**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

实验名称 实验四 借助FlexBison进行语法分析

学生姓名 胡钧耀

学 号 06192081

专业班级 计算机科学与技术2019-4班

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**

[实验（四） 借助Flex/Bison进行语法分析 1](#_Toc97395038)

[4.1 实验目的 1](#_Toc97395039)

[4.2 实验内容 1](#_Toc97395040)

[4.3 实验要求 1](#_Toc97395041)

[4.4 移进归约冲突 1](#_Toc97395042)

[4.4.1 移进归约冲突简介 1](#_Toc97395043)

[4.4.2 移进归约冲突实例 1](#_Toc97395044)

[4.5 本次实验出现的移进/规约冲突 2](#_Toc97395045)

[4.6 双系统实验步骤与结果 3](#_Toc97395046)

[4.6.1 Windows 3](#_Toc97395047)

[4.6.2 Linux 6](#_Toc97395048)

[4.7 结果分析 9](#_Toc97395049)

[4.7.1 *test.c*与符号表 9](#_Toc97395050)

[4.7.2 *main.c*与语法分析树 9](#_Toc97395051)

[4.8 实验总结 11](#_Toc97395052)

[4.8.1 gcc编译找不到yyinput，使用了但没有定义 11](#_Toc97395053)

[4.8.2 gcc编译找不到input 11](#_Toc97395054)

[4.8.3 print\_symtab显式声明报警 12](#_Toc97395055)

[4.8.4 Linux语义操作逗号 12](#_Toc97395056)

[4.8.5 Linux中的ULONG\_MAX 12](#_Toc97395057)

[4.8.6 程序评价与收获 12](#_Toc97395058)

1. 借助Flex/Bison进行语法分析
   1. 实验目的

阅读C语言文法的相关参考资料，利用bison实现一个C语言语法分析器。相关概念简介。

* 1. 实验内容

利用语法分析器生成工具bison编写一个C语言的语法分析程序，与词法分析器结合，能够根据语言的上下文无关文法，识别输入的单词序列是否文法的句子。

* 1. 实验要求

阅读flex源文件*input.lex*、bison源文件*cgrammar-new.y*，并参考*实验四 借助FlexBison进行语法分析.pdf*上机调试。

以给定的测试文件*test.c*作为输入，输出运行结果到输出文件*out.txt*中。

* 1. 移进归约冲突
     1. 移进归约冲突简介

移进分析动作表示句柄尚未在分析栈顶行程，正期继续移进符号以形成句柄，规约表明当前分析栈的栈顶已形成当前句型的句柄β，要立即进行规约。当我们预读了词素的时候，既可以对分析栈里面已有的词素进行规约也可以对预读的词素进行移进，这就是移进规约冲突。

* + 1. 移进归约冲突实例

给出如下一段文法规则。

expr: expr – expr | expr \* expr | - expr

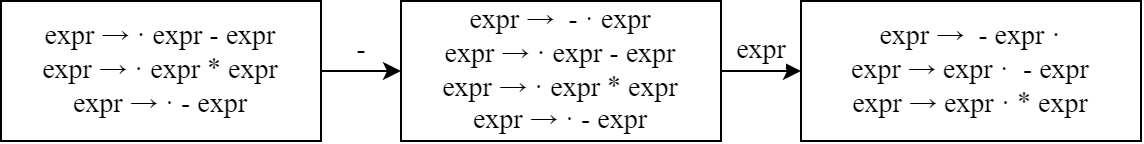


图 1 移进规约冲突示例

以上述规则为例，在进行移进的过程中，在最右侧的状态集合中既有移进项目存在（expr → expr · - expr和expr → expr · \* expr），又有规约项目存在（expr → - expr ·）。通常有两种解决办法。

第一，使用%prec定义规则对应的符号（也就是定义此规则和哪个符号的优先级相同）。

第二，使用%left、%right、%noassoc，和%precedence来定义符号的优先级和结合性（分别是左结合、右结合、没有结合性、未定义的结合性）。

所以需要先定义规则对应的虚拟符号，再定义这个虚拟符号和某个规则的优先级关系，这样就定义了此规则和某个规则的优先级了。

* 1. 本次实验出现的移进/规约冲突

本次实验文法规则有一条如下。

Stmt

: ……

……

| IF '(' Exp ')' Stmt { $$ = …… }

| IF '(' Exp ')' Stmt ELSE Stmt { $$ = …… }

……

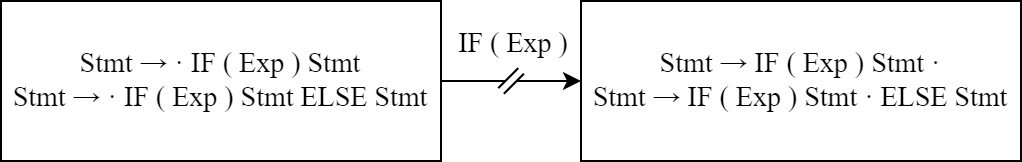


图 2 本实验移进规约冲突

左侧状态集合不断移进，识别IF ( EXP ) Stmt之后，到达右侧状态集合，在最右侧的状态集合中既有移进项目存在（Stmt → IF ( Exp ) Stmt ·），又有规约项目存在（Stmt → IF ( Exp ) Stmt · ELSE Stmt）。于是出现了移进归约冲突。

