编译器专题实验报告

实验三:语法分析

实验内容（必做）：

任务一、完成计算器

- 支持加减乘除和括号

任务二、完成文法分析，实现输出类似语法分析树的结果

- 需要考虑出错处理

实验内容（选做）：

任务三、在任务二的基础上使用lex文件完成词法分析识别词素，实现flex和bison的联合编译，使用脚本文件或make或批处理文件实现一键编译。

实验结果：（独立模式）：

任务一和任务二已经使用脚本文件实现一键编译

任务一结果

```
lenovo@LAPTOP-U5SNIAL6 lab3 pwsh no config ./test.bat test

F:\junior_year_B\bianyi_lab\lab3>bison --yacc -dv test.y

F:\junior_year_B\bianyi_lab\lab3>flex test.l

F:\junior_year_B\bianyi_lab\lab3>gcc -o test y.tab.c lex.yy.c

F:\junior_year_B\bianyi_lab\lab3>test.exe
1+1
2
4*(1+4)
20
```

任务二结果

输入设置为1.c

```c
//1.c
int main(){
    i=1;
}
```

批处理结果为：

```
lenovo@LAPTOP-U5SNIAL6 lab3_2 pwsh no config ./test.bat test

F:\junior_year_B\bianyi_lab\lab3_2>flex test.l

F:\junior_year_B\bianyi_lab\lab3_2>bison --yacc -dv test.y

F:\junior_year_B\bianyi_lab\lab3_2>gcc -o test y.tab.c lex.yy.c
```

```
F:\junior_year_B\bianyi_lab\lab3_2>test.exe  0<1.c
CompUnit
>--FuncDef
|>--int
|>--main
|>--(
|>--FuncFParams
||>--E
|>--)
|>--Block
||>--{
||>--BlockItemBlock
|||>--BlockItemBlock
||||>--E
|||>--BlockItem
||||>--Stmt
|||||>--LVal
||||||>--i
||||||>--ExpBlock
|||||||>--E
|||||>--=
|||||>--Exp
||||||>--AddExp
|||||||>--MulExp
||||||||>--UnaryExp
|||||||||>--PrimaryExp
||||||||||>--Number
||||||||||>--1
|||||>--;
||>--}
```

输入设置为2.c

```c
//2.c
int main(){
    hello
}
```

批处理结果为

```
F:\junior_year_B\bianyi_lab\lab3_2>test.exe  0<2.c
syntax error at line 3: }
lenovo@LAPTOP-U5SNIAL6 lab3_2 pwsh no config
```

由上面实验结果可以看出，成功实现了输出类似语法分析树的结果，同时考虑了出错处理

另外遇到的问题和解决思路（可选）：

在实现Bison文件时，定义翻译规则时考虑了c程序的识别，该部分耗时较长。

代码很原创（可选）：

任务一代码：

.bat文件

```
@echo off
set str=%1
@echo on
bison --yacc -dv %str%.y
flex %str%.l
gcc -o %str% y.tab.c lex.yy.c
test.exe
```

test.l

```
%{
    #include<stdlib.h>
    void yyerror(char*);
    #include "calc.tab.h"
%}

%%
[0-9]+      {yylval=atoi(yytext); return NUMBER;}
[-+*/()\n]   {return *yytext;}
[ \t]       ;

.           yyerror("Error");

%%
int yywrap(void)
{
  return 1;
}
```

test.y

```
%token NUMBER
%left '+' '-' '*' '/'

%{
    #include <stdio.h>
    #include <math.h>
    void yyerror(char *s);
    int yylex();
%}

%%

    prog:
        | prog expr '\n' { printf("%d\n", $2); };
    expr:
```

```
          expr '+' term { $$ = $1 + $3; }
          | expr '-' term { $$ = $1 - $3; }
          | term
      term:
          | term '*' factor { $$ = $1 * $3; }
          | term '/' factor { $$ = $1 / $3; }
          | factor
          ;
      factor:
          NUMBER {};
          | '(' expr ')' { $$ = $2; }
          ;
  %%

  void yyerror(char *s) {
      fprintf(stderr, "%s\n", s);
  }

  int main() {
      yyparse();
      return 0;
  }
```

