

# 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	<b>目标 1：</b> 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	<b>目标 2：</b> 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	<b>目标 3：</b> 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	<b>目标 4：</b> 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	<b>目标 5：</b> 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
4.1 实验目的.....	1
4.2 实验内容.....	1
4.3 实验要求.....	1
4.4 移进归约冲突.....	1
4.4.1 移进归约冲突简介.....	1
4.4.2 移进归约冲突实例.....	1
4.5 本次实验出现的移进/规约冲突.....	2
4.6 双系统实验步骤与结果.....	3
4.6.1 Windows.....	3
4.6.2 Linux.....	6
4.7 结果分析.....	9
4.7.1 <i>test.c</i> 与符号表.....	9
4.7.2 <i>main.c</i> 与语法分析树.....	9
4.8 实验总结.....	11
4.8.1 gcc 编译找不到 <i>yyinput</i> ，使用了但没有定义.....	11
4.8.2 gcc 编译找不到 <i>input</i> .....	11
4.8.3 <i>print_symtab</i> 显式声明报警.....	12
4.8.4 Linux 语义操作逗号.....	12
4.8.5 Linux 中的 <i>ULONG_MAX</i> .....	12
4.8.6 程序评价与收获.....	12

## 实验（四） 借助 Flex/Bison 进行语法分析

### 4.1 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。

### 4.2 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。

### 4.3 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。

### 4.4 移进归约冲突

#### 4.4.1 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  $\beta$ ，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。

#### 4.4.2 移进归约冲突实例

给出如下一段文法规则。

```
expr: expr - expr | expr * expr | - expr
```

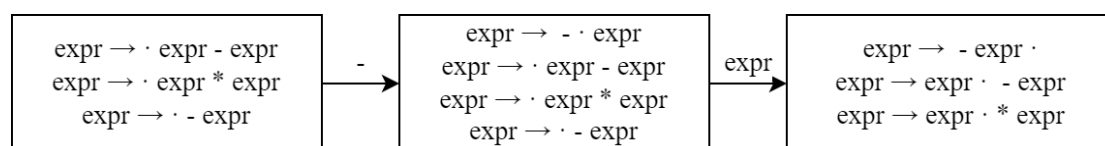


图 1 移进规约冲突示例

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（ $\text{expr} \rightarrow \text{expr} \cdot - \text{expr}$  和  $\text{expr} \rightarrow \text{expr} \cdot * \text{expr}$ ），又有规约项目存在（ $\text{expr} \rightarrow - \text{expr} \cdot$ ）。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。

## 4.5 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 { $$ = ..... }
| IF '(' Exp ')' Stmt ELSE Stmt { $$ = ..... }
.....

```

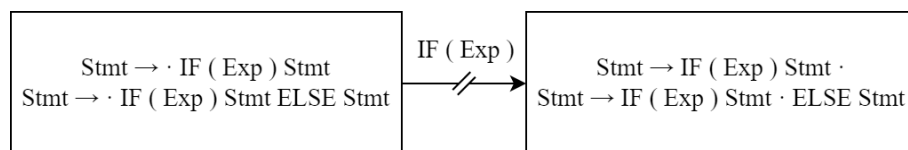


图 2 本实验移进规约冲突

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。

## 4.6 双系统实验步骤与结果

### 4.6.1 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。

```

1  state 341 conflicts: 1 shift/reduce
2
3
4  Grammar
5
6  0 $accept: TransUnit $end
7
8  1 PrimaryExp: IDENTIFIER
9
10 2          | CONSTANT
11 3          | STRING_LITERAL
12 4          | '(' Exp ')'
13
14 5 PostfixExp: PrimaryExp
15 6          | PostfixExp '[' Exp ']'
16 7          | PostfixExp '(' ')'
17 8          | PostfixExp '(' ArgExplist ')'
18 9          | PostfixExp '.' IDENTIFIER
19 10         | PostfixExp PTR_OP IDENTIFIER
20 11         | PostfixExp INC_OP
21 12         | PostfixExp DEC_OP

```

```

6151 state 341
6152
6153 168 Stmt: IF '(' Exp ')' Stmt .
6154 169       | IF '(' Exp ')' Stmt . ELSE Stmt
6155
6156       ELSE shift, and go to state 347
6157
6158       ELSE [reduce using rule 168 (Stmt)]
6159 $default reduce using rule 168 (Stmt)
6160
6161
6162 state 342
6163
6164 170 Stmt: SWITCH '(' Exp ')' Stmt .
6165
6166 $default reduce using rule 170 (Stmt)
6167
6168
6169 state 343
6170
6171 171 Stmt: WHILE '(' Exp ')' Stmt .

```

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。

## 《系统软件开发实践》实验报告