YACC中解决二义性文法的方法通常是指定优先级，%nonassoc意味着没有依赖关系，经常在连接词中和%prec一起使用，用于指定一个规则的优先级。

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

……

A %prec LOWER\_THAN\_ELSE ……

B ELSE ……

上面的修改方案，使得LOWER\_THAN\_ELSE的优先级小于ElSE，同时语法第一句的优先级被指定为了LOWER\_THAN\_ELSE，这样当冲突发生时，编译器将先移进，后规约。

* 1. 双系统实验步骤与结果
     1. Windows

执行以下命令， 生成*lex.yy.c*、 *cgrammar-new.tab.h*、*cgrammar-new.tab.c*。

flex -l input.lex

bison -d cgrammar-new.y

cgrammar-new.y: conflicts: 1 shift/reduce

执行上述命令后，bison提示有一个移进规约冲突。以bison的-v选项生成状态机描述文件*cgrammar-new.output*， 即执行如下代码。

bison -v cgrammar-new.y

*cgrammar-new.output*文件内容如下。

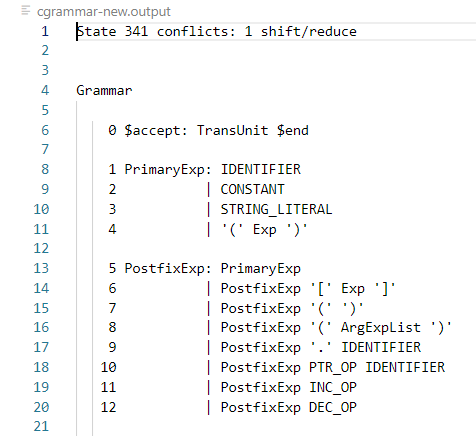
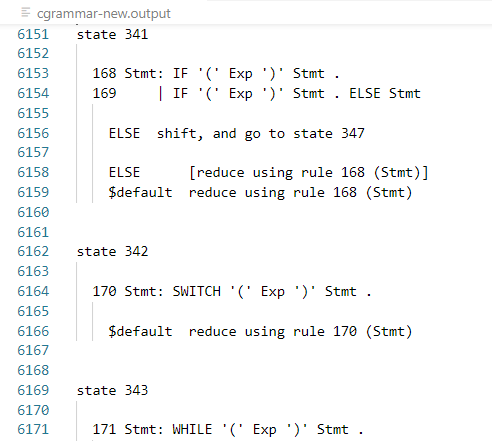
 

图 3 *cgrammar-new.output*文件内容

修改以下两处：第一，在yacc的头部加入如下代码。

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

第二，在 355 行加入如下代码。

%prec LOWER\_THAN\_ELSE

重新编译，可以发现已经消除了移进规约错误。

使用gcc编译器，编译*lex.yy.c*、*cgrammar-new.tab.c*、*main.c、parser.c*。生成可执行文件*2\_2.exe*，并分析源文件*test.c*。

gcc lex.yy.c cgrammar-new.tab.c main.c parser.c -o"2\_2.exe"