任务二代码：

.bat文件

```
@echo off
set str=%1
@echo on
flex %str%.l
bison --yacc -dv %str%.y
gcc -o %str% y.tab.c lex.yy.c
test.exe <1.c

```

test.l文件

```
%option yylineno

%{
    #include "y.tab.h"
    #define UNEXPECTED 0
%}

INT                 int
VOID                void
CONST               const
IF                  if
ELSE                else
WHILE               while
BREAK               break
CONTINUE            continue
RETURN              return
MULDIVSUR           "*"|"/"|"%"
ADDSUB              "+"|"-"
```

```
CMP                 "<"|">"|"<="|">="
EQNEQ               "=="|"!="
ASSIGN              "="
NONZERO             [1-9]
DIGIT               [0-9]
LETTER              [A-Za-z]
OCTAL_DIGIT         [0-7]
OCTAL_CONST         0{OCTAL_DIGIT}*
ILLEGAL_OCTAL_CONST 0[0-9a-wy-zA-WY-Z]({LETTER}|{DIGIT})*
HEX_PREFIX          0x|0X
HEX_DIGIT           [0-9a-fA-F]
HEX_CONST           {HEX_PREFIX}{HEX_DIGIT}+
ILLEGAL_HEX_CONST   {HEX_PREFIX}({LETTER}|{DIGIT})*
NONDIGIT            {LETTER}|"_"
ID                  {NONDIGIT}({DIGIT}|{NONDIGIT})*
DEC_CONST           {NONZERO}{DIGIT}*
COMMENT1            "/*"[^*]*"*"+([^*/][^*]*"*"+)*"/"
COMMENT2            "//".*


%%

{INT}                   { yylval.str=strdup(yytext); return INT; }
{VOID}                  { yylval.str=strdup(yytext); return VOID; }
{OCTAL_CONST}           { yylval.str=strdup(yytext); return OCTAL_CONST; }
{ILLEGAL_OCTAL_CONST}   { yylval.str=strdup(yytext); return HEX_CONST; }
{HEX_CONST}             { yylval.str=strdup(yytext); return HEX_CONST; }
{ILLEGAL_HEX_CONST}     { yylval.str=strdup(yytext); return DEC_CONST; }
{DEC_CONST}             { yylval.str=strdup(yytext); return DEC_CONST; }
{CONST}                 { yylval.str=strdup(yytext); return CONST; }
{IF}                    { yylval.str=strdup(yytext); return IF; }
{ELSE}                  { yylval.str=strdup(yytext); return ELSE; }
{WHILE}                 { yylval.str=strdup(yytext); return WHILE; }
{BREAK}                 { yylval.str=strdup(yytext); return BREAK; }
{CONTINUE}              { yylval.str=strdup(yytext); return CONTINUE; }
{RETURN}                { yylval.str=strdup(yytext); return RETURN; }
{MULDIVSUR}             { yylval.str=strdup(yytext); return MULDIVSUR; }
{ADDSUB}                { yylval.str=strdup(yytext); return ADDSUB; }
{CMP}                   { yylval.str=strdup(yytext); return CMP; }
{EQNEQ}                 { yylval.str=strdup(yytext); return EQNEQ; }
{ASSIGN}                { yylval.str=strdup(yytext); return ASSIGN; }
{ID}                    { yylval.str=strdup(yytext); return ID; }
"("                     { yylval.str=strdup(yytext); return yytext[0]; }
")"                     { yylval.str=strdup(yytext); return yytext[0]; }
"["                     { yylval.str=strdup(yytext); return yytext[0]; }
"]"                     { yylval.str=strdup(yytext); return yytext[0]; }
"{"                     { yylval.str=strdup(yytext); return yytext[0]; }
"}"                     { yylval.str=strdup(yytext); return yytext[0]; }
";"                     { yylval.str=strdup(yytext); return yytext[0]; }
","                     { yylval.str=strdup(yytext); return yytext[0]; }
"&&"                    { yylval.str=strdup(yytext); return AND; }
"||"                    { yylval.str=strdup(yytext); return OR; }
{COMMENT1}|{COMMENT2}   { }
[ \t\n]                 { }
.                       { yylval.str=strdup(yytext); return UNEXPECTED; }
%%
```

```
74
75  int yywrap(void)
76  {
77      return 1;
78  }
```