```
D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验2>gcc lex.yy.c cgrammar-new.tab.c main.c parser.c -o"2_2.exe"
D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验2>2_2.exe < test.c

/
void main()
{
    int i = 0;
    int j = 0;
}

void t1()
{
    int i = 0;
}
/

typedef unsigned int uint;

uint xx;
uint yy;

Abstract Syntax Tree ...

node  prev  next  parent  child  line  sti
33    0     0     0     32    9     0     0     0 + goal_
32    0    55    33    31    9     0     0     0 + extdef_
31    0     0     32    7     9     0     0     0 + funcdef_
7     0    30    31    3     5     0     0     0 | + funcdecl_
3     0     6     7     1     4     0     0     0 | | + decl_spec_
1     0     0     3     0     4     0     0     0 | | + void_
6     3     0     0     5     5     0     0     0 | | + direct_decl_
5     0     0     6     4     4     0     0     0 | | + funcdecl_
4     0     0     5     2     4     0     0     0 | | + ident_
2     0     0     4     0     4     1     0     0 | | + IDENT_ (main)
30    7     0     0    29    9     0     0     0 | + funcbody_
29    0     0     30    18    9     0     0     0 | + compound_stmt_
18    0     0     29    17    6     0     0     0 | + declarations_
17    0    28    18    10    6     0     0     0 | | + decl_init_
10    0    16    17     8     6     0     0     0 | | | + decl_spec_
8     0     0    10     0     6     0     0     0 | | | + int_
16    10    0     0    15    6     0     0     0 | | | + init_declarators_
15    0     0    16    12    6     0     0     0 | | | + declaratorinit_
12    0    14    15    11    6     0     0     0 | | | + direct_decl_
11    0     0    12     9     6     0     0     0 | | | | + ident_
9     0     0    11     0     6     2     0     0 | | | | + IDENT_ (i)
14    12    0     0    13     6     0     0     0 | | | + assign_
13    0     0    14     0     6     3     0     0 | | | + CONST_ (0)
28    17    0     0    21     7     0     0     0 | | + decl_init_
21    0    27    28    19     7     0     0     0 | | + decl_spec_
19    0     0    21     0     7     0     0     0 | | | + int_
27    21    0     0    26     7     0     0     0 | | + init_declarators_
26    0     0    27    23     7     0     0     0 | | + declaratorinit_
23    0    25    26    22     7     0     0     0 | | | + direct_decl_
22    0     0    23    20     7     0     0     0 | | | | + ident_
20    0     0    22     0     7     4     0     0 | | | | + IDENT_ (j)
25    23    0     0    24     7     0     0     0 | | | + assign_
24    0     0    25     0     7     3     0     0 | | | + CONST_ (0)
55    32    66     0    54    14     0     0     0 | + extdef_
54    0     0    55    40    14     0     0     0 | | + funcdef_
40    0    53    54    36    12     0     0     0 | | | + funcdecl_
36    0    39    40    34    11     0     0     0 | | | | + decl_spec_
34    0     0    36     0    11     0     0     0 | | | | + void_
39    36    0     0    38    12     0     0     0 | | | + direct_decl_
38    0     0    39    37    11     0     0     0 | | | + funcdecl_
37    0     0    38    35    11     0     0     0 | | | | + ident_
35    0     0    37     0    11     5     0     0 | | | | + IDENT_ (t1)
53    40    0     0    52    14     0     0     0 | | + funcbody_
52    0     0    53    51    14     0     0     0 | | + compound_stmt_
51    0     0    52    50    13     0     0     0 | | + declarations_
50    0     0    51    43    13     0     0     0 | | | + decl_init_
43    0    49    50    41    13     0     0     0 | | | + decl_spec_
41    0     0    43     0    13     0     0     0 | | | | + int_
49    43    0     0    48    13     0     0     0 | | | + init_declarators_
48    0     0    49    45    13     0     0     0 | | | + declaratorinit_
45    0    47    48    44    13     0     0     0 | | | + direct_decl_
44    0     0    45    42    13     0     0     0 | | | | + ident_
42    0     0    44     0    13     2     0     0 | | | | + IDENT_ (i)
47    45    0     0    46    13     0     0     0 | | | + assign_
46    0     0    47     0    13     3     0     0 | | | + CONST_ (0)
66    55    75     0    65    17     0     0     0 | + extdef_
65    0     0    66    56    17     0     0     0 | | + decl_init_
56    0    64    65    57    17     0     0     0 | | + typedef_
57    0     0    56    60    17     0     0     0 | | | + unsigned_
60    0     0    57    58    17     0     0     0 | | | + decl_spec_
58    0     0    60     0    17     0     0     0 | | | | + int_
64    56    0     0    63    17     0     0     0 | | | + init_declarators_
63    0     0    64    62    17     0     0     0 | | | + declarator_
62    0     0    63    61    17     0     0     0 | | | + direct_decl_
61    0     0    62    59    17     0     0     0 | | | | + ident_
59    0     0    61     0    17     6     0     0 | | | | + IDENT_ (uint)
75    66    84     0    74    19     0     0     0 | + extdef_
74    0     0    75    69    19     0     0     0 | | + decl_init_
69    0    73    74    67    19     0     0     0 | | | + decl_spec_
67    0     0    69     0    19     6     0     0 | | | | + type_name_ (uint)
73    69    0     0    72    19     0     0     0 | | | + init_declarators_
72    0     0    73    71    19     0     0     0 | | | + declarator_
71    0     0    72    70    19     0     0     0 | | | + direct_decl_
70    0     0    71    68    19     0     0     0 | | | | + ident_
68    0     0    70     0    19     7     0     0 | | | | + IDENT_ (xx)
84    75    0     0    83    20     0     0     0 | + extdef_
83    0     0    84    78    20     0     0     0 | | + decl_init_
78    0    82    83    76    20     0     0     0 | | | + decl_spec_
76    0     0    78     0    20     6     0     0 | | | | + type_name_ (uint)
82    78    0     0    81    20     0     0     0 | | | + init_declarators_
81    0     0    82    80    20     0     0     0 | | | + declarator_
80    0     0    81    79    20     0     0     0 | | | + direct_decl_
79    0     0    80    77    20     0     0     0 | | | | + ident_
77    0     0    79     0    20     8     0     0 | | | | + IDENT_ (yy)

