用 Yacc 设计语法分析器 2

一、实验目的：

学习如何使用 Lex 和 Yacc 设计一个语法分析器，并学习如何在语法分析的同时生成分析树。

二、实验内容：

给产生式加上动作，动作为生成一棵语法分析树。这棵分析树的结构可以使用或参照例子 parser1 中 ast.h 文件中定义的分析树结构。

三、实验要求：

1. 输入为实验 1 所给语言写的源程序文件；输出为一棵语法分析树，这棵语法分析树的表示方法可以是这样两种：1.将分析树的数据结构打印出来；2.按分析树的结构输出一个 C 语言源程序文件（即输入是所给语言的源程序文件，输出为语义相同的 C 语言源程序文件）。
2. 在 cygwin 下用 flex，bison 和 gcc 工具将实验调试通过，并且你写的语法分析器至少应该能通过例子 parser1 中 testcases 目录下的 test0.p 和 test1.p 两个测试例。

四、具体实现

ast.c 源程序

```c
/* 本文件给出抽象语法树上各数据结构的构造函数 */

#include "ast.h"

/* make id node 标识符和变量结点 */
a_id A_Id(a_pos pos, string val){
  a_id ret = checked_malloc(sizeof(*ret));
  ret->pos = pos;
  ret->val = val;
  return ret;
}

/* make exp node from integer 整形数变量 */
a_exp A_IntExp(a_pos pos, int ival){
  a_exp ret = checked_malloc(sizeof(*ret));
  ret->kind = A_intExp;
  ret->pos = pos;
  ret->exp.ival = ival;
  return ret;
}

/* make exp node from real 浮点数变量 */
a_exp A_RealExp(a_pos pos, double fval){
  a_exp ret = checked_malloc(sizeof(*ret));
  ret->kind = A_realExp;
  ret->pos = pos;
  ret->exp.fval = fval;
  return ret;
}
```

```
30
31  /* make exp node from id node 变量说明 */
32  a_exp A_VarExp(a_pos pos, a_id var){
33    a_exp ret = checked_malloc(sizeof(*ret));
34    ret->kind = A_varExp;
35    ret->pos = pos;
36    ret->exp.var = var;
37    return ret;
38  }
39
40  /* make binary operands exp node 算术表达式 */
41  a_exp A_OpExp(a_pos pos, a_op op, a_exp left, a_exp right){
42    a_exp ret = checked_malloc(sizeof(*ret));
43    ret->kind = A_opExp;
44    ret->pos = pos;
45    ret->exp.biopExp.op = op;
46    ret->exp.biopExp.left = left;
47    ret->exp.biopExp.right = right;
48    return ret;
49  }
50
51  /* make boolean exp node 关系表达式 */
52  a_bexp A_BExp(a_pos pos, a_bop bop, a_exp left, a_exp right){
53    a_bexp ret = checked_malloc(sizeof(*ret));
54    ret->pos = pos;
55    ret->bexp.bop = bop;
56    ret->bexp.left = left;
57    ret->bexp.right = right;
58    return ret;
59  }
```

```
60
61  /* make assign statement 赋值语句 */
62  a_stm A_Assign (a_pos pos, a_id var, a_exp exp){
63    a_stm ret = checked_malloc(sizeof(*ret));
64    ret->kind = A_assign;
65    ret->pos = pos;
66    ret->stm.assign.var = var;
67    ret->stm.assign.exp = exp;
68    return ret;
69  }
70
71  /* make if statement if语句 */
72  a_stm A_If(a_pos pos, a_bexp b, a_stm s1, a_stm s2){
73    a_stm ret = checked_malloc(sizeof(*ret));
74    ret->kind = A_if;
75    ret->pos = pos;
76    ret->stm.iff.b = b;
77    ret->stm.iff.s1 = s1;
78    ret->stm.iff.s2 = s2;
79    return ret;
80  }
81
82  /* make while statement while语句 */
83  a_stm A_While(a_pos pos, a_bexp b, a_stm s){
84    a_stm ret = checked_malloc(sizeof(*ret));
85    ret->kind = A_while;
86    ret->pos = pos;
87    ret->stm.whilee.b = b;
88    ret->stm.whilee.s = s;
89    return ret;
90  }
```

```
91
92    /* make sequence statement 复合语句*/
93  a_stm A_Seq(a_pos pos, a_stm_list sl){
94      a_stm ret = checked_malloc(sizeof(*ret));
95      ret->kind = A_seq;
96      ret->pos = pos;
97      ret->stm.seq = sl;
98      return ret;
99    }
100
101   /* make statement list 复合语句*/
102 a_stm_list A_StmList(a_stm s, a_stm_list sl){
103     a_stm_list ret = checked_malloc(sizeof(*ret));
104     ret->head = s;
105     ret->tail = sl;
106     return ret;
107   }
108
109   /* make var list 变量说明表*/
110 a_var_list A_VarList(a_id v, a_var_list vl){
111     a_var_list ret = checked_malloc(sizeof(*ret));
112     ret->head = v;
113     ret->tail = vl;
114     return ret;
115   }
116
117   /* make variable declaration node 变量说明表*/
118 a_dec A_VarDec(a_pos pos, a_var_list vl, ttype t){
119     a_dec ret = checked_malloc(sizeof(*ret));
120     ret->type = t;
121     ret->pos = pos;
122     ret->varlist = vl;
123     return ret;
124   }
```

```
125
126   /* make variable declaration list 变量说明表*/
127 a_dec_list A_DecList(a_dec vd, a_dec_list vdl){
128     a_dec_list ret = checked_malloc(sizeof(*ret));
129     ret->head = vd;
130     ret->tail = vdl;
131     return ret;
132   }
133
134   /* make program node 程序 分程序*/
135 a_prog A_Prog (a_pos pos, char * name, a_dec_list dl, a_stm_list sl){
136     a_prog ret = checked_malloc(sizeof(*ret));
137     ret->name = name;
138     ret->pos =pos;
139     ret->declist = dl;
140     ret->stmlist = sl;
141     return ret;
142   }
```