test.y文件

```
1   %start CompUnit
2   %expect 1
3
4   %{
5       #include "parser.h"
6   %}
7
8   %union
9   {
10      int     num;
11      char*   str;
12      struct ASTnode* node; /*"struct" is indispensable*/
13  }
14
15  %token <str> INT VOID CONST IF ELSE WHILE BREAK CONTINUE RETURN ID OCTAL_CONST
    HEX_CONST DEC_CONST
16  %right <str> ASSIGN
17  %left <str> OR
18  %left <str> AND
19  %left <str> EQNEQ
20  %left <str> CMP
21  %left <str> ADDSUB
22  %left <str> MULDIVSUR
23
24  %type<node> Number CompUnit Decl FuncDef ConstDecl VarDecl ConstDef
    ConstDefBlock ConstExpBlock ConstInitVal ConstExp ConstInitFlag ConstValBlock
    VarDef
25      VarDefFlag InitVal Exp InitValFlag InitValBlock FuncFParams Block FuncFParam
    FuncFParamBlock ExpBlockFlag ExpBlock BlockItemBlock BlockItem
26      Stmt LVal ExpFlag StmtFlag Cond AddExp LOrExp PrimaryExp UnaryExp
    FuncFParamsFlag FuncRParams UNARYOP CommaExpBlock MulExp RelExp EqExp LAndExp
27
28  %%
29  CompUnit:       CompUnit Decl                           {
30
     connectASTnode(2,$1,$2);
31
     ASThead=$$=newASTnode(TEXT,"CompUnit",0,NULL,$1);
32                                                          }
33                  | CompUnit FuncDef                      {
34
     connectASTnode(2,$1,$2);
35
     ASThead=$$=newASTnode(TEXT,"CompUnit",0,NULL,$1);
36                                                          }
37                  | Decl
    {ASThead=$$=newASTnode(TEXT,"CompUnit",0,NULL,$1);}
```

```
38                  | FuncDef
    {ASThead=$$=newASTnode(TEXT,"CompUnit",0,NULL,$1);}
39  Decl:           ConstDecl
    {$$=newASTnode(TEXT,"Decl",0,NULL,$1);}
40                  | VarDecl
    {$$=newASTnode(TEXT,"Decl",0,NULL,$1);}
41  ConstDecl:      CONST INT ConstDef ConstDefBlock ';'    {
42                                                              ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
43
     *n2=newASTnode(TEXT,$2,0,NULL,NULL),
44
     *n5=newASTnode(TEXT,";",0,NULL,NULL);
45
     connectASTnode(5,n1,n2,$3,$4,n5);
46
     $$=newASTnode(TEXT,"ConstDecl",0,NULL,n1);
47                                                          }
48  ConstDefBlock:  ConstDefBlock ',' ConstDef              {
49                                                              ASTnode
    *n=newASTnode(TEXT,",",0,$3,NULL);
50
     connectASTnode(3,$1,n,$3);
51
     $$=newASTnode(TEXT,"ConstDefBlock",0,NULL,$1);
52                                                          }
53                  | /*E*/                                 {
54                                                              ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
55
     $$=newASTnode(TEXT,"ConstDefBlock",0,NULL,n);
56                                                          }
57  ConstDef:       ID ConstExpBlock ASSIGN ConstInitVal    {
58                                                              ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
59
     *n3=newASTnode(TEXT,$3,0,NULL,NULL);
60
     connectASTnode(4,n1,$2,n3,$4);
61
     $$=newASTnode(TEXT,"ConstDef",0,NULL,n1);
62                                                          }
63  ConstExpBlock:  ConstExpBlock '[' ConstExp ']'          {
64                                                              ASTnode
    *n2=newASTnode(TEXT,"[",0,NULL,NULL),
65
     *n4=newASTnode(TEXT,"]",0,NULL,NULL);
66
     connectASTnode(4,$1,n2,$3,n4);
67
     $$=newASTnode(TEXT,"ConstExpBlock",0,NULL,$1);
68                                                          }
69                  | /*E*/                                 {
70                                                              ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
```

```
71
     $$=newASTnode(TEXT,"ConstExpBlock",0,NULL,n);
72                                                          }
73   ConstInitVal:   ConstExp
     {$$=newASTnode(TEXT,"ConstInitVal",0,NULL,$1);}