AST is empty.
Created out.txt ...
```

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:

```

1 Symbol Table ...
2
3
4 sti leng type term
5 1 4 0 1 <identifier> main
6 2 1 0 1 <identifier> i
7 3 1 0 2 <constant> 0
8 4 1 0 1 <identifier> j
9 5 2 0 1 <identifier> t1
10 6 4 0 4 {typedef} uint
11 7 2 0 1 <identifier> xx
12 8 2 0 1 <identifier> yy
13
14 Abstract Syntax Tree ...
15
16 node prev next parent child line sti
17 33 0 0 0 32 9 0 0 0 + goal_
18 32 0 55 33 31 9 0 0 0 + extdef_
19 31 0 0 32 7 9 0 0 0 + funcdef_
20 7 0 30 31 3 5 0 0 0 + funcdecl_
21 3 0 6 7 1 4 0 0 0 | + decl_spec_
22 1 0 0 3 0 4 0 0 0 | | + void_
23 6 3 0 0 5 5 0 0 0 | + direct_decl_
24 5 0 0 6 4 4 0 0 0 | + funcdecl_
25 4 0 0 5 2 4 0 0 0 | + ident_
26 2 0 0 4 0 4 1 0 0 | + IDENT_ (main)
27 30 7 0 0 29 9 0 0 0 + funcbody_
28 29 0 0 30 18 9 0 0 0 + compound_stmt_
29 18 0 0 29 17 6 0 0 0 + declarations_
30 17 0 28 18 10 6 0 0 0 + decl_init_
31 10 0 16 17 8 6 0 0 0 | + decl_spec_
32 8 0 0 10 0 6 0 0 0 | | + int_
33 16 10 0 0 15 6 0 0 0 | + init_declarators_
34 15 0 0 16 12 6 0 0 0 | + declaratorinit_
35 12 0 14 15 11 6 0 0 0 | + direct_decl_
36 11 0 0 12 9 6 0 0 0 | | + ident_
37 9 0 0 11 0 6 2 0 0 | | + IDENT_ (i)
38 14 12 0 0 13 6 0 0 0 | | + assign_
39 13 0 0 14 0 6 3 0 0 | | + CONST_ (0)
40 28 17 0 0 21 7 0 0 0 + decl_init_
41 21 0 27 28 19 7 0 0 0 + decl_spec_
42 19 0 0 21 0 7 0 0 0 | + int_
43 27 21 0 0 26 7 0 0 0 + init_declarators_
44 26 0 0 27 23 7 0 0 0 + declaratorinit_
45 23 0 25 26 22 7 0 0 0 + direct_decl_
46 22 0 0 23 20 7 0 0 0 | + ident_
47 20 0 0 22 0 7 4 0 0 | | + IDENT_ (j)
48 25 23 0 0 24 7 0 0 0 | + assign_
49 24 0 0 25 0 7 3 0 0 | | + CONST_ (0)
50 55 32 66 0 54 14 0 0 0 + extdef_
51 54 0 0 55 40 14 0 0 0 + funcdef_
52 40 0 53 54 36 12 0 0 0 + funcdecl_
53 36 0 39 40 34 11 0 0 0 | + decl_spec_
54 34 0 0 36 0 11 0 0 0 | | + void_
55 39 36 0 0 38 12 0 0 0 | + direct_decl_
56 38 0 0 39 37 11 0 0 0 + funcdecl_
57 37 0 0 38 35 11 0 0 0 | + ident_
58 35 0 0 37 0 11 5 0 0 | + IDENT_ (t1)
59 53 40 0 0 52 14 0 0 0 + funcbody_
60 52 0 0 53 51 14 0 0 0 + compound_stmt_
61 51 0 0 52 50 13 0 0 0 + declarations_
62 50 0 0 51 43 13 0 0 0 + decl_init_
63 43 0 49 50 41 13 0 0 0 + decl_spec_
64 41 0 0 43 0 13 0 0 0 | + int_
65 49 43 0 0 48 13 0 0 0 + init_declarators_
66 48 0 0 49 45 13 0 0 0 + declaratorinit_
67 45 0 47 48 44 13 0 0 0 + direct_decl_
68 44 0 0 45 42 13 0 0 0 | + ident_
69 42 0 0 44 0 13 2 0 0 | | + IDENT_ (i)
70 47 45 0 0 46 13 0 0 0 + assign_
71 46 0 0 47 0 13 3 0 0 | | + CONST_ (0)
72 66 55 75 0 65 17 0 0 0 + extdef_
73 65 0 0 66 56 17 0 0 0 + decl_init_
74 64 0 64 65 57 17 0 0 0 + typedef_
75 57 0 0 56 60 17 0 0 0 | + unsigned_
76 60 0 0 57 58 17 0 0 0 | + decl_spec_
77 58 0 0 60 0 17 0 0 0 | | + int_
78 64 56 0 0 63 17 0 0 0 + init_declarators_
79 63 0 0 64 62 17 0 0 0 + declarator_
80 62 0 0 63 61 17 0 0 0 + direct_decl_
81 61 0 0 62 59 17 0 0 0 | + ident_
82 59 0 0 61 0 17 6 0 0 | + IDENT_ (uint)
83 75 66 84 0 74 19 0 0 0 + extdef_
84 74 0 0 75 69 19 0 0 0 + decl_init_
85 69 0 73 74 67 19 0 0 0 + decl_spec_
86 73 0 0 69 0 19 6 0 0 | + type_name_ (uint)
87 67 69 0 0 72 19 0 0 0 + init_declarators_
88 72 0 0 73 71 19 0 0 0 + declarator_
89 71 0 0 72 70 19 0 0 0 + direct_decl_
90 70 0 0 71 68 19 0 0 0 | + ident_
91 68 0 0 70 0 19 7 0 0 | + IDENT_ (xx)
92 84 75 0 0 83 20 0 0 0 + extdef_
93 83 0 0 84 78 20 0 0 0 + decl_init_
94 78 0 82 83 76 20 0 0 0 + decl_spec_
95 76 0 0 78 0 20 6 0 0 | + type_name_ (uint)
96 82 78 0 0 81 20 0 0 0 + init_declarators_
97 81 0 0 82 80 20 0 0 0 + declarator_
98 80 0 0 81 79 20 0 0 0 + direct_decl_
99 79 0 0 80 77 20 0 0 0 | + ident_
100 77 0 0 79 0 20 8 0 0 | + IDENT_ (yy)
101 End of Output.
102

```

图 5 Windows 输出文件 *out.txt* 结果



## 4.6.2 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。

```

6187 状态 341
6188
6189 168 Stmt: IF '(' Exp ')' Stmt .
6190      | IF '(' Exp ')' Stmt . ELSE Stmt
6191
6192      ELSE  偏移, 并进入状态 347
6193
6194      ELSE      [使用规则 168 以归约 (Stmt)]
6195      $default  使用规则 168 以归约 (Stmt)
6196
6197

```

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。

```
问题 输出 调试控制台 终端
hujunyao@HJYUbuntu:~/系统软件开发实践/实验4$ gcc -o c.out lex.yy.c cgrammar-new.tab.c main.c parser.c
hujunyao@HJYUbuntu:~/系统软件开发实践/实验4$ ./c.out < test.c

/
void main()
{
    int i = 0;
    int j = 0;

}

void t1()
{
    int i = 0;
}
/

typedef unsigned int uint;

uint xx;
uint yy;

Abstract Syntax Tree ...

