

词法分析、语法分析程序实验

郑戈涵17338233

实验目的

扩充已有的样例语言TINY，为扩展TINY语言TINY+构造词法分析和语法分析程序，从而掌握词法分析和语法分析程序的构造方法

实验内容

了解样例语言TINY及TINY编译器的实现，了解扩展TINY语言TINY+，用EBNF描述TINY+的语法，用C语言扩展TINY的词法分析和语法分析程序，构造TINY+的语法分析器。

实验要求

将TINY+源程序翻译成对应的TOKEN序列，并能检查一定的词法错误。将TOKEN序列转换成语法分析树，并能检查一定的语法错误。

实验步骤

扩充语言

Tiny语言参考<https://jlu.myweb.cs.uwindsor.ca/214/language.htm>。再其上增加了几个功能：

1. while循环语句
2. 使用and 和 or的表达式
3. bool表达式增加大于小于，大于等于，小于等于的情况
4. 支持负整数和负实数

增加后变化的产生式如下：

```
1 Statement -> Block
2     | LocalVarDecl
3     | AssignStmt
4     | ReturnStmt
5     | IfStmt
6     | WriteStmt
7     | ReadStmt
8     | WhileStmt
9     ;
10 WhileStmt -> WHILE '(' BoolExpression ')' Statement
11     ;
12 BoolExpression -> Expression Eq Expression
13     | Expression Ne Expression
14     | Expression '>' Expression
15     | Expression '<' Expression
16     | Expression Le Expression
17     | Expression Ge Expression
18     | BoolExpression And BoolExpression
19     | BoolExpression Or BoolExpression
20     ;
```

本次实验使用了linux下的flex和bison。

语法树定义

语法树是与分析部分无关的内容，在语法分析时作为副作用顺便生成语法树。包含三个功能，创建，遍历和删除。由于语法树是多叉树，而且分支个数不确定，因此使用二叉树来实现。具体做法是同一层次的其他叶节点按顺序依次生成右叶节点。下一层的叶节点生成在当前节点的左叶节点。定义如下：

```
1 struct AST
2 {
3     int lineNo;
4     char* name;
5     struct AST *leftNode;
6     struct AST *rightNode;
7     union
8     {
9         char* idName;
10        int intVal;
11        float floatVal;
12    };
13 };
```

标识符这样的语法单元在打印时需要打印具体值，因此使用union保存。

创建语法树节点

由于分支个数不确定，我使用了变长参数的函数。

```
1 struct AST *createASTNode(char *name, int num, ...)
2 {
3     va_list valist;
4     struct AST *root = (struct AST *)malloc(sizeof(struct AST));
5     root->idName=NULL;
6     root->leftNode=NULL;
7     root->rightNode=NULL;
8     //examine if alloc failed
9     if (!root)
10    {
11        yyerror("RUN OUT OF MEMORY.");
12        exit(0);
13    }
14    root->name = name;
15    // init variable args
16    va_start(valist, num);
17    if (num > 0)
18    {
19        // get next argument
20        struct AST *child = va_arg(valist, struct AST *);
21        root->leftNode = child;
22        // synthesize lineNo property from child
23        root->lineNo = child->lineNo;
24        if (num > 1)
25        {
26            for (int i = 0; i < num - 1; ++i)
27            {
```

```

28         // generate other branch as child for tree of same level(use
binary tree to simulate tree of multiple branch)
29         child->rightNode = va_arg(valist, struct AST *);
30         child = child->rightNode;
31     }
32 }
33 }
34 else
35 {
36     int arg = va_arg(valist, int);
37     root->lineNo = arg;
38     /* 3 case of terminal:
39     *   string
40     *   int
41     *   float
42     */
43     if (!strcmp(root->name, "Identifier") || !strcmp(root->name,
"StringConstant"))
44     {
45         root->idName = (char *)malloc(sizeof(char) * BUFFSIZE);
46         strcpy(root->idName, yytext);
47     }
48     else if (!strcmp(root->name, "IntConstant"))
49         root->intVal = atoi(yytext);
50     else if (!strcmp(root->name, "RealConstant"))
51         root->floatVal = atof(yytext);
52 }
53 return root;
54 }