2\_2.exe < test.c

运行结果见下页。

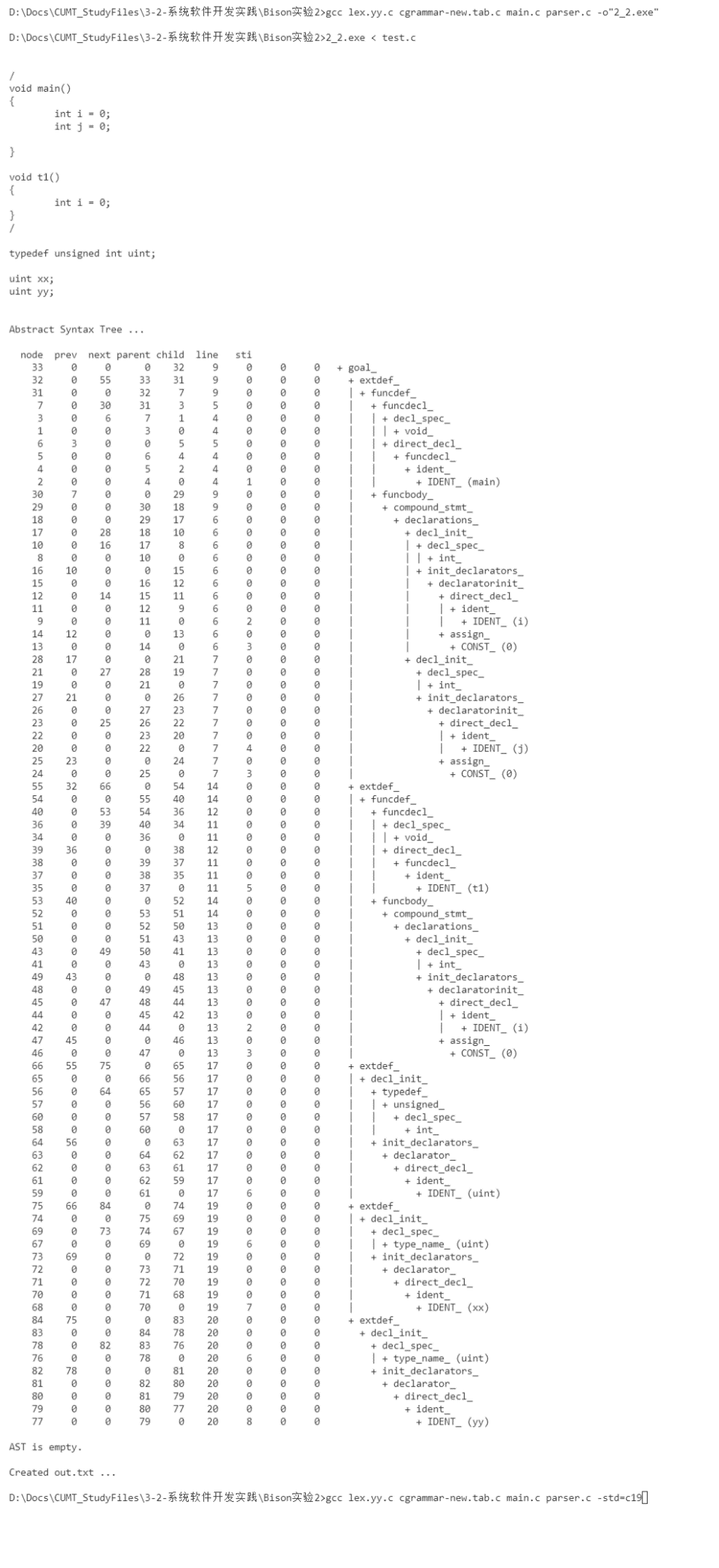


图 4 Windows程序运行结果

输出文件*out.txt*，包含符号表和抽象语法树，内容如下：

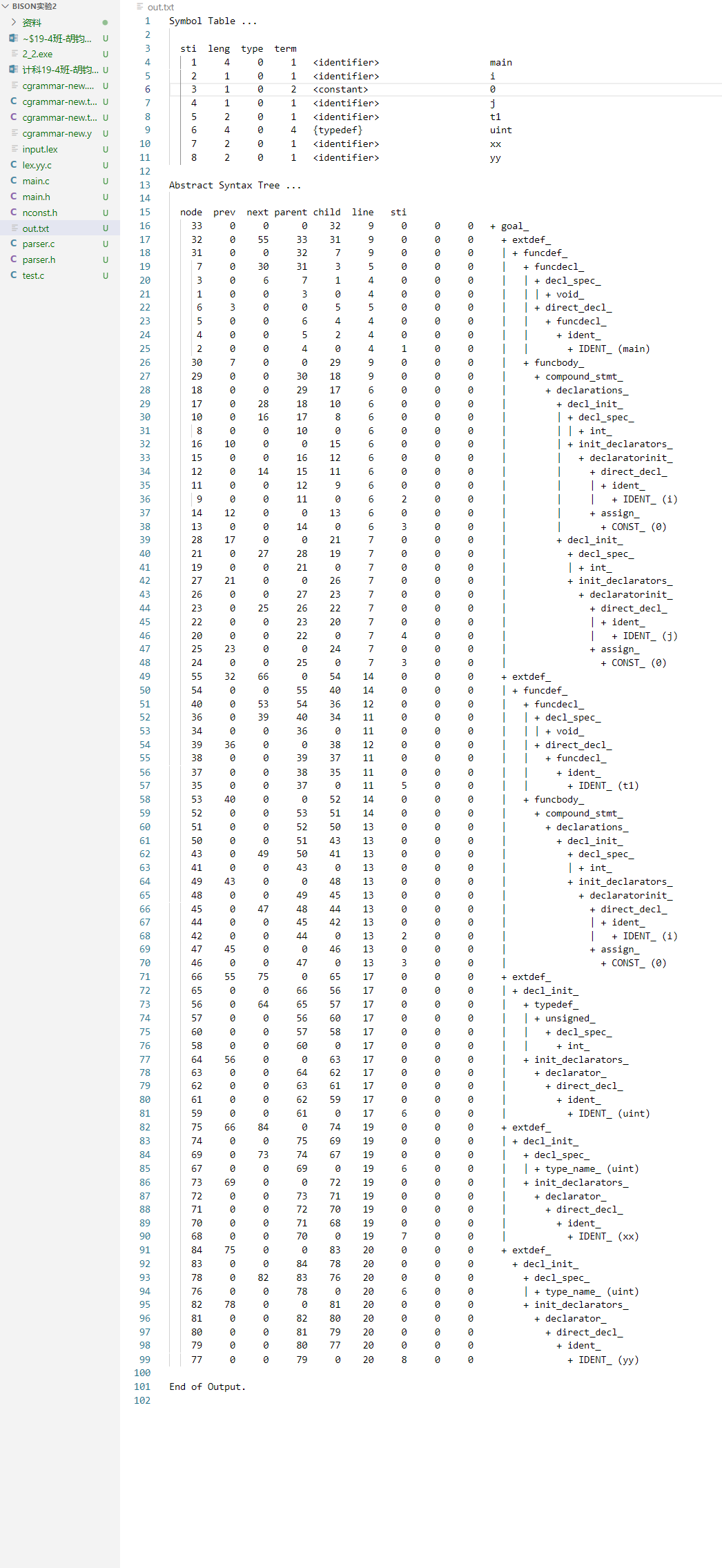


图 5 Windows输出文件*out.txt*结果

* + 1. Linux

考虑到VS Code支持多个操作系统平台，为了尝试体验，这次不直接使用Linux终端，和Windows一样，同样在VS Code中进行编译、修改等操作。

Linux上的*io.h*头文件没有包含在标准库*/usr/include*中，使用如下命令可以查找该文件位置，发现而是在*/usr/include/x86\_64-linux-gnu/sys/*文件夹中。

find /usr/include -name "io.h"

在*input.lex*修改如下头文件。

#include <sys/io.h>

执行以下命令， 生成*lex.yy.c*、 *cgrammar-new.tab.h*、*cgrammar-new.tab.c*。