node  prev  next  parent  child  line  sti
33      0      0      0      32      9      0
32      0     55     33     31      9      0
31      0      0     32      7      9      0
7       0     30     31      3      5      0
3       0      6      7      1      4      0
1       0      0      3      0      4      0
6       3      0      0      5      5      0
5       0      0      6      4      4      0
4       0      0      5      2      4      0
2       0      0      4      0      4      1
30      7      0      0     29      9      0
29      0      0     30     18      9      0
18      0      0     29     17      6      0
17      0     28     18     10      6      0
10      0     16     17      8      6      0
8       0      0     10      6      6      0
16     10      0      0     15      6      0
15      0      0     16     12      6      0
12      0     14     15     11      6      0
11      0      0     12      9      6      0
9       0      0     11      0      6      2
14     12      0      0     13      6      0
13      0      0     14      0      6      3
28     17      0      0     21      7      0
21      0     27     28     19      7      0
19      0      0     21      0      7      0
27     21      0      0     26      7      0
26      0      0     27     23      7      0
23      0     25     26     22      7      0
22      0      0     23     20      7      0
20      0      0     22      0      7      4
25     23      0      0     24      7      0
24      0      0     25      0      7      3
55     32     66      0     54     14      0
54      0      0     55     40     14      0
40      0     53     54     36     12      0
36      0     39     40     34     11      0
34      0      0     36      0     11      0
39     36      0      0     38     12      0
38      0      0     39     37     11      0
37      0      0     38     35     11      0
35      0      0     37      0     11      5
53     40      0      0     52     14      0
52      0      0     53     51     14      0
51      0      0     52     50     13      0
50      0      0     51     43     13      0
43      0     49     50     41     13      0
41      0      0     43      0     13      0
49     43      0      0     48     13      0
48      0      0     49     45     13      0
45      0     47     48     44     13      0
44      0      0     45     42     13      0
42      0      0     44      0     13      2
47     45      0      0     46     13      0
46      0      0     47      0     13      3
66     55     75      0     65     17      0
65      0      0     66     56     17      0
56      0     64     65     57     17      0
57      0      0     56     60     17      0
60      0      0     57     58     17      0
58      0      0     60      0     17      0
64     56      0      0     63     17      0
63      0      0     64     62     17      0
62      0      0     63     61     17      0
61      0      0     62     59     17      0
59      0      0     61      0     17      6
75     66     84      0     74     19      0
74      0      0     75     69     19      0
69      0     73     74     67     19      0
67      0      0     69      0     19      6
73     69      0      0     72     19      0
72      0      0     73     71     19      0
71      0      0     72     70     19      0
70      0      0     71     68     19      0
68      0      0     70      0     19      7
84     75      0      0     83     20      0
83      0      0     84     78     20      0
78      0     82     83     76     20      0
76      0      0     78      0     20      6
82     78      0      0     81     20      0
81      0      0     82     80     20      0
80      0      0     81     79     20      0
79      0      0     80     77     20      0
77      0      0     79      0     20      8

+ goal
+ extdef
+ funcdecl
+ funcdecl
+ decl_spec
+ void
+ direct decl
+ funcdecl
+ ident
+ IDENT_ (main)
+ funcbody
+ compound stmt
+ declarations
+ decl init
+ decl_spec
+ int
+ init declarators
+ declaratorinit
+ direct decl
+ ident
+ IDENT_ (i)
+ assign
+ CONST_ (0)
+ decl init
+ decl_spec
+ int
+ init declarators
+ declaratorinit
+ direct decl
+ ident
+ IDENT_ (j)
+ assign
+ CONST_ (0)
+ extdef
+ funcdecl
+ funcdecl
+ decl_spec
+ void
+ direct decl
+ funcdecl
+ ident
+ IDENT_ (t1)
+ funcbody
+ compound stmt
+ declarations
+ decl init
+ decl_spec
+ int
+ init declarators
+ declaratorinit
+ direct decl
+ ident
+ IDENT_ (i)
+ assign
+ CONST_ (0)
+ extdef
+ decl init
+ typedef
+ unsigned
+ decl_spec
+ int
+ init declarators
+ declarator
+ direct decl
+ ident
+ IDENT_ (uint)
+ extdef
+ decl init
+ decl_spec
+ type name (uint)
+ init declarators
+ declarator
+ direct decl
+ ident
+ IDENT_ (xx)
+ extdef
+ decl init
+ decl_spec
+ type name (uint)
+ init declarators
+ declarator
+ direct decl
+ ident
+ IDENT_ (yy)

AST is empty.
Created out.txt ...
```

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：

out.txt