```

遍历

遍历我使用的是扁平化的显示方法，同一层次的节点都对应同样大小的缩进，使用前序遍历。

```

1 void traverse(struct AST *root, int level)
2 {
3     if (!root)
4         return;
5     for (int i = 0; i < level; ++i)
6         printf("\t");
7     if (root->lineNo != -1)
8     {
9         printf("%s ", root->name);
10        if (!strcmp(root->name, "Identifier"))
11            printf(":%s ", root->idName);
12        else if (!strcmp(root->name, "StringConstant"))
13            printf(":%s", root->idName);
14        else if (!strcmp(root->name, "IntConstant"))
15            printf(":%d", root->intVal);
16        else if (!strcmp(root->name, "Real"))
17            printf(":%f", root->floatVal);
18        else
19            printf("( %d)", root->lineNo);
20    }
21    printf("\n");
22    traverse(root->leftNode, level + 1);
23    traverse(root->rightNode, level);

```

删除

使用后序遍历。

```
1 void destroy(struct AST *root){
2     if(!root)return;
3     if(root->leftNode)
4         destroy(root->leftNode);
5     if(root->rightNode)
6         destroy(root->rightNode);
7     free(root);
```

语法分析

虽然原理上是先词法分析再语法分析，但是实际上是从产生式才知道需要哪些token，所以先做语法分析。

将所有产生式转换为对应的语法，并添加生成语法树的语义动作。调用上面实现的创建函数即可。产生式的左端是非终端符号。右端出现的终端符号都需要定义为token，**无论是终端符号还是非终端符号都需要声明为语法树节点类型**。错误处理可以使用error语法，在对应的语法下方增加一个产生式，用yyerror输出错误信息。

syn_tree.y代码如下，%left代表运算符的优先级，实际上出现冲突时bison也会自动选择一种处理方式：

```
1  %{
2  #include<stdio.h>
3  #include "syn_tree.h"
4  %}
5  %union{
6  struct AST* a;
7  }
8
9  %token <a> Le Ge Eq Ne Def And Or IntConstant RealConstant REAL
    StringConstant Identifier Void INT WHILE If Else Return Operator
    BEGIN_KEY END_KEY MAIN WRITE READ
10 %type <a> Programs Program MethodDecl FormalParams FormalParam Block
    Statements Statement LocalVarDecl
11 Type AssignStmt ReturnStmt IfStmt WriteStmt ReadStmt BoolExpression
    Expression MultiplicativeExpr
12 PrimaryExpr ActualParams WhileStmt
13
14 %left '+' '-'
15 %left '*' '/'
16 %left And Or
17
18 %%
19 Programs :
20     Program{$$=createASTNode("Programs",1,$1);printf("Syntax tree
    :\n");traverse($$,0);destroy($$);}
21     ;
22 Program : MethodDecl {$$=createASTNode("Program",1,$1);}
23     |MethodDecl Program {$$=createASTNode("Program",2,$1,$2);}
24     ;
25
```

```

26 MethodDecl : Type Identifier '(' FormalParams ')'
Block{$$=createASTNode("MethodDecl",4,$1,$2,$4,$6);}
27 | Type MAIN Identifier '(' FormalParams ')'
Block{$$=createASTNode("MethodDecl",5,$1,$2,$3,$5,$7);}
28 ;
29 FormalParams : FormalParams ','
FormalParam{$$=createASTNode("FormalParams",2,$1,$3);}
30 | FormalParam{$$=createASTNode("FormalParams",1,$1);}
31 | /* empty */{$$=createASTNode("FormalParams",0,-1);}
32 ;
33 FormalParam : Type Identifier{$$=createASTNode("FormalParam",2,$1,$2);}
34 ;
35
36
37 Block : BEGIN_KEY Statements END_KEY
{$$=createASTNode("Block",3,$1,$2,$3);}
38 | error { yyerror("keyword typo? \n"); }
39 ;
40
41 Statements :Statements Statement{$$=createASTNode("Statements",2,$1,$2);}
42 | Statement{$$=createASTNode("Statements",1,$1);}
43 ;
44
45 Statement : Block{$$=createASTNode("Statement",1,$1);}
46 | LocalVarDecl{$$=createASTNode("Statement",1,$1);}
47 | AssignStmt{$$=createASTNode("Statement",1,$1);}
48 | ReturnStmt{$$=createASTNode("Statement",1,$1);}
49 | IfStmt{$$=createASTNode("Statement",1,$1);}
50 | WriteStmt{$$=createASTNode("Statement",1,$1);}
51 | ReadStmt{$$=createASTNode("Statement",1,$1);}
52 | WhileStmt{$$=createASTNode("Statement",1,$1);}
53 ;
54
55
56
57 LocalVarDecl : Type Identifier ';'
{$$=createASTNode("LocalVarDecl",2,$1,$2);}
58 | Type AssignStmt {$$=createASTNode("LocalVarDecl",2,$1,$2);}
59 | Type error ';' { yyerror("Maybe missing Identifier? \n"); }
60 ;
61
62 WhileStmt : WHILE '(' BoolExpression ')'
Statement{$$=createASTNode("WhileStmt",3,$1,$3,$5);}
63 ;
64
65
66 Type : INT{$$=createASTNode("Type",1,$1);}
67 | REAL{$$=createASTNode("Type",1,$1);}
68 | StringConstant{$$=createASTNode("Type",1,$1);}
69 ;
70
71 AssignStmt : Identifier Def Expression
';'{$$=createASTNode("AssignStmt",3,$1,$2,$3);}
72 | Identifier Def StringConstant
';'{$$=createASTNode("AssignStmt",3,$1,$2,$3);}
73 | error ';' { yyerror("Maybe missing ';'? \n"); }
74 ;
75 ReturnStmt : Return Expression ';'{$$=createASTNode("ReturnStmt",2,$1,$2);}