flex -l input.lex

bison -d cgrammar-new.y

cgrammar-new.y: 警告: 1 项偏移/归约冲突 [-Wconflicts-sr]

执行上述命令后，bison提示有一个移进规约冲突。以bison的-v选项生成状态机描述文件*cgrammar-new.output*， 即执行如下代码。

bison -v cgrammar-new.y

*cgrammar-new.output*文件内容如下。

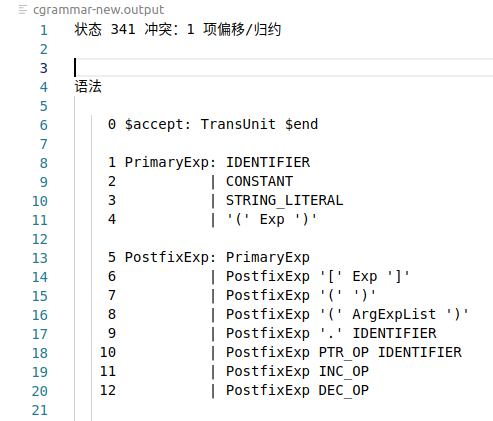
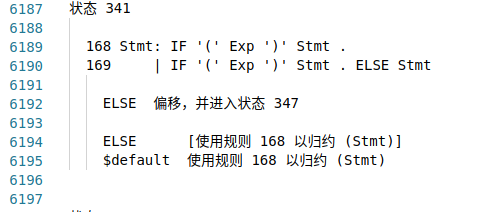
 

图 6 *cgrammar-new.output*文件内容

修改的两段代码和Windows的修改一致。重新编译，可以发现已经消除了移进规约错误。

使用gcc编译器，编译*lex.yy.c*、*cgrammar-new.tab.c*、*main.c、parser.c*。生成可执行文件*c-grammar.out*，并分析源文件*test.c*。