- 这部分相对简单，根据 ast.h 头文件对各种结点数据结构的定义，对传过来的参数进行赋值即可。

Yacc 文件规则段

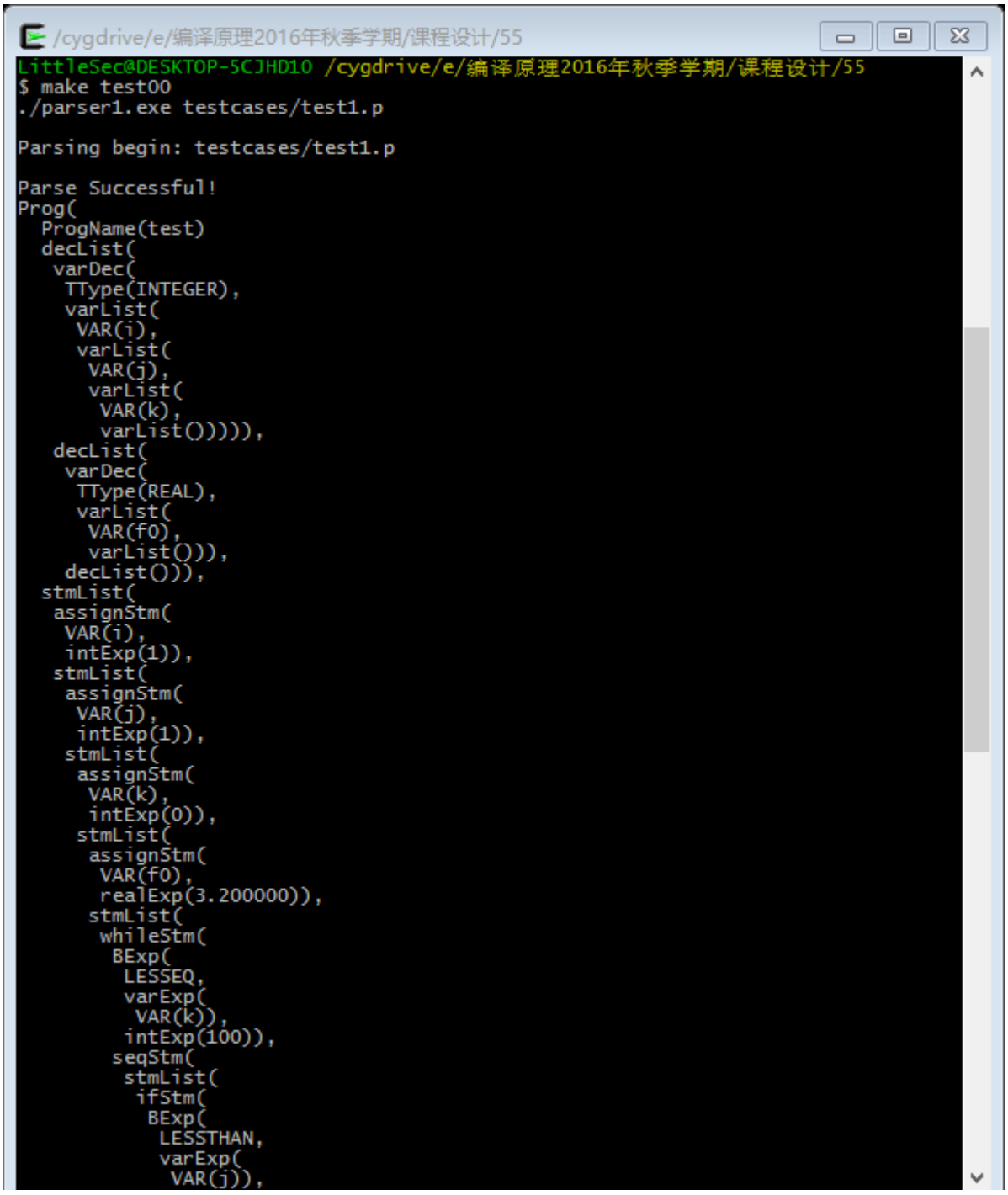
```
70  program :   PROGRAM ID SEMICOLON vardec BEGINN stmtslist END PERIOD
71                  {program = A_Prog(EM_tokPos, $2, $4, $6);}
72              ; /* 程序 分程序*/
73
74  vardec  :   VAR declist {$$ = $2;}
75              ; /* 变量说明 */
76
77  declist :   idlist COLON INTEGER SEMICOLON declist_other_part
78                  {$$ = A_DecList(A_VarDec(EM_tokPos, $1, T_int), $5);}
79              | idlist COLON REAL SEMICOLON declist_other_part
80                  {$$ = A_DecList(A_VarDec(EM_tokPos, $1, T_real), $5);}
81              ; /* 变量说明表 */
82
83  declist_other_part : declist
84              | {$$ = NULL;}
85              ;
86
87  idlist  :   ID {$$ = A_VarList(A_Id(EM_tokPos, $1), NULL);}
88              | ID COMMA idlist {$$ = A_VarList(A_Id(EM_tokPos, $1), $3);}
89              ; /* 变量表 */
90
91  stmtslist:  stmts {$$ = A_StmList($1, NULL);}
92              | stmts SEMICOLON stmtslist {$$ = A_StmList($1, $3);}
93              ; /* 语句表 */
94
95  stmts   :   asstmts
96              | ifstmts
97              | whilestmts
98              | comstmts
99              ; /* 语句 */
100
101 asstmts :   ID ASSIGN aritexp
102                 {$$ = A_Assign(EM_tokPos, A_Id(EM_tokPos, $1), $3);}
103             ; /* 赋值语句 */
104
105 ifstmts:    IF relaexp THEN stmts ELSE stmts
106                 {$$ = A_If(EM_tokPos, $2, $4, $6);}
107             ; /* 条件语句 */
```

```
