**中国矿业大学计算机学院**

**系统软件开发实践报告**

课程名称 系统软件开发实践

报告时间 2022.3.9

学生姓名 王杰永

学 号 03190886

专 业 计算机科学与技术

任课教师 张博

成绩考核

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 课程教学目标 | | | 占比 | | 得分 |
| 1 | **目标1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用FLEX脚本语言描述单词结构。 | | | 15% | |  |
| 2 | **目标2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用Bison脚本语言描述语法结构。 | | | 15% | |  |
| 3 | **目标3：**针对计算器需求描述，采用Flex/Bison设计实现高级解释器，进行系统设计，形成结构化设计方案。 | | | 30% | |  |
| 4 | **目标4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。 | | | 30% | |  |
| 5 | **目标5：**培养独立解决问题的能力,理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。 | | | 10% | |  |
| 总成绩 | | | | | |  |
| 指导教师 | |  | 评阅日期 | |  | |

目录

[一 综合实验1 1](#_Toc97729640)

[1.1 实验目的 1](#_Toc97729641)

[1.2 实验内容 1](#_Toc97729642)

[1.3 实验步骤 1](#_Toc97729643)

[1.3.1 Windows系统下进行实验 1](#_Toc97729644)

[1.3.2 Linux系统下进行实验 2](#_Toc97729645)

[1.4 抽象语法树的构建 3](#_Toc97729646)

[1.4.1 产生式 3](#_Toc97729647)

[1.4.2 1+2-3\*2/5的抽象语法树 3](#_Toc97729648)

[1.4.3 (1+2)-(2\*6)的抽象语法树 4](#_Toc97729649)

[1.5 实验总结 4](#_Toc97729650)

[1.5.1 你在编程过程中遇到了哪些难题？ 4](#_Toc97729651)

[1.5.2 你的收获有哪些？ 5](#_Toc97729652)

1. 综合实验1
   1. 实验目的

阅读《flex&Bison》第三章。使用 flex 和 Bison 开发一个具有全部功能的计算器，包括如下功能：

1) 支持变量；

2) 实现复制功能；

3) 实现比较表达式（大于小于等）；

4) 实现 if/then/else 和 do/while 流程控制；

5) 用户可以自定义函数；

6) 简单的错误恢复机制。

* 1. 实验内容

1)阅读 flex Python 第三章 P47~60，重点学习抽象语法树。

2)阅读 fb3-1.y、fb3-1.l、fb3-1funcs.c、fb3-1.h。

3)撰写实验报告，结合实验结果，给出移进规约过程，即抽象语法树的构建过程，如(1+2)-(2\*6)、1+2-3\*2/5。

4)提交报告和实验代码。

* 1. 实验步骤
     1. Windows系统下进行实验

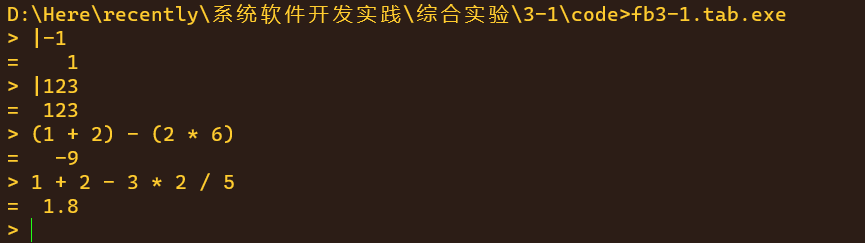
Windows环境下，编译flex与bison文件后，生成了对应的词法/语法分析程序的c语言文件。使用**gcc**编译器编译，得到可执行程序，运行结果顺利，结果如下图所示。

图1-1 windows系统下的运行结果

但是，其他使用cl编译器的同学，编译生成的c文件后，运行可执行文件，不论输入的值是多少，计算器输出值均为0.

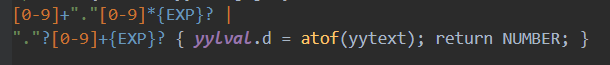
为了弄明白问题所在，我查看flex文件，发现其中匹配数字的模式对应的动作如下图：

图1-2 匹配数字模式的动作

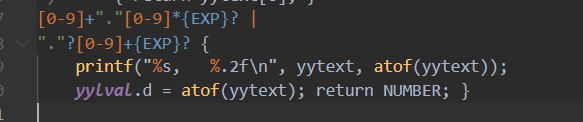
我加入了如下printf语句。目的是为了查看是否成功匹配到数字（输出yytext）、并且是否成功将串转换成浮点数（输出atof的返回值）。

图1-3 加入打印语句

重新编译，运行，发现数字可以成功匹配到，但是atof函数返回值始终是0。于是可以将问题定位至atof函数的使用上。

查询手册得知，atof函数位于头文件<stdlib.h>中，而程序没有引用stdlib库，因此出现了错误。故在flex文件头部加入该头文件引用，发现可以解决问题。而使用gcc编译时，**自动**引入stdlib库，因此不会出现问题。