gcc -o”4.out” lex.yy.c cgrammar-new.tab.c main.c parser.c

./4.out < test.c

运行结果见下页。

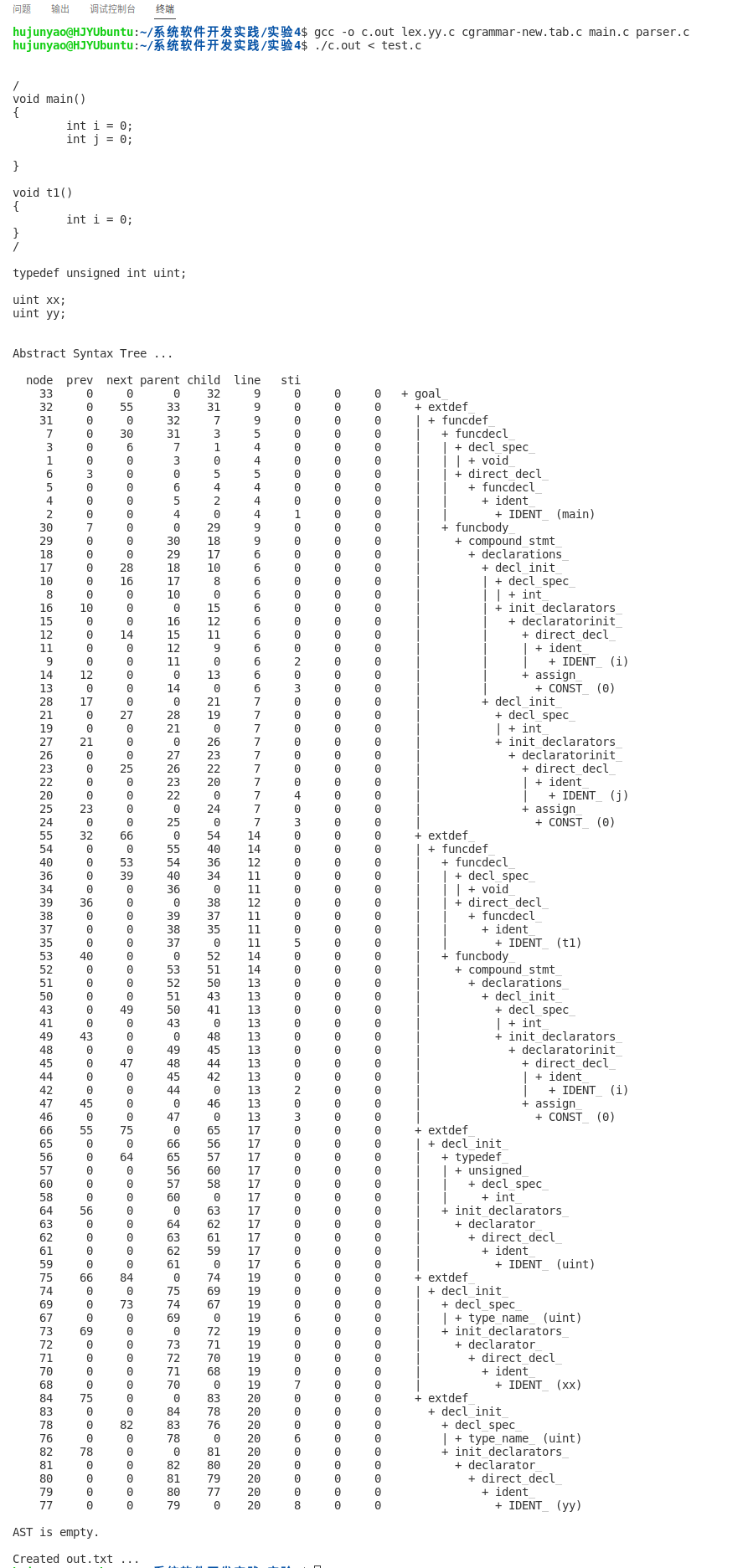


图 7 Linux输出文件*out.txt*结果

输出文件*out.txt*，包含符号表和抽象语法树，内容如下：

