C++面向对象技术实验报告

实验一

编写/调试桌面计算机程序

学　　号：**PE20060014**

姓　　名：王晨

指导老师：谢兴生

2020 年 10 月 31 日

一、 实验条件

1、 硬件条件：Mac 一台

2、 软件条件：Mac OS Catalina

Clion 2020

二、 实验过程

一、初步的函数结构

```
Token get_token();    // 分词流处理函数，读入并处理传入的分词
double expression();  // 计算函数，处理符号+/-并返回计算结果
double term ();       // 处理符号*/%的函数
double primary();     // 数字和括号处理
main();               // 主函数
```

二、主要的类和函数的具体实现

1.Token 的静态结构

```
class Token {
public:
  char kind;                    // 类别
  double value;                 // 对于数字为具体值
  Token(char ch)                // 获取 char
        :kind(ch), value(0) { }
  Token(char ch, double val)     //获取 char 的类型和值
        :kind(ch), value(val) { }
};
```

2.Token_stream 类，用于依次读取字符，产生 T

```
class Token_stream {
public:
    Token_stream() :full(0), buffer(0) { }      // make a Token_stream that reads from cin
    Token get_token();              // get a Token (get() is defined elsewhere)
    void putback(Token t);        // put a Token back
    void ignore(char);
private:
    bool full;              // 标志位，判断 buffer 中是否已经有 token
```

```
    Token buffer;        // 缓冲区，用于 putback token
};
```

3.putback 函数，用于将 token 存入 buffer 中

```
void Token_stream::putback(Token t)
{
    if (full)
    error("putback() into a full buffer");          //只允许放入一个 token，否则报错
    buffer = t;             // copy t to buffer
    full = true;           // buffer is now full
}
```

4. get_token 函数,用于从 token 流中读取一个 token

```
Token Token_stream::get_token()       // read a token from cin
{
    if (full) {                 //  如果 buffer 中已有 token，则将其取出
        full = false;
        return buffer;
    }
    char ch;
    cin >> ch;           // note that >> skips whitespace (space, newline, tab, etc.)

    switch (ch) {
        case ';':       // for "print"
        case 'q':       // for "quit"
        case '(': case ')': case '+': case '-': case '*': case '/':
            return Token(ch);               // let each character represent itself
        case '.':
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
        {
            cin.putback(ch);                // put digit back into the input stream
            double val;
            cin >> val;                      // read a floating-point number
            return Token(number, val); // rather than Token('8',val)
        }
        default:
            error("Bad token");
    }
}
```

5.expression 函数，用于读取和计算得到的 term，可以计算+/-

```
double expression()
{
    double left = term();              // read and evaluate a Term
    Token t = ts.get_token();          // get the next token
    while (true) {
        switch (t.kind) {
            case '+':
                left += term();        // evaluate Term and add
                t = ts.get_token();
                break;
            case '-':
                left -= term();        // evaluate Term and subtract
                t = ts.get_token();
                break;
            default:
                ts.putback(t);
                return left;           // finally: no more + or -: return the answer
        }
    }
}
```

6.term 函数，与 expression 类似，用于处理计算*/%

```
double term()
{
    double left = primary();
    Token t = ts.get_token();          // get the next token
    while (true) {
        switch (t.kind) {
            case '*':
                left *= primary();
                t = ts.get_token();
                break;
            case '/':
            {
                double d = primary();
                if (d == 0) error("divide by zero");          //当除数为 0，报错
                left /= d;
                t = ts.get_token();
                break;
```

```
            }
            default:
                ts.putback(t);
                return left;
        }
    }
}
```

7.primary 函数，用于处理数字和括号

```
double primary()        // read and evaluate a Primary
{
    Token t = ts.get_token();
    switch (t.kind) {
        case '(':                       // handle '(' expression ')'
        {
            double d = expression();
            t = ts.get_token();
            if (t.kind != ')') error("')' expected");
            return d;
        }
        case number:                    // rather than case '8':
            return t.value;             // return the number's value
        default:
            error("primary expected");
    }
}
```