74                 |'{'ConstInitFlag'}'                     {
75                                                              ASTnode
     *n1=newASTnode(TEXT,"{",0,NULL,NULL),
76
      *n3=newASTnode(TEXT,"}",0,NULL,NULL);
77
      connectASTnode(3,n1,$2,n3);
78
      $$=newASTnode(TEXT,"ConstInitVal",0,NULL,n1);
79                                                          }
80   ConstInitFlag:  ConstInitVal ConstValBlock              {
81
      connectASTnode(2,$1,$2);
82
      $$=newASTnode(TEXT,"ConstInitFlag",0,NULL,$1);
83                                                          }
84                 | /*E*/                                  {
85                                                              ASTnode
     *n=newASTnode(TEXT,"E",0,NULL,NULL);
86
      $$=newASTnode(TEXT,"ConstInitFlag",0,NULL,n);
87                                                          }
88   ConstValBlock:  ConstValBlock ',' ConstInitVal          {
89                                                              ASTnode
     *n2=newASTnode(TEXT,",",0,NULL,NULL);
90
      connectASTnode(3,$1,n2,$3);
91
      $$=newASTnode(TEXT,"ConstValBlock",0,NULL,$1);
92                                                          }
93                 | /*E*/                                  {
94                                                              ASTnode
     *n=newASTnode(TEXT,"E",0,NULL,NULL);
95
      $$=newASTnode(TEXT,"ConstValBlock",0,NULL,n);
96                                                          }
97   VarDecl:        INT VarDef VarDefFlag ';'               {
98                                                              ASTnode
     *n1=newASTnode(TEXT,$1,0,NULL,NULL),
99
      *n4=newASTnode(TEXT,";",0,NULL,NULL);
100
      connectASTnode(4,n1,$2,$3,n4);
101
      $$=newASTnode(TEXT,"VarDecl",0,NULL,n1);
102                                                         }
103  VarDefFlag:    ',' VarDef VarDefFlag                   {
104                                                             ASTnode
     *n1=newASTnode(TEXT,",",0,NULL,NULL);
105
      connectASTnode(3,n1,$2,$3);
```

```
106
       $$=newASTnode(TEXT,"VarDefFlag",0,NULL,n1);
107                                                     }
108                     | /*E*/                         {
109                                                         ASTnode
       *n=newASTnode(TEXT,"E",0,NULL,NULL);
110
       $$=newASTnode(TEXT,"VarDefFlag",0,NULL,n);
111                                                     }
112     VarDef:         ID ConstExpBlock                {
113                                                         ASTnode
       *n1=newASTnode(TEXT,$1,0,NULL,NULL);
114
       connectASTnode(2,n1,$2);
115
       $$=newASTnode(TEXT,"VarDef",0,NULL,n1);
116                                                     }
117                     | ID ConstExpBlock ASSIGN InitVal       {
118                                                         ASTnode
       *n1=newASTnode(TEXT,$1,0,NULL,NULL),
119
       *n3=newASTnode(TEXT,$3,0,NULL,NULL);
120
       connectASTnode(4,n1,$2,n3,$4);
121
       $$=newASTnode(TEXT,"VarDef",0,NULL,n1);
122                                                     }
123     InitVal:          Exp
       {$$=newASTnode(TEXT,"InitVal",0,NULL,$1);}
124                     | '{'InitValFlag'}'             {
125                                                         ASTnode
       *n1=newASTnode(TEXT,"{",0,NULL,NULL),
126
       *n3=newASTnode(TEXT,"}",0,NULL,NULL);
127
       connectASTnode(3,n1,$2,n3);
128
       $$=newASTnode(TEXT,"InitVal",0,NULL,n1);
129                                                     }
130     InitValFlag:    InitVal InitValBlock            {
131
       connectASTnode(2,$1,$2);
132
       $$=newASTnode(TEXT,"InitValFlag",0,NULL,$1);
133                                                     }