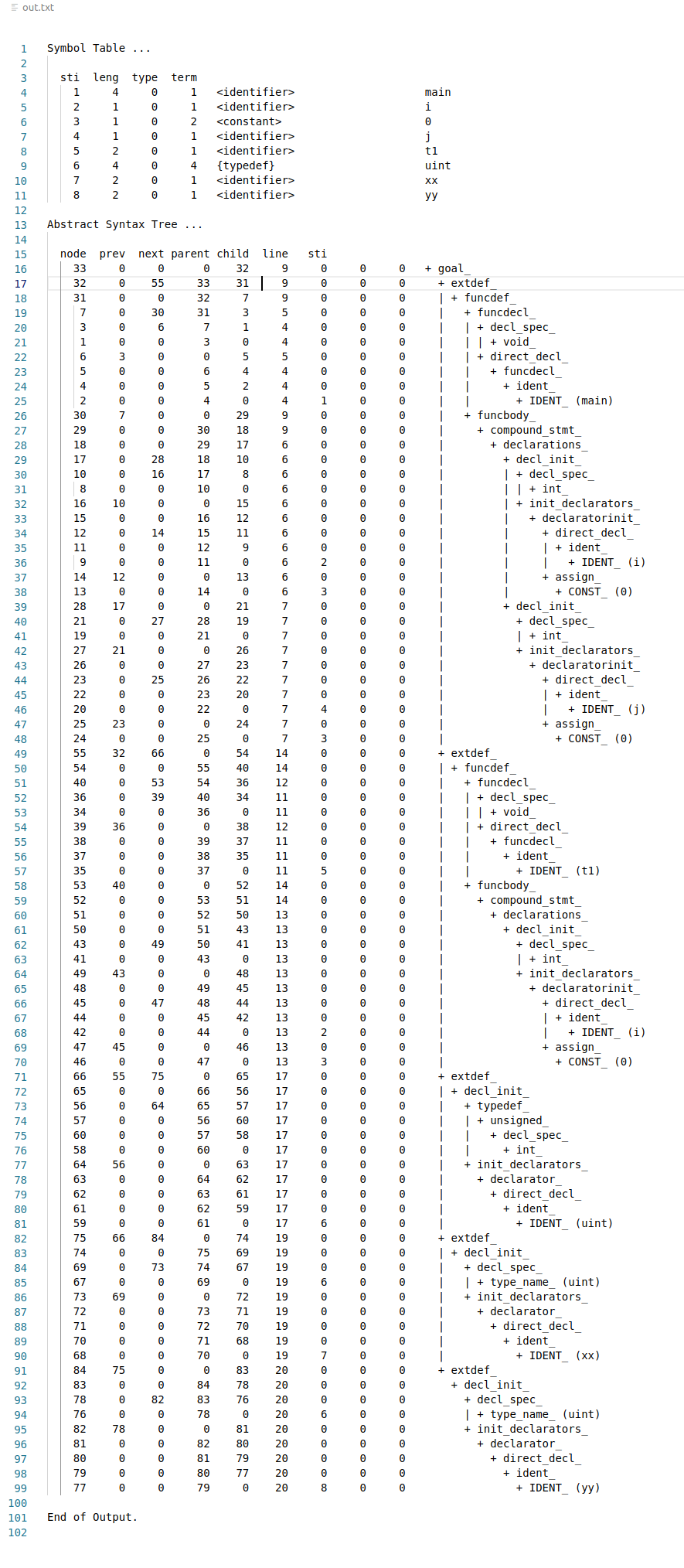


图 8 Linux运行结果

* 1. 结果分析
     1. *test.c*与符号表

从符号表中可以看出，成功分析出了main、i、j、xx、yy、t1等标识符，分析出了uint类型符，常量0。同时也识别出了每一个标识符的长度。其中sti表示符号序号，leng表示符号长度，type表示符号类型，如integer，float等，term表示终结符的类型。