109 whilestmts: WHILE relaexp DO stmts
110                 {$$ = A_While(EM_tokPos, $2, $4);}
111             ; /* WHILE语句 */
112
113 comstmts:   BEGINN stmtslist END
114                 {$$ = A_Seq(EM_tokPos, $2);}
115             ; /* 复合语句 */
116
117 aritexp :   term
118             | aritexp PLUS term {$$ = A_OpExp(EM_tokPos, A_plusOp, $1, $3);}
119             | aritexp MINUS term {$$ = A_OpExp(EM_tokPos, A_minusOp, $1, $3);}
120             ; /* 算术表达式 */
121
122 term    :   factor
123             | term TIMES factor {$$ = A_OpExp(EM_tokPos, A_timesOp, $1, $3);}
124             | term DIVIDE factor {$$ = A_OpExp(EM_tokPos, A_divideOp, $1, $3);}
125             ; /* 项 */
126
127 factor  :   ID {$$ = A_VarExp(EM_tokPos, A_Id(EM_tokPos, $1));}
128             | constant
129             | LPAREN aritexp RPAREN {$$ = $2;}
130             ; /* 因式 */
131
132 relaexp :   aritexp LT aritexp {$$ = A_BExp(EM_tokPos, A_ltOp, $1, $3);}
133             | aritexp LE aritexp {$$ = A_BExp(EM_tokPos, A_leOp, $1, $3);}
134             | aritexp GT aritexp {$$ = A_BExp(EM_tokPos, A_gtOp, $1, $3);}
135             | aritexp GE aritexp {$$ = A_BExp(EM_tokPos, A_geOp, $1, $3);}
136             | aritexp EQ aritexp {$$ = A_BExp(EM_tokPos, A_eqOp, $1, $3);}
137             | aritexp NEQ aritexp {$$ = A_BExp(EM_tokPos, A_neqOp, $1, $3);}
138             ; /* 关系表达式 */
139
140 constant:   INT {$$ = A_IntExp(EM_tokPos, $1);}
141             | FLOAT {$$ = A_RealExp(EM_tokPos, $1);}
142             ;
```