134                     | /*E*/                         {
135                                                         ASTnode
       *n=newASTnode(TEXT,"E",0,NULL,NULL);
136
       $$=newASTnode(TEXT,"InitValFlag",0,NULL,n);
137                                                     }
138     InitValBlock:   InitValBlock ',' InitVal        {
139                                                         ASTnode
       *n2=newASTnode(TEXT,",",0,NULL,NULL);
140
       connectASTnode(3,$1,n2,$3);
```

```
141
     $$=newASTnode(TEXT,"InitValBlock",0,NULL,$1);
142                                                         }
143                     | /*E*/                             {
144                                                             ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
145
     $$=newASTnode(TEXT,"InitValFlag",0,NULL,n);
146                                                         }
147 FuncDef:        INT ID '(' FuncFParams')' Block         {
148                                                             ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
149
     *n2=newASTnode(TEXT,$2,0,NULL,NULL),
150
     *n3=newASTnode(TEXT,"(",0,NULL,NULL),
151
     *n5=newASTnode(TEXT,")",0,NULL,NULL);
152
     connectASTnode(6,n1,n2,n3,$4,n5,$6);
153
     $$=newASTnode(TEXT,"FuncDef",0,NULL,n1);
154                                                         }
155                     | VOID ID '(' FuncFParams')' Block  {
156                                                             ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
157
     *n2=newASTnode(TEXT,$2,0,NULL,NULL),
158
     *n3=newASTnode(TEXT,"(",0,NULL,NULL),
159
     *n5=newASTnode(TEXT,")",0,NULL,NULL);
160
     connectASTnode(6,n1,n2,n3,$4,n5,$6);
161
     $$=newASTnode(TEXT,"FuncDef",0,NULL,n1);
162                                                         }
163 FuncFParams:    FuncFParam FuncFParamBlock              {
164
     connectASTnode(2,$1,$2);
165
     $$=newASTnode(TEXT,"FuncFParams",0,NULL,$1);
166                                                         }
167                     | /*E*/                             {
168                                                             ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
169
     $$=newASTnode(TEXT,"FuncFParams",0,NULL,n);
170                                                         }
171 FuncFParamBlock:FuncFParamBlock ',' FuncFParam          {
172                                                             ASTnode
    *n2=newASTnode(TEXT,",",0,NULL,NULL);
173
     connectASTnode(3,$1,n2,$3);
174
     $$=newASTnode(TEXT,"FuncFParamBlock",0,NULL,$1);
```

```
175                                                     }
176                 | /*E*/                            {
177                                                         ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
178
     $$=newASTnode(TEXT,"FuncFParamBlock",0,NULL,n);
179                                                     }
180 FuncFParam:     INT ID ExpBlockFlag                 {
181                                                         ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
182
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
183
     connectASTnode(3,n1,n2,$3);
184
     $$=newASTnode(TEXT,"FuncFParam",0,NULL,n1);
185                                                     }
186 ExpBlockFlag:   '['']' ExpBlock                     {
187                                                         ASTnode
    *n1=newASTnode(TEXT,"[",0,NULL,NULL),
188
     *n2=newASTnode(TEXT,"]",0,NULL,NULL);
189
     connectASTnode(3,n1,n2,$3);
190
     $$=newASTnode(TEXT,"ExpBlockFlag",0,NULL,n1);
191                                                     }
192                 | /*E*/                            {
193                                                         ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