```

1 Symbol Table ...
2
3 sti leng type term
4 1 4 0 1 <identifier> main
5 2 1 0 1 <identifier> i
6 3 1 0 2 <constant> 0
7 4 1 0 1 <identifier> j
8 5 2 0 1 <identifier> t1
9 6 4 0 4 (typedef) uint
10 7 2 0 1 <identifier> xx
11 8 2 0 1 <identifier> yy
12
13 Abstract Syntax Tree ...
14
15 node prev next parent child line sti
16 33 0 0 0 32 9 0 0 0 + goal
17 32 0 55 33 31 9 0 0 0 + extdef
18 31 0 0 32 7 9 0 0 0 + funcdef
19 7 0 30 31 3 5 0 0 0 + funcdecl
20 3 0 6 7 1 4 0 0 0 | + decl_spec_
21 1 0 0 3 0 4 0 0 0 | + void
22 6 3 0 0 5 5 0 0 0 | + direct_decl_
23 5 0 0 6 4 4 0 0 0 | + funcdecl
24 4 0 0 5 2 4 0 0 0 | + ident_
25 2 0 0 4 0 4 1 0 0 | + IDENT_ (main)
26 30 7 0 0 29 9 0 0 0 + funcbody
27 29 0 0 30 18 9 0 0 0 + compound_stmt
28 18 0 0 29 17 6 0 0 0 + declarations_
29 17 0 28 18 10 6 0 0 0 + decl_init
30 10 0 16 17 8 6 0 0 0 | + decl_spec_
31 8 0 0 10 0 6 0 0 0 | + int_
32 16 10 0 0 15 6 0 0 0 | + init_declarators_
33 15 0 0 16 12 6 0 0 0 | + declaratorinit_
34 12 0 14 15 11 6 0 0 0 | + direct_decl_
35 11 0 0 12 9 6 0 0 0 | + ident_
36 9 0 0 11 0 6 2 0 0 | + IDENT_ (i)
37 14 12 0 0 13 6 0 0 0 | + assign_
38 13 0 0 14 0 6 3 0 0 | + CONST_ (0)
39 28 17 0 0 21 7 0 0 0 + decl_init
40 21 0 27 28 19 7 0 0 0 + decl_spec_
41 19 0 0 21 0 7 0 0 0 | + int_
42 27 21 0 0 26 7 0 0 0 | + init_declarators_
43 26 0 0 27 23 7 0 0 0 | + declaratorinit_
44 23 0 25 26 22 7 0 0 0 | + direct_decl_
45 22 0 0 23 20 7 0 0 0 | + ident_
46 20 0 0 22 0 7 4 0 0 | + IDENT_ (j)
47 25 23 0 0 24 7 0 0 0 | + assign_
48 24 0 0 25 0 7 3 0 0 | + CONST_ (0)
49 55 32 66 0 54 14 0 0 0 + extdef
50 54 0 0 55 40 14 0 0 0 + funcdef
51 40 0 53 54 36 12 0 0 0 + funcdecl
52 36 0 39 40 34 11 0 0 0 | + decl_spec_
53 34 0 0 36 0 11 0 0 0 | + void
54 39 36 0 0 38 12 0 0 0 | + direct_decl_
55 38 0 0 39 37 11 0 0 0 | + funcdecl
56 37 0 0 38 35 11 0 0 0 | + ident_