识别主要流程如下所示。在lex中通过识别每一个标识符并return相应的符号，bison识别到之后进行相应的语法规则处理。

例如对于标识符，在*input.lex*中，字母或下划线开头，由字母数字下划线组成的字符串，使用check\_type()函数来判断该字符串的类型。

Node为抽象语法树的节点，是一个结构体，Symbol是符号的结构体，定义在*parser.h*中。

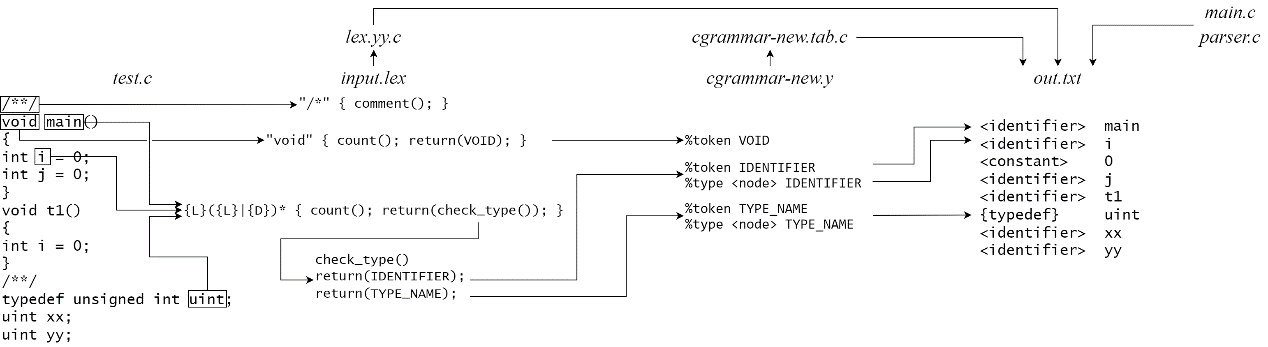


图 9 识别主要步骤

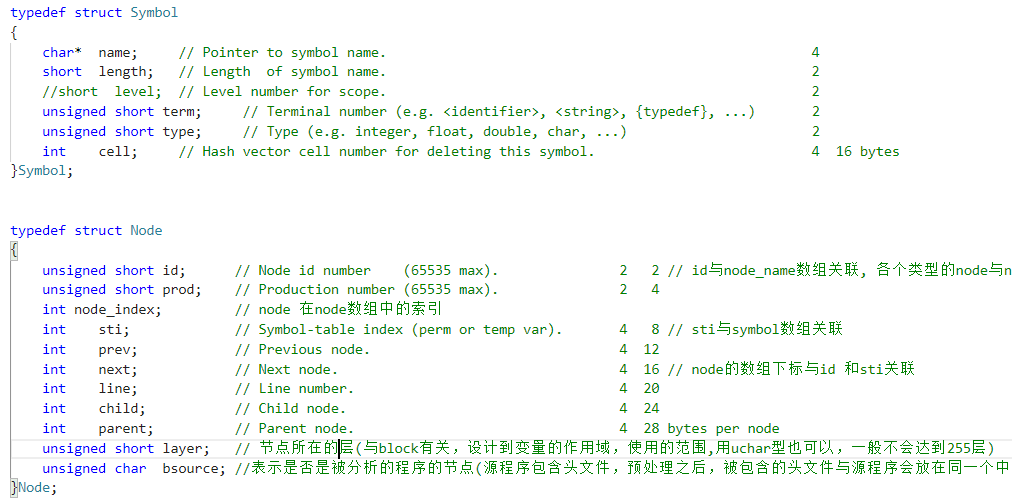


图 10 节点的定义

* + 1. *main.c*与语法分析树

Node表示节点，parent表示父节点，child表示子节点，next表示兄弟节点，line表示行数，sti表示序号，最后的符号串表示节点名字。在*cgrammar-new.y*中，每一个动作都通过调用link函数，来完成树节点的相互链接。