194
     $$=newASTnode(TEXT,"ExpBlockFlag",0,NULL,n);
195                                                     }
196 ExpBlock:       ExpBlock '['Exp']'                  {
197                                                         ASTnode
    *n2=newASTnode(TEXT,"[",0,NULL,NULL),
198
     *n4=newASTnode(TEXT,"]",0,NULL,NULL);
199
     connectASTnode(4,$1,n2,$3,n4);
200
     $$=newASTnode(TEXT,"FuncDef",0,NULL,$1);
201                                                     }
202                 | /*E*/                            {
203                                                         ASTnode
    *n=newASTnode(TEXT,"E",0,NULL,NULL);
204
     $$=newASTnode(TEXT,"ExpBlock",0,NULL,n);
205                                                     }
206 Block:          '{' BlockItemBlock '}'              {
207                                                         ASTnode
    *n1=newASTnode(TEXT,"{",0,NULL,NULL),
208
     *n3=newASTnode(TEXT,"}",0,NULL,NULL);
209
     connectASTnode(3,n1,$2,n3);
```

```
210
     $$=newASTnode(TEXT,"Block",0,NULL,n1);
211                                                  }
212  BlockItemBlock: BlockItemBlock BlockItem        {
213
     connectASTnode(2,$1,$2);
214
     $$=newASTnode(TEXT,"BlockItemBlock",0,NULL,$1);
215                                                  }
216                | /*E*/                           {
217                                                      ASTnode
     *n=newASTnode(TEXT,"E",0,NULL,NULL);
218
     $$=newASTnode(TEXT,"BlockItemBlock",0,NULL,n);
219                                                  }
220  BlockItem:        Decl
     {$$=newASTnode(TEXT,"BlockItem",0,NULL,$1);}
221                | Stmt
     {$$=newASTnode(TEXT,"BlockItem",0,NULL,$1);}
222  Stmt:           LVal ASSIGN Exp ';' %prec ASSIGN {
223                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL),
224
     *n4=newASTnode(TEXT,";",0,NULL,NULL);
225
     connectASTnode(4,$1,n2,$3,n4);
226
     $$=newASTnode(TEXT,"Stmt",0,NULL,$1);
227                                                  }
228                | ExpFlag';'                      {
229                                                      ASTnode
     *n2=newASTnode(TEXT,";",0,NULL,NULL);
230
     connectASTnode(2,$1,n2);
231
     $$=newASTnode(TEXT,"Stmt",0,NULL,$1);
232                                                  }
233                | Block
     {$$=newASTnode(TEXT,"Stmt",0,NULL,$1);}
234                | IF'('Cond')' Stmt StmtFlag      {
235                                                      ASTnode
     *n1=newASTnode(TEXT,$1,0,NULL,NULL),
236
     *n2=newASTnode(TEXT,"(",0,NULL,NULL),
237
     *n4=newASTnode(TEXT,")",0,NULL,NULL);
238
     connectASTnode(6,n1,n2,$3,n4,$5,$6);
239
     $$=newASTnode(TEXT,"Stmt",0,NULL,n1);
240                                                  }
241                | WHILE'('Cond')' Stmt            {
242                                                      ASTnode
     *n1=newASTnode(TEXT,$1,0,NULL,NULL),
243
     *n2=newASTnode(TEXT,"(",0,NULL,NULL),
```

```
244
       *n4=newASTnode(TEXT,")",0,NULL,NULL);
245
       connectASTnode(5,n1,n2,$3,n4,$5);
246
       $$=newASTnode(TEXT,"Stmt",0,NULL,n1);
247                                                    }
248                  | BREAK';'                        {
249                                                        ASTnode
      *n1=newASTnode(TEXT,$1,0,NULL,NULL),
250
       *n2=newASTnode(TEXT,";",0,NULL,NULL);
251
       connectASTnode(2,n1,n2);
252
       $$=newASTnode(TEXT,"Stmt",0,NULL,n1);
253                                                    }
254                  | CONTINUE';'                     {
255                                                        ASTnode
      *n1=newASTnode(TEXT,$1,0,NULL,NULL),
256
       *n2=newASTnode(TEXT,";",0,NULL,NULL);
257
       connectASTnode(2,n1,n2);
258
       $$=newASTnode(TEXT,"Stmt",0,NULL,n1);