```

```

76 ;
77 IfStmt : If '(' BoolExpression ')'
Statement{$$=createASTNode("IfStmt",3,$1,$3,$5);}
78 | If '(' BoolExpression ')' Statement Else
Statement{$$=createASTNode("IfStmt",5,$1,$3,$5,$6,$7);}
79 ;
80 WriteStmt : WRITE '(' Expression ',' StringConstant ')'
';'{$$=createASTNode("WriteStmt",3,$1,$3,$5);}
81 | WRITE error ';' { yyerror("Maybe missing Identifier or
StringConstant? \n"); }
82 ;
83 ReadStmt : READ '(' Identifier ',' StringConstant ')'
';'{$$=createASTNode("ReadStmt",3,$1,$3,$5);}
84 | READ error ';' { yyerror("Maybe missing Identifier or StringConstant?
\n"); }
85 ;
86
87 BoolExpression : Expression Eq Expression
{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
88 | Expression Ne Expression
{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
89 | Expression '>'
Expression{$$=createASTNode("BoolExpression",3,$1,createASTNode("Operator(>
)",0,yylineno),$3);}
90 | Expression '<'
Expression{$$=createASTNode("BoolExpression",3,$1,createASTNode("Operator(<
)",0,yylineno),$3);}
91 | Expression Le
Expression{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
92 | Expression Ge
Expression{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
93 | BoolExpression And
BoolExpression{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
94 | BoolExpression Or
BoolExpression{$$=createASTNode("BoolExpression",3,$1,$2,$3);}
95 ;
96
97 Expression : Expression '+' MultiplicativeExpr
{$$=createASTNode("Expression",3,$1,createASTNode("Operator(+)
",0,yylineno),$3);}
98 | Expression '-'
MultiplicativeExpr{$$=createASTNode("Expression",3,$1,createASTNode("Operat
or(-)
",0,yylineno),$3);}
99 | MultiplicativeExpr{$$=createASTNode("Expression",1,$1);}
100 | error ';' { yyerror("Maybe missing ';' or operand? \n"); }
101 ;
102 MultiplicativeExpr : MultiplicativeExpr '*'
PrimaryExpr{$$=createASTNode("MultiplicativeExpr",3,$1,createASTNode("Opera
tor(*)
",0,yylineno),$3);}
103 | MultiplicativeExpr '/'
PrimaryExpr{$$=createASTNode("MultiplicativeExpr",3,$1,createASTNode("Oper
ator(/)
",0,yylineno),$3);}
104 | PrimaryExpr{$$=createASTNode("MultiplicativeExpr",1,$1);}
105 ;
106 PrimaryExpr : IntConstant {$$=createASTNode("PrimaryExpr",1,$1);}
107 | RealConstant {$$=createASTNode("PrimaryExpr",1,$1);}
108 | Identifier {$$=createASTNode("PrimaryExpr",1,$1);}
109 | '(' Expression ')' {$$=createASTNode("PrimaryExpr",1,$2);}