* + 1. Linux系统下进行实验

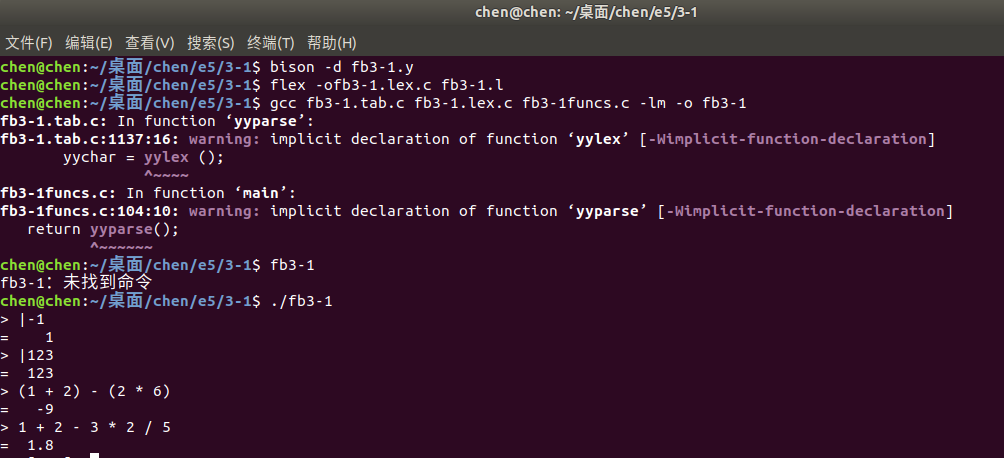
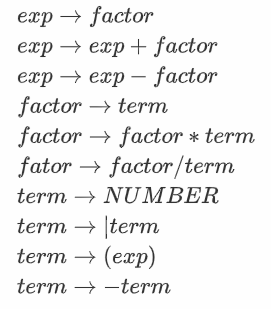
Linux系统下由于不存在cl编译器，默认安装了gcc编译器，因此在编译后的运行阶段没有任何错误。程序结果如下图。

图1-4 Linux系统下的运行结果

* 1. 抽象语法树的构建

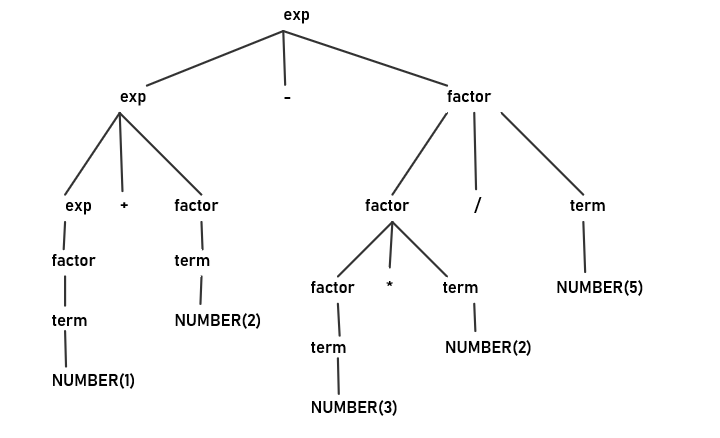
在上次实验中就已经发现，Bison默认采用LALR(1)的语法分析方法。向后多看一个字符，即可以构建整棵抽象语法树。

* + 1. 产生式

查看文件，我们从中可以读出该计算器语法分析所用的产生式。“计算器文法”忽略开始符号calclist与结束符号EOL后的产生式集合主要如下：

基于此，我们便可以构建特定表达式的抽象语法树了，如图1-5、1-6所示

* + 1. 1+2-3\*2/5的抽象语法树

图1-5 抽象语法树

* + 1. (1+2)-(2\*6)的抽象语法树

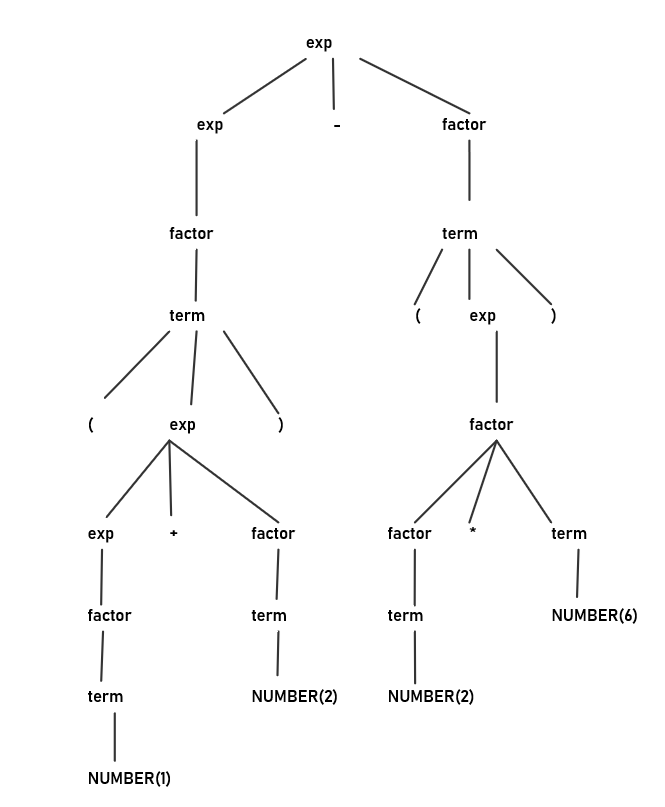


图1-6 抽象语法树

* 1. 实验总结
     1. 你在编程过程中遇到了哪些难题？

正如在1.3.1部分所述，由于在windows系统下也使用gcc编译器进行编译，因此实验完成的很顺利，没有遇到任何问题。

不过在与其他同学交流后，发现使用cl编译器编译后，计算器的输出结果恒为0。由此我也动手尝试使用cl编译器，遇到了同样的问题。解决的步骤已于1.3.1部分体现。

* + 1. 你的收获有哪些？

本次实验的源代码相较于前两周更为复杂，我也得以阅读并分析更复杂的词法、语法规则。对计算机语言编译过程中的前端工作有了更深的理解。

通过解决cl编译器环境下遇到的问题，我也收获了gcc编译器的相关知识。