即可进行分析的好处 —— 它不仅允许 vector，还可以直接接受 string，所以分析的时候直接把去掉分隔符号的 string 输入进去即可。

可以看到上文首先定义了一个终结符号的 enum class，因为要方便下文的书写，所以实际上 tsl 部分的 enum class 只是一个完全复制版，不会被再次打印，因为上文其实也是要使用。

然后定义了一个映射函数 —— 因为具体到下文需要进行一下 token（例如这里是 char type）到 enum class 具体值的映射。

这里的权衡：为什么不是要求 token 的类型重载一个 get_type 函数，而是要求统一写映射，因为这样其实更加方便外部使用，例如现在可以直接用 string 而不需要一定用 vector。

然后定义了一个用于绑定在非终结符的 C++ 类（结构体）expr。下文可以直接进

行使用。

然后是翻译的地方写了语法制导翻译的过程，通过不断取值进行计算。

下面是调用这个生成之后结果的代码：

```
void test_Sample() {
    std::string s;
    std::cout << "Input a formula" << std::endl;
    std::getline(cin, s);
    while (s != "q") {
        try {
            calculator_stuff::analyze(rules_translator::utils::trimDivider(s));
        }
        catch (string &e) {
            std::cout << e << ": (input content)" << s << std::endl;
        }
        std::getline(cin, s);
    }
};
```

通过这种方式就可以调用生成的内容。