```

```

110 | Identifier '(' ActualParams
    | '$$=createASTNode("PrimaryExpr",2,$1,$3);}
111 ;
112 ActualParams : ActualParams ','
    Expression{$$=createASTNode("ActualParams",2,$1,$3);}
113 | Expression{$$=createASTNode("ActualParams",1,$1);}
114 ;
115 %%

```

词法分析

词法分析用到的token都在语法分析时决定了，因此将其声明在规则中，在后面追加一个语法块决定捕获到token时执行的动作，对于生成语法树来说需要生成叶节点供语法分析时使用。对于单纯的词法分析只需要返回token枚举。lexer.l代码如下：

```

1  %{
2  #include "syn_tree.h"
3  #include "syn_tree.tab.h"
4  int cur_line_num = 1;
5  void lex_error(char* msg, int line);
6  %}
7  %option yylineno
8
9  POSITIVE          ([0-9]+)
10 INT               [+~]?{POSITIVE}
11 REAL              {INT}.{POSITIVE}
12 STRING            (\042[^\042\n]*\042)
13 OPERATOR          ([+*~/%=,;!<>(){}])
14 SINGLE_COMMENT1   ("//[^\n]*")
15 SINGLE_COMMENT2   (\/\*(?:[^\n]*|\/\n)*\/)
16 Identifier        ([_a-zA-Z][_a-zA-Z0-9]*)
17
18 %%
19
20 [\n]              { cur_line_num++;
21                      }
22 [ \t\r\n]+        { /* ignore all spaces */
23                      }
24 {SINGLE_COMMENT1}   { /* skip for single line comment */
25                      }
26 {SINGLE_COMMENT2}   { /* skip for single line comment */
27                      }
28
29 {OPERATOR}         { yylval.a=createASTNode("OP",0,yylineno);return
yytext[0];           }
30
31 "<="              { yylval.a=createASTNode("Le",0,yylineno);return Le;
32                      }
33 ">="              { yylval.a=createASTNode("Ge",0,yylineno);return Ge;
34                      }
35 "=="              { yylval.a=createASTNode("Def",0,yylineno);return Def;
36                      }
37 "!="              { yylval.a=createASTNode("Eq",0,yylineno);return Eq;
38                      }
39 "!="              { yylval.a=createASTNode("Ne",0,yylineno);return Ne;
40                      }

```

```

32  "&&"          { yylval.a=createASTNode("And",0,yylineno);return And;
                      }
33  "||"          { yylval.a=createASTNode("Or",0,yylineno);return Or;
                      }
34  "INT"         { yylval.a=createASTNode("INT",0,yylineno);return INT;
                      }
35  "REAL"        { yylval.a=createASTNode("REAL",0,yylineno);return REAL;
                      }
36  "WHILE"       { yylval.a=createASTNode("WHILE",0,yylineno);return
WHILE;              }
37  "IF"          { yylval.a=createASTNode("If",0,yylineno);return If;
                      }
38  "ELSE"        { yylval.a=createASTNode("Else",0,yylineno);return Else;
                      }
39  "RETURN"      { yylval.a=createASTNode("Return",0,yylineno);return
Return;              }
40  "WRITE"       { yylval.a=createASTNode("WRITE",0,yylineno);return
WRITE;               }
41  "READ"        { yylval.a=createASTNode("READ",0,yylineno);return READ;
                      }
42  "BEGIN"       { yylval.a=createASTNode("BEGIN",0,yylineno);return
BEGIN_KEY;           }
43  "END"         { yylval.a=createASTNode("END",0,yylineno);return
END_KEY;             }
44  "MAIN"        { yylval.a=createASTNode("MAIN",0,yylineno);return MAIN;
                      }
45  {INT}         { yylval.a=createASTNode("IntConstant",0,yylineno);
return IntConstant;  }
46  {REAL}        {
yylval.a=createASTNode("RealConstant",0,yylineno);return RealConstant;  }
47  {STRING}      {
yylval.a=createASTNode("StringConstant",0,yylineno);return StringConstant;}
48  {Identifier}  { yylval.a=createASTNode("Identifier",0,yylineno);return
Identifier;          }
49
50  <<EOF>>      { return 0;
                      }
51
52  .             { lex_error("Unrecognized content", cur_line_num);
                      }
53
54  %%
55
56  void lex_error(char* msg, int line) {
57      printf("\nError at line %-3d: %s\n\n", line, msg);
58  }
59
60  int yywrap(void) {
61      return 1;
62  }