*out.txt*中输出的部分语法分析树的形式是一种子女兄弟链的二叉树输出方式，将其一部分复原为树型，展示如下。

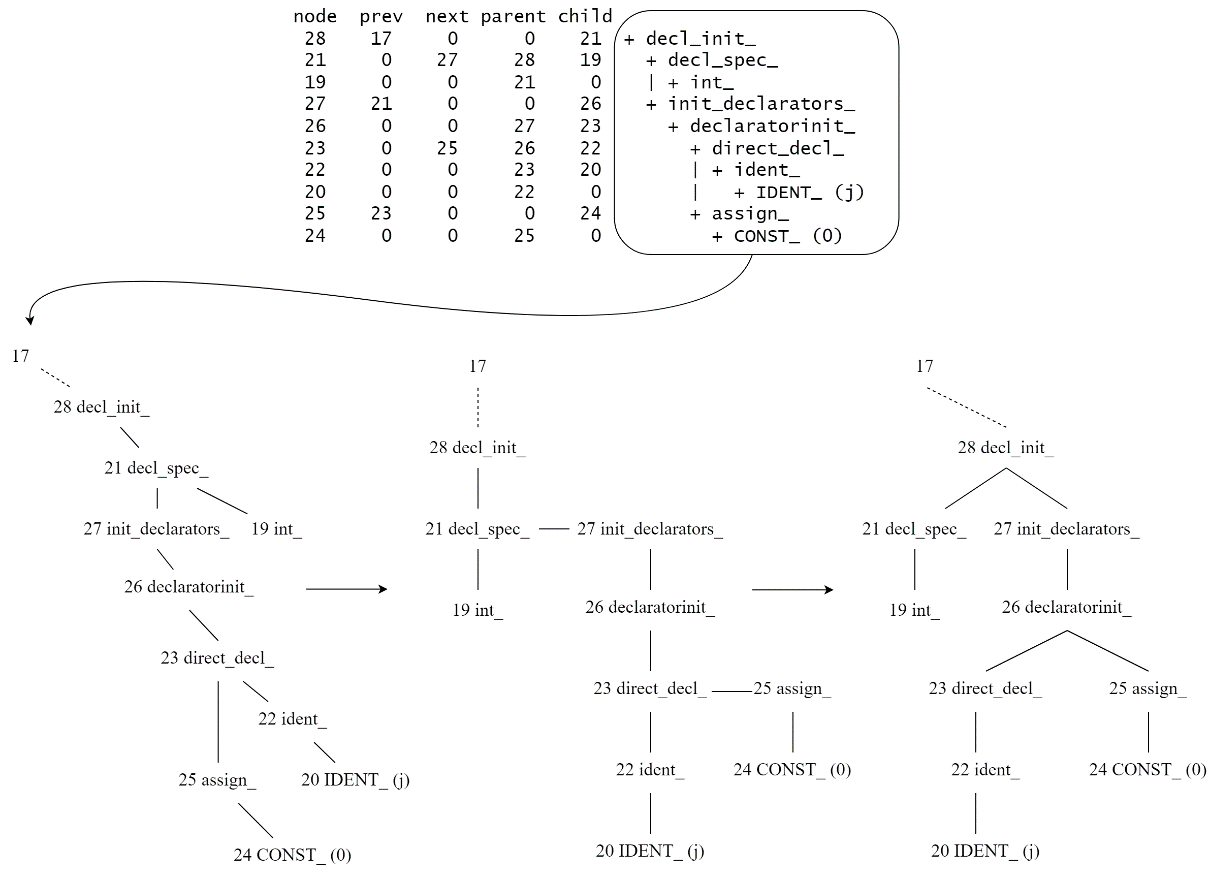


图 11 根据*output.txt*所给子女兄弟链表转换为语法分析树（部分）

在*cgrammar-new.y*的语法规则中，每识别出一个节点，都使用创建新的抽象语法树或使用link函数将解析结果放到对应节点上。

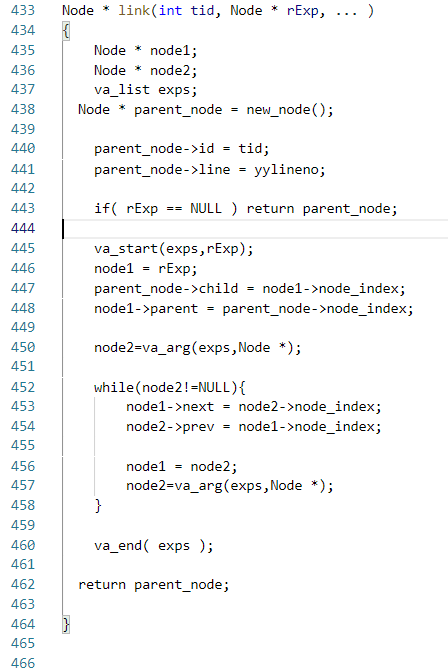


图 12 构造子女兄弟链link函数

* 1. 实验总结
     1. gcc编译找不到yyinput，使用了但没有定义

使用gcc编译后，得到以下错误提示。

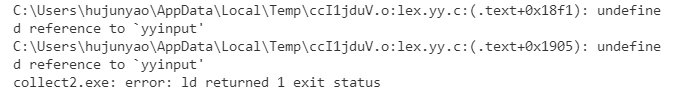


图 13 第一次错误提示

修改*lex.yy.c*，使其能顺利编译。

#ifdef \_\_cplusplus

static int yyinput()

#else

static int input()

#endif

改为如下代码。

static int yyinput()

* + 1. gcc编译找不到input

重新编译，错误提示如下。



图 14 第二次错误提示

修改*lex.yy.c*，使其能顺利编译。

#ifdef \_\_cplusplus

return yyinput();

#else

return input();

#endif

改为如下代码。

return yyinput();

* + 1. print\_symtab显式声明报警

报警提示，根据实验三所学，只需在main.c文件中添加该函数的显式声明，只需要注意该函数的参数类型和返回类型即可。

extern void  print\_symtab (char \*\* );

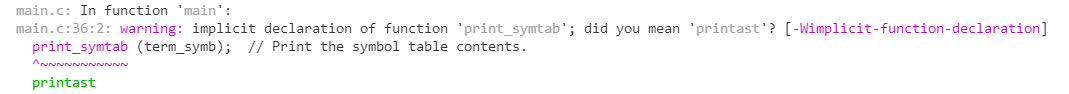


图 15 报警提示

* + 1. Linux语义操作逗号

在*cgrammar-new.y*中，给这一行加上逗号即可。

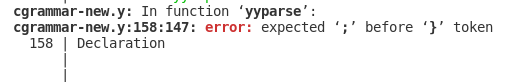


图 16 逗号出错

* + 1. Linux中的ULONG\_MAX

在*parcser.c*中，加上一行引用头文件即可。

#include <limits.h>

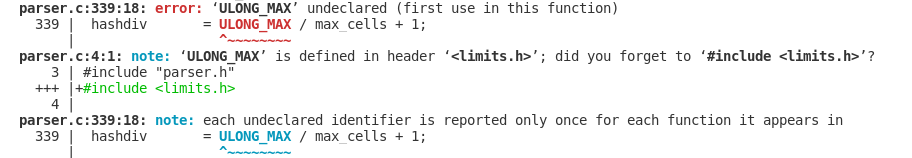


图 17 ULONG\_MAX出错

* + 1. 程序评价与收获

这一次实验进一步熟悉了使用flex和bison进行联合进行语法分析的步骤，对于制作一款编译器有了进一步的感受。通过分析源代码文件，我更好地理解了语法制导翻译地步骤流程以及抽象语法树的生成过程。