57 35 0 0 37 0 11 5 0 0 | + IDENT_ (t1)
58 53 40 0 0 52 14 0 0 0 + funcbody
59 52 0 0 53 51 14 0 0 0 + compound_stmt
60 51 0 0 52 50 13 0 0 0 + declarations_
61 50 0 0 51 43 13 0 0 0 + decl_init
62 43 0 49 50 41 13 0 0 0 | + decl_spec_
63 41 0 0 43 0 13 0 0 0 | + int_
64 49 43 0 0 48 13 0 0 0 | + init_declarators_
65 48 0 0 49 45 13 0 0 0 | + declaratorinit_
66 45 0 47 48 44 13 0 0 0 | + direct_decl_
67 44 0 0 45 42 13 0 0 0 | + ident_
68 42 0 0 44 0 13 2 0 0 | + IDENT_ (i)
69 47 45 0 0 46 13 0 0 0 | + assign_
70 46 0 0 47 0 13 3 0 0 | + CONST_ (0)
71 66 55 75 0 65 17 0 0 0 + extdef
72 65 0 0 66 56 17 0 0 0 + decl_init
73 56 0 64 65 57 17 0 0 0 + typedef
74 57 0 0 56 60 17 0 0 0 | + unsigned_
75 60 0 0 57 58 17 0 0 0 | + decl_spec_
76 58 0 0 60 0 17 0 0 0 | + int_
77 64 56 0 0 63 17 0 0 0 + init_declarators_
78 63 0 0 64 62 17 0 0 0 + declarator
79 62 0 0 63 61 17 0 0 0 + direct_decl_
80 61 0 0 62 59 17 0 0 0 + ident_
81 59 0 0 61 0 17 6 0 0 + IDENT_ (uint)
82 75 66 84 0 74 19 0 0 0 + extdef
83 74 0 0 75 69 19 0 0 0 + decl_init
84 69 0 73 74 67 19 0 0 0 + decl_spec_
85 67 0 0 69 0 19 6 0 0 | + type_name_ (uint)
86 73 69 0 0 72 19 0 0 0 | + init_declarators_
87 72 0 0 73 71 19 0 0 0 + declarator
88 71 0 0 72 70 19 0 0 0 + direct_decl_
89 70 0 0 71 68 19 0 0 0 + ident_
90 68 0 0 70 0 19 7 0 0 + IDENT_ (xx)
91 84 75 0 0 83 20 0 0 0 + extdef
92 83 0 0 84 78 20 0 0 0 + decl_init
93 78 0 82 83 76 20 0 0 0 + decl_spec_
94 76 0 0 78 0 20 6 0 0 | + type_name_ (uint)
95 82 78 0 0 81 20 0 0 0 + init_declarators_
96 81 0 0 82 80 20 0 0 0 + declarator
97 80 0 0 81 79 20 0 0 0 + direct_decl_
98 79 0 0 80 77 20 0 0 0 + ident_
99 77 0 0 79 0 20 8 0 0 + IDENT_ (yy)
100
101 End of Output.
102

```

图 8 Linux 运行结果

## 4.7 结果分析

### 4.7.1 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。

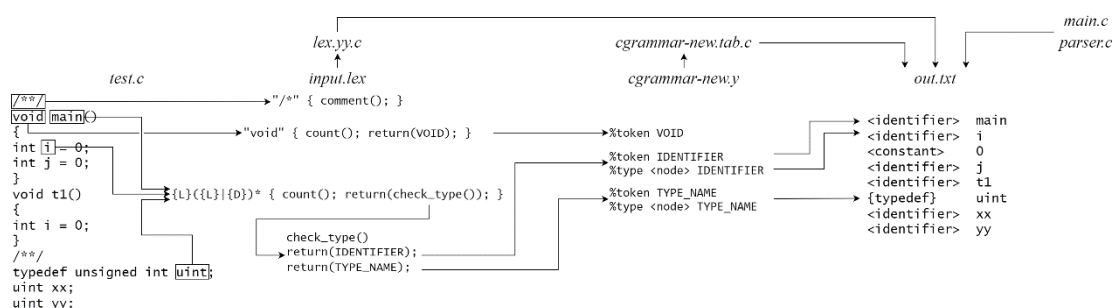


图 9 识别主要步骤