```

生成代码使用makefile，如下：

```

1  CC=gcc
2  TARGET= main
3  LEXER= lex
4  TESTFILE=test.cpp

```



```

5 all: $(TARGET) lex
6
7
8 lex_test: lex
9     ./$$(LEXER) < $(TESTFILE)
10
11 $$(LEXER): lex_only.yy.c main.c
12     $$(CC) -o $$@ $$^ -DLEX
13
14 syn_test: $(TARGET) $(TESTFILE)
15     ./$$(TARGET) < $(TESTFILE)
16
17 $$(TARGET): main.c syn_tree.tab.c lex.yy.c syn_tree.c
18
19 syn_tree.tab.c: syn_tree.y
20     bison -d $$<
21
22 lex_only.yy.c: lex_only.l
23     flex -o $$@ $$<
24
25 lex.yy.c: lexer.l
26     flex $$<
27
28 check:$(TARGET)
29     valgrind --leak-check=full \
30         --show-leak-kinds=all \
31         --track-origins=yes \
32         --verbose \
33         --log-file=valgrind-out.txt \
34         ./$$(TARGET)<$(TESTFILE)
35
36 clean:
37     rm valgrind-out.txt *.log *.tab.c *.yy.c *.tab.h $$(LEXER) $$(TARGET) 2>
38     /dev/null || echo > /dev/null

```

实验结果

使用网站提供的测试代码，修改11行加入新特性。

```

1  /** this is a comment line in the sample program */
2  INT f2(INT x, INT y )
3  BEGIN
4      INT z;
5      z := x*x - y*y;
6      RETURN z;
7  END
8  INT MAIN f1()
9  BEGIN
10     INT x;
11     WHILE(x<3 && y>3) READ(x, "A41.input");
12     REAL y:=-2112.0;
13     READ(y, "A42.input");
14     INT z;
15     z := f2(x,y) + f2(y,x);
16     WRITE (z, "A4.output");
17 END

```

词法分析

使用下面的命令进行词法分析,

```
1 | make lex_test > test.log
```

test.log中的结果如下:

```
1 flex -o lex_only.yy.c lex_only.l
2 gcc -o lex lex_only.yy.c main.c -DLEX
3 ./lex < test.cpp
4 (INT,INT)
5 (Identifier,f2)
6 ((,())
7 (INT,INT)
8 (Identifier,x)
9 (,,)
10 (INT,INT)
11 (Identifier,y)
12 (,))
13 (BEGIN_KEY,BEGIN)
14 (INT,INT)
15 (Identifier,z)
16 (;,;)
17 (Identifier,z)
18 (Def,:=)
19 (Identifier,x)
20 (*,*)
21 (Identifier,x)
22 (-,-)
23 (Identifier,y)
24 (*,*)
25 (Identifier,y)
26 (;,;)
27 (Return,RETURN)
28 (Identifier,z)
29 (;,;)
30 (END_KEY,END)
31 (INT,INT)
32 (MAIN,MAIN)
33 (Identifier,f1)
34 ((,())
35 (,))
36 (BEGIN_KEY,BEGIN)
37 (INT,INT)
38 (Identifier,x)
39 (;,;)
40 (WHILE,WHILE)
41 ((,())
42 (Identifier,x)
43 (<,<)
44 (IntConstant,3)
45 (And,&&)
46 (Identifier,y)
47 (>,>)
```

```
48 (IntConstant,3)
49 (,))
50 (READ,READ)
51 ((,(
52 (Identifier,x)
53 (,,,
54 (StringConstant,"A41.input")
55 (,))
56 (;,;)
57 (REAL,REAL)
58 (Identifier,y)
59 (Def,:=)
60 (RealConstant,-2112.0)
61 (;,;)
62 (READ,READ)
63 ((,(
64 (Identifier,y)
65 (,,,
66 (StringConstant,"A42.input")
67 (,))
68 (;,;)
69 (INT,INT)
70 (Identifier,z)
71 (;,;)
72 (Identifier,z)
73 (Def,:=)
74 (Identifier,f2)
75 ((,(
76 (Identifier,x)
77 (,,,
78 (Identifier,y)
79 (,))
80 (+,+)
81 (Identifier,f2)
82 ((,(
83 (Identifier,y)
84 (,,,
85 (Identifier,x)
86 (,))
87 (;,;)
88 (WRITE,WRITE)
89 ((,(
90 (Identifier,z)
91 (,,,
92 (StringConstant,"A4.output")
93 (,))
94 (;,;)
95 (END_KEY,END)
```