259                                                    }
260                  | RETURN ExpFlag';'               {
261                                                        ASTnode
      *n1=newASTnode(TEXT,$1,0,NULL,NULL),
262
       *n3=newASTnode(TEXT,";",0,NULL,NULL);
263
       connectASTnode(3,n1,$2,n3);
264
       $$=newASTnode(TEXT,"Stmt",0,NULL,n1);
265                                                    }
266    ExpFlag:          Exp
      {$$=newASTnode(TEXT,"ExpFlag",0,NULL,$1);}
267                  | /*E*/                           {
268                                                        ASTnode
      *n=newASTnode(TEXT,"E",0,NULL,NULL);
269
       $$=newASTnode(TEXT,"ExpFlag",0,NULL,n);
270                                                    }
271    StmtFlag:       ELSE Stmt                       {
272                                                        ASTnode
      *n1=newASTnode(TEXT,$1,0,NULL,NULL);
273
       connectASTnode(2,n1,$2);
274
       $$=newASTnode(TEXT,"StmtFlag",0,NULL,n1);
275                                                    }
276                  | /*E*/                           {
277                                                        ASTnode
      *n=newASTnode(TEXT,"E",0,NULL,NULL);
```

```
278
 $$=newASTnode(TEXT,"StmtFlag",0,NULL,n);
279                                                 }
280 Exp:            AddExp
    {$$=newASTnode(TEXT,"Exp",0,NULL,$1);}
281 Cond:           LOrExp
    {$$=newASTnode(TEXT,"Cond",0,NULL,$1);}
282 LVal:           ID ExpBlock                     {
283                                                     ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL);
284
 connectASTnode(2,n1,$2);
285
 $$=newASTnode(TEXT,"LVal",0,NULL,n1);
286                                                 }
287 PrimaryExp:     '('Exp')'                       {
288                                                     ASTnode
    *n1=newASTnode(TEXT,"(",0,NULL,NULL),
289
 *n3=newASTnode(TEXT,")",0,NULL,NULL);
290
 connectASTnode(3,n1,$2,n3);
291
 $$=newASTnode(TEXT,"PrimaryExp",0,NULL,n1);
292                                                 }
293                 | LVal
    {$$=newASTnode(TEXT,"PrimaryExp",0,NULL,$1);}
294                 | Number
    {$$=newASTnode(TEXT,"PrimaryExp",0,NULL,$1);}
295 Number:         OCTAL_CONST                     {
296                                                     ASTnode
    *n=newASTnode(NUM,NULL,OCT2DEC($1),NULL,NULL);
297
 $$=newASTnode(TEXT,"Number",0,NULL,n);
298                                                 }
299                 | HEX_CONST                     {
300                                                     ASTnode
    *n=newASTnode(NUM,NULL,HEX2DEC($1),NULL,NULL);
301
 $$=newASTnode(TEXT,"Number",0,NULL,n);
302                                                 }
303                 | DEC_CONST                     {
304                                                     ASTnode
    *n=newASTnode(NUM,NULL,atoi($1),NULL,NULL);
305
 $$=newASTnode(TEXT,"Number",0,NULL,n);
306                                                 }
307 UnaryExp:       PrimaryExp
    {$$=newASTnode(TEXT,"UnaryExp",0,NULL,$1);}
308                 | ID '(' FuncFParamsFlag ')'    {
309                                                     ASTnode
    *n1=newASTnode(TEXT,$1,0,NULL,NULL),
310
 *n2=newASTnode(TEXT,"(",0,NULL,NULL),
311
 *n4=newASTnode(TEXT,")",0,NULL,NULL);
```

```
312
     connectASTnode(4,n1,n2,$3,n4);
313
     $$=newASTnode(TEXT,"PrimaryExp",0,NULL,n1);
314                                                  }
315                  | UNARYOP UnaryExp              {
316
     connectASTnode(2,$1,$2);
317
     $$=newASTnode(TEXT,"UnaryExp",0,NULL,$1);
318                                                  }
319  FuncFParamsFlag:FuncRParams
     {$$=newASTnode(TEXT,"FuncFParamsFlag",0,NULL,$1);}