- 这一部分实际上是实验四的文法规则后加上语义动作。具体的语义动作需要参考 ast.h 和 ast.c 源程序的实现。

实验结果

1. 使用 makefile 文件编译后对 test1.p 文件测试（样例给的 makefile 文件中是对 test0.p 文件测试，在此对 makefile 的测试文件语句进行了修改。）结果如下图所示，显示分析开始→分析成功→分析树→分析结束


/cygdrive/e/编译原理2016年秋季学期/课程设计/55
LittleSec@DESKTOP-5CJHD1O /cygdrive/e/编译原理2016年秋季学期/课程设计/55
$ make test00
./parser1.exe testcases/test1.p
Parsing begin: testcases/test1.p
Parse Successful!
Prog(
ProgName(test)
decList(
varDec(
TType(INTEGER),
varList(
VAR(i),
varList(
VAR(j),
varList(
VAR(k),
varList()))))),
decList(
varDec(
TType(REAL),
varList(
VAR(f0),
varList()))),
decList())),
stmList(
assignStm(
VAR(i),
intExp(1)),
stmList(
assignStm(
VAR(j),
intExp(1)),
stmList(
assignStm(
VAR(k),
intExp(0)),
stmList(
assignStm(
VAR(f0),
realExp(3.200000)),
stmList(
whileStm(
BExp(
LESSEQ,
varExp(
VAR(k)),
intExp(100)),
seqStm(
stmList(
ifStm(
BExp(
LESSTHAN,
varExp(
VAR(j)),

```
            intExp(20)),
           seqStm(
            stmList(
             assignStm(
              VAR(j),
              varExp(
               VAR(i))),
             stmList(
              assignStm(
               VAR(k),
               opExp(
                PLUS,
                varExp(
                 VAR(k)),
                intExp(1))),
              stmList(
               assignStm(
                VAR(f0),
                opExp(
                 TIMES,
                 varExp(
                  VAR(f0)),
                 realExp(0.200000))),
               stmList()))),
           seqStm(
            stmList(
             assignStm(
              VAR(j),
              varExp(
               VAR(k))),
             stmList(
              assignStm(
               VAR(k),
               opExp(
                MINUS,
                varExp(
                 VAR(k)),
                intExp(2))),
              stmList(
               assignStm(
                VAR(f0),
                opExp(
                 DIVIDE,
                 varExp(
                  VAR(f0)),
                 realExp(0.200000))),
               stmList()))))),
          stmList()))
,
       stmList()))))))
Parsing end: testcases/test1.p

LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/55
$
```

2. 下图是分析出错的情况，对测试文件 test1.p 文件修改，把第一个 BEGIN 关键字去掉。同样的给出错误提示和位置，此时分析树只用 Prog()，这是在文法规则中定义的程序开始的地方，因而只要对输入文件分析，必然会出现该行。

```
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/55
$ make test00
./parser1.exe testcases/test1.p

Parsing begin: testcases/test1.p
testcases/test1.p:5.5: syntax error
Prog()
Parsing end: testcases/test1.p
```

五、 心得与体会

1. 进一步掌握语法分析的作用。
2. 分析树有利于直观地看到语法分析的过程。
3. 实验前应当仔细阅读 ast.h 头文件，了解给出的抽象语法树（分析树）的数据结构，尤其是各个结点和构造函数。
4. 实际上对 ast.c 源程序的补充并不难，难在补充时需要清除该构造函数中用到的结点的数据结构，由于 ast.h 头文件中定义的数据结构中多次用到了联合体和枚举变量，这些是平时编程课上不常用的，所以需要经常查看，ast.h 头文件，确保赋值语句的左值不出错。
5. 同时结点和构造函数返回值用到的数据类型是相同的，但是名字可能不一致，这就需要实验室耐心地翻看 ast.h 头文件。
6. 可以通过 cygwin 的报错来帮助自己查出用错构造函数的地方，因为在写语义动作时，就是在调用函数，如果数据类型不一致，对于 C 语言，如果是指针类型则会抛出存在隐式转换类型的警告，这时候就可以判断自己写的语义动作是否用错了函数。
7. 在写语义动作以及补充构造函数时，数据类型较多，过程比较繁琐，应当沉下心，仔细一点。