语法分析

使用下面的命令进行语法分析,

```
1 | make syn_test >test.log
```

test.log中的结果如下:

```

1 flex lexer.l
2 gcc      main.c syn_tree.tab.c lex.yy.c syn_tree.c  -o main
3 ./main < test.cpp
4 >Syntax tree :
5 Programs (2)
6   Program (2)
7     MethodDecl (2)
8       Type (2)
9         INT (2)
10      Identifier :f2
11     FormalParams (2)
12       FormalParams (2)
13         FormalParam (2)
14           Type (2)
15             INT (2)
16           Identifier :x
17         FormalParam (2)
18           Type (2)
19             INT (2)
20           Identifier :y
21     Block (3)
22       BEGIN (3)
23       Statements (4)
24         Statements (4)
25           Statements (4)
26             Statement (4)
27               LocalVarDecl (4)
28                 Type (4)
29                   INT (4)
30                 Identifier :z
31             Statement (5)
32               AssignStmt (5)
33                 Identifier :z
34               Def (5)
35                 Expression (5)
36                   Expression (5)
37                     MultiplicativeExpr (5)
38                       MultiplicativeExpr (5)
39                         PrimaryExpr (5)
40                           Identifier :x
41                         Operator(*) (5)
42                         PrimaryExpr (5)
43                           Identifier :x
44                         Operator(-) (5)
45                         MultiplicativeExpr (5)
46                           MultiplicativeExpr (5)
47                             PrimaryExpr (5)
48                               Identifier :y
49                             Operator(*) (5)
50                             PrimaryExpr (5)
51                               Identifier :y
52             Statement (6)
53               ReturnStmt (6)
54                 Return (6)
55                 Expression (6)
56                   MultiplicativeExpr (6)
57                     PrimaryExpr (6)

```

```

58 Identifier :z
59 END (7)
60 Program (8)
61 MethodDecl (8)
62 Type (8)
63 INT (8)
64 MAIN (8)
65 Identifier :f1
66
67 Block (9)
68 BEGIN (9)
69 Statements (10)
70 Statements (10)
71 Statements (10)
72 Statements (10)
73 Statements (10)
74 Statements (10)
75 Statements (10)
76 Statement (10)
77 LocalVarDecl (10)
78 Type (10)
79 INT (10)
80 AssignStmt (10)
81 Identifier :x
82 Def (10)
83 Expression (10)
84 MultiplicativeExpr (10)
85 PrimaryExpr (10)
86 RealConstant :-333.329987
87 Statement (11)
88 whilestmt (11)
89 WHILE (11)
90 BoolExpression (11)
91 BoolExpression (11)
92 Expression (11)
93 MultiplicativeExpr (11)
94 PrimaryExpr (11)
95 Identifier :x
96 Operator(<) (11)
97 Expression (11)
98 MultiplicativeExpr (11)
99 PrimaryExpr (11)
100 IntConstant :3
101 And (11)
102 BoolExpression (11)
103 Expression (11)
104 MultiplicativeExpr (11)
105 PrimaryExpr (11)
106 Identifier :y
107 Operator(>) (11)
108 Expression (11)
109 MultiplicativeExpr (11)
110 PrimaryExpr (11)
111 IntConstant :3
112 Statement (11)
113 ReadStmt (11)
114 READ (11)
115 Identifier :x