320                  | /*E*/                         {
321                                                      ASTnode
     *n=newASTnode(TEXT,"E",0,NULL,NULL);
322
     $$=newASTnode(TEXT,"FuncFParamsFlag",0,NULL,n);
323                                                  }
324  FuncRParams:    Exp CommaExpBlock               {
325
     connectASTnode(2,$1,$2);
326
     $$=newASTnode(TEXT,"FuncRParams",0,NULL,$1);
327                                                  }
328  CommaExpBlock:  CommaExpBlock ',' Exp           {
329                                                      ASTnode
     *n2=newASTnode(TEXT,",",0,NULL,NULL);
330
     connectASTnode(3,$1,n2,$3);
331
     $$=newASTnode(TEXT,"CommaExpBlock",0,NULL,$1);
332                                                  }
333                  | /*E*/                         {
334                                                      ASTnode
     *n=newASTnode(TEXT,"E",0,NULL,NULL);
335
     $$=newASTnode(TEXT,"CommaExpBlock",0,NULL,n);
336                                                  }
337  UNARYOP:        ADDSUB                          {
338                                                      ASTnode
     *n=newASTnode(TEXT,$1,0,NULL,NULL);
339
     $$=newASTnode(TEXT,"UNARYOP",0,NULL,n);
340                                                  }
341                  | '!'                           {
342                                                      ASTnode
     *n=newASTnode(TEXT,"!",0,NULL,NULL);
343
     $$=newASTnode(TEXT,"UNARYOP",0,NULL,n);
344                                                  }
345  MulExp:         UnaryExp
     {$$=newASTnode(TEXT,"MulExp",0,NULL,$1);}
346                  | MulExp MULDIVSUR UnaryExp     {
347                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
```

```
348
     connectASTnode(3,$1,n2,$3);
349
     $$=newASTnode(TEXT,"MulExp",0,NULL,$1);
350                                                  }
351  AddExp:         MulExp
     {$$=newASTnode(TEXT,"AddExp",0,NULL,$1);}
352                  | AddExp ADDSUB MulExp          {
353                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
354
     connectASTnode(3,$1,n2,$3);
355
     $$=newASTnode(TEXT,"MulExp",0,NULL,$1);
356                                                  }
357  RelExp:         AddExp
     {$$=newASTnode(TEXT,"RelExp",0,NULL,$1);}
358                  | RelExp CMP AddExp             {
359                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
360
     connectASTnode(3,$1,n2,$3);
361
     $$=newASTnode(TEXT,"RelExp",0,NULL,$1);
362                                                  }
363  EqExp:          RelExp
     {$$=newASTnode(TEXT,"EqExp",0,NULL,$1);}
364                  | EqExp EQNEQ RelExp            {
365                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
366
     connectASTnode(3,$1,n2,$3);
367
     $$=newASTnode(TEXT,"EqExp",0,NULL,$1);
368                                                  }
369  LAndExp:        EqExp
     {$$=newASTnode(TEXT,"LAndExp",0,NULL,$1);}
370                  | LAndExp AND EqExp             {
371                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
372
     connectASTnode(3,$1,n2,$3);
373
     $$=newASTnode(TEXT,"LAndExp",0,NULL,$1);
374                                                  }
375  LOrExp:         LAndExp
     {$$=newASTnode(TEXT,"LOrExp",0,NULL,$1);}
376                  | LOrExp OR LAndExp             {
377                                                      ASTnode
     *n2=newASTnode(TEXT,$2,0,NULL,NULL);
378
     connectASTnode(3,$1,n2,$3);
379
     $$=newASTnode(TEXT,"LOrExp",0,NULL,$1);
380                                                  }
```

```
381  ConstExp:       AddExp
     {$$=newASTnode(TEXT,"ConstExp",0,NULL,$1);}
382  %%
383
384  int main()
385  {
386      yyparse();
387      outputAST(ASThead,0);
388      freeAST(ASThead);
389      return 0;
390  }
```