



|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 18040400054 | **姓名** | **周师扬** |
| **班级** | **1803054** | **任课教师** | **张淑平** |
| **实验名称** | 模块编程 | | |
| **实验学期** | **2019 – 2020 学年第2学期** | | |
| **实验日期** | 选择日期 | **实验地点** |  |
| **报告成绩** |  | | |

西安电子科技大学计算机科学与技术学院

# 实验目的

[通过实现一个计算器程序，熟悉 C++为提供的模块与分别编译相关机制，掌握模块化程序设计范型的基本策略，能够灵活运用相应机制，提高编程水平。]

# 实验环境

操作系统：[MacOS]

开发工具：[clion(Mac版)和Xcode]

# 实验内容

## **改造实现桌面计算器**

结合第8、9章、习题 8.5.3 的要求，将第6章的桌面计算器按照以下要求改造实现：

1. 采用多个头文件、多个源文件的方式实现完整程序，使得源代码的物理结构和逻辑结构保持一致；
2. 输入中每遇到一个分号或回车(‘\n’)就认为一个完整表达式结束，并将该完整表达式记作一行。在处理过程中累计行数；
3. 对于输入中存在错误时，除了提示错误现象外，还需提示相应的出错行号。
4. 必须支持命令行参数，用以指明从何处读取数据。命令行参数中的第2项开始为输入文件之路径，可省略。

* 输入文件就是普通的文本文件，其中预先输入了若干表达式，内容和来自标准输入的完全相同。
* 若命令行参数未指明输入文件，则程序从标准输入设备(cin)读取表达式；

若命令行指明了1~N个输入文件，则程序依次从这些文件读入表达式。

# 数据结构与算法设计

## **计算名字的相关数据**

**模块结构及文件组织设计：**

模块1：主控模块，仅包括文件main.cpp ，定义了 main()函数。

模块2：Lexer模块，包括文件lexer.cpp和lexer.hpp。

定义了Lexer模块，包含了一种枚举类型Token\_value，get\_token()函数、paserCommandLine()函数、switch\_input()函数和skip()函数。

模块3：Parser模块，包括文件parser.cpp和parser.h。

定义了exper()函数、term()函数、prim()函数。

模块4：Error模块，包括文件error.cpp和error.h。

定义了error()函数。

**关键数据结构设计：**

数据结构1(main.cpp中的数据结构)：

Lexer::Token\_value curr\_tok=Lexer::PRINT;  
int line=1;//标记错误行号,有错误时输出行号  
map<string, double> table;//一个map类型的表table  
int no\_of\_errors;  
double number\_value;//记录数值  
string string\_value;//记录数值

数据结构2(lexer.cpp中的数据结构):

std::istream \*Lexer::input=nullptr;//定义输入流指针input并且初始化为空指针  
std::vector<std::string> Lexer::fileNames;//用于存储多个文件名  
enum Token\_value{ //定义一种枚举类型，方便读者和使用者的使用  
 NAME, NUMBER, END, PLUS='+',  
 MINUS='-', MUL='\*', DIV='/',  
 PRINT=';', ASSIGN='=', LP='(', RP=')',  
 ERR,  
 };

**算法1.1 int main()**

作 用：主控函数，用于分辨输入的方式是标准输入还是文件输入，也实现对题目所需其他内容的测试。

参 数：int argc,char \*argv[]。

返回值：return no\_of\_error或者return 0。

计算过程：

1. 初始化pi值为3.1415924535，初始化e值为2.178。
2. 判断argc的值，如果大于2，那么就报错“Too many arguments！”，然后返回0；
3. 调用parseCommandLine()函数对命令行进行解析，之后输入，如果遇到空的表达式则跳过，如果遇到表达式有错误，则丢弃错误表达式的多余部分。
4. 如果input不等于标准输入std：：cin，那么就直接delete掉input。

**算法1.2 int Lexer::parseCommandLine()**

作 用：解析命令行参数对应的含义。

参 数：int argc , char \* argv[]。

返回值：return 0。

计算过程：

1. 判断argc值，如果等于1，那么令\*input为标准输入std：：cin。
2. 如果argc值不为1，那么打开第一个输入文件并且判断文件是否成功打开，如果input->good()值为true，则表示文件成功打开，这时剔除第一个文件名。
3. 最后一个文件如果成功打开那么返回0，否则报警。
4. 循环第二条，一直到最后一个文件打开进行第三条。

**算法1.3 int Lexer::switch\_input()**

作 用：判断文件输入的状态。

参 数：无参数。

返回值：return 0或1。

计算过程：

1. 判断现在的文件名是否为空，如果是，则返回1，表示文件已经完全读取完毕。
2. 如果文件名不为空，则重复Lexer::parseCommandLine()的操作。