```

typedef struct Symbol
{
    char* name;           // Pointer to symbol name.                4
    short length;         // Length of symbol name.                2
    //short level;        // Level number for scope.                2
    unsigned short term;   // Terminal number (e.g. <identifier>, <string>, {typedef}, ...) 2
    unsigned short type;   // Type (e.g. integer, float, double, char, ...)                2
    int cell;              // Hash vector cell number for deleting this symbol.            4 16 bytes
}Symbol;

typedef struct Node
{
    unsigned short id;      // Node id number (65535 max).                2 2 // id与node_name数组关联, 各个类型的node与n
    unsigned short prod;    // Production number (65535 max).              2 4
    int node_index;         // node 在node数组中的索引                    4
    int sti;                // Symbol-table index (perm or temp var).        4 8 // sti与symbol数组关联
    int prev;               // Previous node.                              4 12
    int next;               // Next node.                                  4 16 // node的数组下标与id 和sti关联
    int line;               // Line number.                                4 20
    int child;              // Child node.                                 4 24
    int parent;             // Parent node.                                4 28 bytes per node
    unsigned short layer;   // 节点所在的层(与block有关, 设计到变量的作用域, 使用的范围,用uchar型也可以, 一般不会达到255层)
    unsigned char bsource;  // 表示是否是被分析的程序的节点(源程序包含头文件, 预处理之后, 被包含的头文件与源程序会放在同一个中
}Node;