```

```

116             StringConstant : "A41.input"
117         Statement (12)
118             LocalVarDecl (12)
119                 Type (12)
120                     REAL (12)
121             AssignStmt (12)
122                 Identifier : y
123                 Def (12)
124                 Expression (12)
125                     MultiplicativeExpr (12)
126                         PrimaryExpr (12)
127                             IntConstant : 2112
128         Statement (13)
129             ReadStmt (13)
130                 READ (13)
131                 Identifier : y
132                 StringConstant : "A42.input"
133     Statement (14)
134         LocalVarDecl (14)
135             Type (14)
136                 INT (14)
137             Identifier : z
138 Statement (15)
139     AssignStmt (15)
140         Identifier : z
141         Def (15)
142         Expression (15)
143             Expression (15)
144                 MultiplicativeExpr (15)
145                     PrimaryExpr (15)
146                         Identifier : f2
147                         ActualParams (15)
148                             ActualParams (15)
149                                 Expression (15)
150                                     MultiplicativeExpr (15)
151                                         PrimaryExpr (15)
152                                             Identifier : x
153                                     Expression (15)
154                                         MultiplicativeExpr (15)
155                                             PrimaryExpr (15)
156                                                 Identifier : y
157                                     Operator(+) (15)
158                                         MultiplicativeExpr (15)
159                                             PrimaryExpr (15)
160                                                 Identifier : f2
161                                                 ActualParams (15)
162                                                     ActualParams (15)
163                                                         Expression (15)
164                                                             MultiplicativeExpr (15)
165                                                                 PrimaryExpr (15)
166                                                                     Identifier : y
167                                                             Expression (15)
168                                                                 MultiplicativeExpr (15)
169                                                                    PrimaryExpr (15)
170                                                                        Identifier : x
171 Statement (16)
172     WriteStmt (16)
173         WRITE (16)

```

```

174         Expression (16)
175             MultiplicativeExpr (16)
176                 PrimaryExpr (16)
177                     Identifier :z
178                 StringConstant : "A4.output"
179         END (17)
180

```

错误处理

词法错误处理

使用下面的代码进行错误处理测试，错误为：

1. 第10行的#3为非法串
2. 第13行漏了引号

```

1  /** this is a comment line in the sample program */
2  INT f2(INT x, INT y )
3  BEGIN
4      INT z;
5      z := x*x - y*y;
6      RETURN z;
7  END
8  INT MAIN f1()
9  BEGIN
10     INT #3;
11     WHILE(x<3 && y>3) READ(x, "A41.input");
12     INT y:=-2112.0;
13     READ(y, "A42.input");
14     INT z;
15     z := f2(x,y) + f2(y,x);
16     WRITE (z, "A4.output");
17 END

```

使用下面的命令进行测试

```
1 | make lex_test > test.log
```

test.log中的结果如下：

```

1  flex -o lex_only.yy.c lex_only.l
2  gcc -o lex lex_only.yy.c main.c -DLEX
3  ./lex < test.cpp
4  (INT,INT)
5  (Identifier,f2)
6  ((,())
7  (INT,INT)
8  (Identifier,x)
9  (,,, )
10 (INT,INT)
11 (Identifier,y)
12 (,))
13 (BEGIN_KEY,BEGIN)
14 (INT,INT)
15 (Identifier,z)

```

```

16  (;;;)
17  (Identifier,z)
18  (Def,:=)
19  (Identifier,x)
20  (*,*)
21  (Identifier,x)
22  (-,-)
23  (Identifier,y)
24  (*,*)
25  (Identifier,y)
26  (;;;)
27  (Return,RETURN)
28  (Identifier,z)
29  (;;;)
30  (END_KEY,END)
31  (INT,INT)
32  (MAIN,MAIN)
33  (Identifier,f1)
34  ((,()
35  (,))
36  (BEGIN_KEY,BEGIN)
37  (INT,INT)
38
39  Error at line 10 : Unrecognized content
40
41  (IntConstant,3)
42  (;;;)
43  (WHILE,WHILE)
44  ((,()
45  (Identifier,x)
46  (<,<)
47  (IntConstant,3)
48  (And,&&)
49  (Identifier,y)
50  (>,>)
51  (IntConstant,3)
52  (,))
53  (READ,READ)
54  ((,()
55  (Identifier,x)
56  (,,, )
57  (StringConstant,"A41.input")
58  (,))
59  (;;;)
60  (INT,INT)
61  (Identifier,y)
62  (Def,:=)
63  (RealConstant,-2112.0)
64  (;;;)
65  (READ,READ)
66  ((,()
67  (Identifier,y)
68  (,,, )
69
70  Error at line 13 : Unrecognized content
71
72  (Identifier,A42)
73  (.,.)