**算法1.4 Lexer::Token\_value Lexer::get\_token()**

作 用：读取文件的美容并且进行分析与解析。

参 数：无参数。

返回值：return Token\_value枚举类型中的各种状态值，即return curr\_tok。

计算过程：

（1） 如果curr\_tok值不为END，那么一直读取下去。

（2） 如果遇到了空白字符，那么line值+1，停止本行的读取。

1. 对收到的字符ch进行分类处理，如果收到的是空值0，那么停止输入，返回END；如果ch是‘\n’，那么继续获取ch；如果ch是各种定义过的运算符号，那么重新使用token()函数；如果得到的ch是数值，那么获取并进行处理，并且返回NUMBER；如果以上情况都不是，那么判断是否是无效的字母或者是符号，如果是，返回NAME，否则，直接报错“错误的输入”，返回ERR。

**算法1.5 void Lexer::skip()**

作 用：跳过‘\n’和’;’。

参 数：无参数。

返回值：无返回值。

计算过程：

1. 持续读入字符，遇到换行符或者是分号则直接跳过，进行下一步的读取。

**算法1.6 double Parser::expr()**

作 用：处理加减法。

参 数：bool get

返回值：返回double类型的值left

计算过程：

1. 调用trem()函数，返回double类型的值left。
2. 对修改后的curr\_tok值进行分析，如果是PLUS，则让left自加term()函数的返回值，否则自减，如果不是以上两种情况，那么直接返回值left。

**算法1.7 double Parser::term()**

作 用：处理乘除法。

参 数：bool get

返回值：返回double类型的值left

计算过程：

（1） 调用prim()函数，返回double类型的值left。

（2） 对修改后的curr\_tok值进行分析，如果是MUL，则让left自乘prim()函数的返回值，否则自除，，如果除数是0，那么报错“除数为0！ ”，如果不是以上两种情况，那么直接返回值left。

**算法1.8 double Parser::prim()**

作 用：综合处理得到的字符

参 数：bool get

返回值：返回double类型的值v或e

计算过程：

1. 如果get值为true，那么读取下一位的字符。
2. 分析修改后的curr\_tok值，如果是NUMBER，那么返回number\_value的值给v，并且读取下一位字符；如果是NAME，那么给string\_value的键值对赋值为v并且返回；如果是MINUS，那么直接返回负的prim()函数处理true的值；如果是LP，那么将expr()函数返回的值赋给e，并且判断是否存在RP，如果不存在，那么报错“缺失右括号”；如果是其他情况，那么直接报错“初级错误！ ”，然后如果下一位不是END，则继续下一行的输入。如果为END，那么直接返回设定的标志值9999999999，以便于直接能够发现错误。

**算法1.9 double Error::error()**

作 用：爆出错误的类型和错误行号。

参 数：const std::string & s

返回值：return 1

计算过程：

1. 向屏幕输出错误类型。
2. 向屏幕输出错误行号。

# 5改造实现桌面计算器

## 5.1**计算名字的相关数据**

**测试用例：**

测试用例1:添加命令行参数data.txt，通过读取文件的方式来输入。

文件的内容为：

1+1;2+@;3+#;4\*(1+3)

测试用例2:手动输入。

输入的内容为：

1+1；1+@；3\*（2+1）；1+#3

测试用例3:添加命令行参数test.txt，通过读取文件的方式来输入。

文件的内容为：

1+1;4+$;3\*(2+3)

测试用例4:同时添加命令行参数data.txt和test.txt，通过读取文件的方式来输入。

**输出结果：**

输出结果1:

Result is 2

error: bad token

error line : 1

error: primary expected

error line : 1

Result is 1e+09

error: bad token

error line : 1

error: primary expected

error line : 1

Result is 1e+09

Result is 16

Program ended with exit code: 0

测试结果2:

result is : 2

error: bad token

error line : 2

error: primary expected

error line : 2

result is : 1e+09

result is : 9

result is : error: bad token

error line : 4

error: primary expected

error line : 4

输出结果3:

Result is 2

error: bad token

error line : 1

error: primary expected

error line : 1

Result is 1e+09

Result is 15

输出结果4:Too many arguments !

# 6实验总结

这次实验实际上是上次桌面计算器的改写实现，加强了我对桌面计算器的程序的理解，了解了文法的便利之处。这个实验还有一个目的就是训练我们的模块化编程的能力，在这个实验中我使用了很多的模块，包括Lexer模块、Paeser模块还有error模块，使用模块的好处就是大大增强了文件的可读性，方便使用对象和写者，同时也提高了文件编译的运行速度，也使许多的借口做的简洁明了，使可用的空间和资源得到了有效的利用，这次实验算是初步带领我们进入了面向对象程序设计的领域，深深地感觉到了这个的有趣，期待下一次的实验。