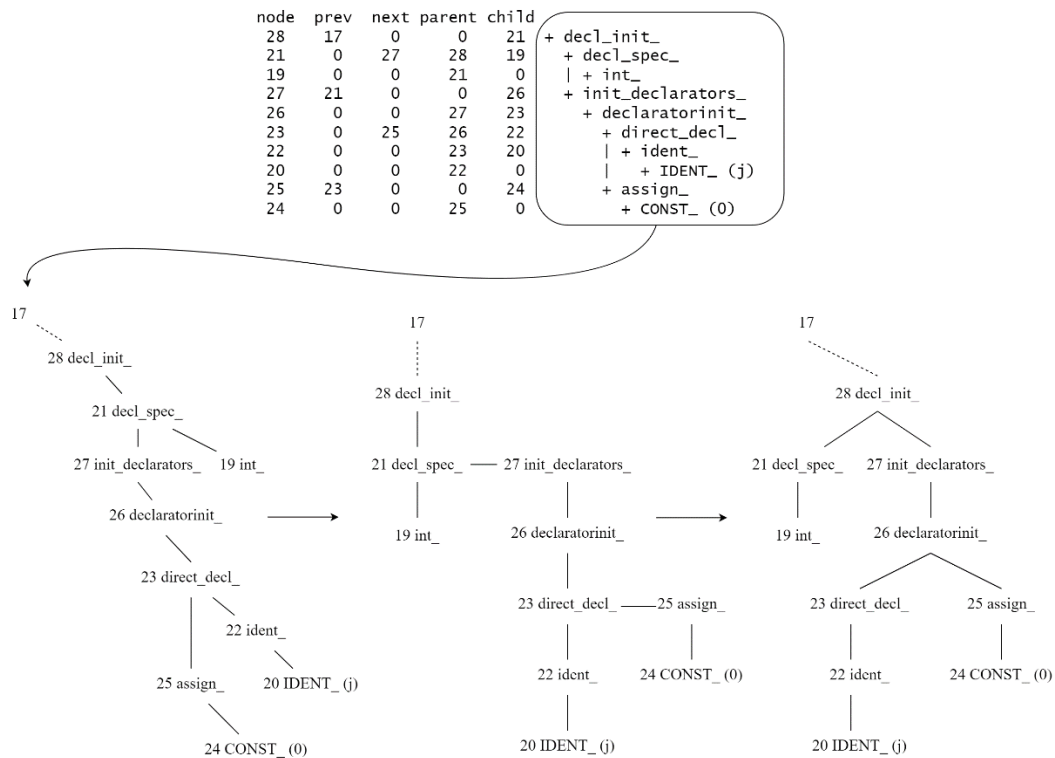
```

图 10 节点的定义

### 4.7.2 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。

```

433 Node * link(int tid, Node * rExp, ... )
434 {
435     Node * node1;
436     Node * node2;
437     va_list exps;
438     Node * parent_node = new_node();
439
440     parent_node->id = tid;
441     parent_node->line = yylineno;
442
443     if( rExp == NULL ) return parent_node;
444
445     va_start(exps,rExp);
446     node1 = rExp;
447     parent_node->child = node1->node_index;
448     node1->parent = parent_node->node_index;
449
450     node2=va_arg(exps,Node *);
451
452     while(node2!=NULL){
453         node1->next = node2->node_index;
454         node2->prev = node1->node_index;
455
456         node1 = node2;
457         node2=va_arg(exps,Node *);
458     }
459
460     va_end( exps );
461
462     return parent_node;
463 }
464
465
466

```

图 12 构造子女兄弟链 *link* 函数

## 4.8 实验总结

### 4.8.1 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。

```
C:\Users\hujunyao\AppData\Local\Temp\ccI1jduV.o:lex.yy.c:(.text+0x18f1): undefined reference to `yyinput'
C:\Users\hujunyao\AppData\Local\Temp\ccI1jduV.o:lex.yy.c:(.text+0x1905): undefined reference to `yyinput'
collect2.exe: error: ld returned 1 exit status
```

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()
```

### 4.8.2 gcc 编译找不到 input 重新编译，错误提示如下。

```
C:\Users\hujunyao\AppData\Local\Temp\ccCwlGW7.o:lex.yy.c:(.text+0x1268): undefined reference to `input'
collect2.exe: error: ld returned 1 exit status
```

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();
```

#### 4.8.3 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 ** );
```

```
main.c: In function 'main':
main.c:36:2: warning: implicit declaration of function 'print_symtab'; did you mean 'printast'? [-Wimplicit-function-declaration]
print_symtab (term_symb); // Print the symbol table contents.
~~~~~
printast
```

图 15 报警提示

#### 4.8.4 Linux 语义操作逗号

在 `cgrammar-new.y` 中，给这一行加上逗号即可。

```
cgrammar-new.y: In function 'yyparse':
cgrammar-new.y:158:147: error: expected ';' before '}' token
158 | Declaration
    |
```

图 16 逗号出错

#### 4.8.5 Linux 中的 ULONG\_MAX

在 `parser.c` 中，加上一行引用头文件即可。

```
#include <limits.h>
```

```
parser.c:339:18: error: 'ULONG_MAX' undeclared (first use in this function)
339 | hashdiv = ULONG_MAX / max_cells + 1;
    |          ^~~~~~
parser.c:4:1: note: 'ULONG_MAX' is defined in header '<limits.h>'; did you forget to '#include <limits.h>'?
3 | #include "parser.h"
+++ |+#include <limits.h>
4 |
parser.c:339:18: note: each undeclared identifier is reported only once for each function it appears in
339 | hashdiv = ULONG_MAX / max_cells + 1;
    |          ^~~~~~
```

图 17 ULONG\_MAX 出错

#### 4.8.6 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。