8.calculate 函数，分离读取和计算功能，使得 main 更加简洁。

```
void calculate()
{
    while (cin)
        try{
            cout << prompt;
            Token t = ts.get_token();
            while (t.kind == print) t = ts.get_token(); // first discard all °∞prints°±
            if (t.kind == quit) return; // quit
            ts.putback(t);
            cout << result << expression() << endl;
        }
        catch (exception& e) {
```

```
            cerr << e.what() << endl; // write error message
            //clean_up_mess(); // <<< The tricky part!
        }

}
```

9. clean_up_mess 和 ignore 函数

```
void clean_up_mess()
{
    ts.ignore(print);
}

void Token_stream::ignore(char c)
{
    if (full && c == buffer.kind) {
        full = false;
        return;
    }
    full = false;

    char ch;
    while (cin >> ch)
        if (ch == c) return;
}
```

10.main 函数，

```
int main()
try {
    calculate();
    keep_window_open("~0"); // cope with Windows console mode
    return 0;
}

catch (exception& e) {
    cerr << e.what() << endl;
    keep_window_open("~1");
    return 1;
}
catch (...) {
    cerr << "exception \n";
```

```
    keep_window_open("~2");
    return 2;
}
```

三、 错误和异常分析

1.第一次测试输入 1 2 3 4+5 6+7 8+9 10 11 12，得到结果 1 4 6 8 10，吃掉了两个表达式。
这是由于在原来的 expression 函数中，

```
default:    return left;
```

在获取一个其他字符后，直接返回，该字符后续无法再使用。因此，应该添加 putback 函数,将字符返回到输入流中，使得后面的函数可以继续使用：

```
default:
        ts.putback(t);
        return left;
```

2.第二次测试，输入 2 3 4 2+3 2*3，得到结果 2 3 4 5，第六个结果缺失，这是由于程序还在等待用户的输入，或者说程序没有得到用户明确结束输入的指令。因此需要引入 print result 和 quit 指令，在 main 函数或者 calculate 函数中：

```
while (t.kind == print)
        t = ts.get_token();
if (t.kind == quit)
        return; // quit
```

此时计算器已经可以基本工作。

3.避免非法输入直接退出
采用 keep_window_open("~~")，计算器会在用户输入～～后才会退出。

4.第三次测试，输入 1@¥z;1+3;原有的 clean_up_mess 函数无法处理第二个错误字符，仍然会抛出 bad token，因此需要添加 ignore 函数来处理这部分字符。具体函数见上部分。

5.一些其他错误：
Term()函数中一开始写了：

```
case '%':
left %= primary();
break;
```

由于%对于浮点数，没有定义，编译不能通过。改进方法，将 double 转化成 int 后再转回 double：

```
case ‘%':      // handle ‘%’
```

```
double d = primary();
int i1= int(left);
int i2= int(d);
left = i1 % i2;
break;
```

void Token_stream::putback(Token t) 一开始没有加上 Token_stream::，没有指定成员函数的类，编译无法通过。

四、心得体会

1. 本次实验通过根据书本的引导，学习了桌面计算器的编写和调试，熟悉了 c++语法，学会了一些典型的程序编写思路和错误调试方法。
2. 桌面计算器的完整功能和优化实际上是复杂的，我们只需要根据实际需求和条件逐步优化程序，有时候一些功能的添加和一些错误的修改往往会带来另一些新的底层错误，这时候需要权衡是否有必要，尤其是需要大规模修改代码结构的时候。
3. 尽量使用有意义的名字，例如用宏定义、typedef 方法来替代晦涩难懂的变量名，减少“魔数”。如果一个常量被多次使用，应当对其进行“字符化”以便优化代码可读性和方便修改。
4. 经常性的进行代码测试，而不是一味地增加功能。尽量将功能函数封装起来，只留下其调用接口。