```



```

74 (Identifier,input)
75 (,))
76 (;;;)
77 (INT,INT)
78 (Identifier,z)
79 (;;;)
80 (Identifier,z)
81 (Def,:=)
82 (Identifier,f2)
83 ((,()
84 (Identifier,x)
85 (,,, )
86 (Identifier,y)
87 (,))
88 (+,+)
89 (Identifier,f2)
90 ((,()
91 (Identifier,y)
92 (,,, )
93 (Identifier,x)
94 (,))
95 (;;;)
96 (WRITE,WRITE)
97 ((,()
98 (Identifier,z)
99 (,,, )
100
101 Error at line 16 : Unrecognized content
102
103 (Identifier,A4)
104 (.,.)
105 (Identifier,output)
106 (,))
107 (;;;)
108 (END_KEY,END)
109

```

可以发现，前面出现词法错误后，后面的词法分析结果也接连出现错误。

语法错误处理

在原来的代码中增加几个错误：

1. 11行没有第二个参数，即READ函数对应的文件的标识符。
2. 14行缺少变量名
3. 16行缺少分号

```

1  /** this is a comment line in the sample program */
2  INT f2(INT x, INT y )
3  BEGIN
4      INT z;
5      z := x*x - y*y;
6      RETURN z;
7  END
8  INT MAIN f1()
9  BEGIN
10     INT x;

```

```

11     WHILE(x<3 && y>3) READ(x, );
12     INT y;
13     READ(y, "A42.input");
14     INT ;
15     z := f2(x,y) + f2(y,x);
16     WRITE (z, "A4.output")
17     END

```

使用下面的命令进行测试

```
1 | make syn_test > test.log
```

测试结果如下:

```

→ syntax_tree git:(master) X make syn_test >test.log
line 11: syntax error
line 11: Maybe missing Identifier or StringConstant?

line 14: syntax error
line 14: Maybe missing Identifier?

line 16: syntax error
line 16: Maybe missing ';' or operand?

make: *** [makefile:15: syn_test] Error 1

```

问题

由于进行了内存分配和回收, 本次实验使用了valgrind工具检查内存问题。经过检查, 发现bison生成的代码中存在内存泄露问题。使用下面的命令进行检查:

```
1 | make check
```

查看valgrind-out.txt, 其中有如下泄露:

```

1  ==16662== HEAP SUMMARY:
2  ==16662==      in use at exit: 17,978 bytes in 41 blocks
3  ==16662==    total heap usage: 247 allocs, 210 frees, 31,258 bytes allocated
4  ==16662==
5  ==16662== Searching for pointers to 41 not-freed blocks
6  ==16662== Checked 75,944 bytes
7  ==16662==
8  ==16662== 8 bytes in 1 blocks are still reachable in loss record 1 of 4
9  ==16662==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
10 ==16662==      by 0x10CE82: yyalloc (in
11 ==16662==      by 0x10C941: yyensure_buffer_stack (in
12 ==16662==      by 0x10ACE4: yylex (in
13 ==16662==      by 0x109769: yyparse (in
14 ==16662==      by 0x10943F: main (in

```

```

15 ==16662==
16 ==16662== 64 bytes in 1 blocks are still reachable in loss record 2 of 4
17 ==16662==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
18 ==16662==    by 0x10CE82: yyalloc (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
19 ==16662==    by 0x10C4B6: yy_create_buffer (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
20 ==16662==    by 0x10AD0E: yylex (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
21 ==16662==    by 0x109769: yyparse (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
22 ==16662==    by 0x10943F: main (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
23 ==16662==
24 ==16662== 1,520 bytes in 38 blocks are definitely lost in loss record 3 of 4
25 ==16662==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
26 ==16662==    by 0x10CF83: createASTNode (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
27 ==16662==    by 0x10AFAE: yylex (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
28 ==16662==    by 0x109769: yyparse (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
29 ==16662==    by 0x10943F: main (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
30 ==16662==
31 ==16662== 16,386 bytes in 1 blocks are still reachable in loss record 4 of 4
32 ==16662==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
33 ==16662==    by 0x10CE82: yyalloc (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
34 ==16662==    by 0x10C4EB: yy_create_buffer (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
35 ==16662==    by 0x10AD0E: yylex (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
36 ==16662==    by 0x109769: yyparse (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
37 ==16662==    by 0x10943F: main (in
/mnt/d/OneDrive/workspace/c++/compiler/exp/3_lex_yacc/syntax_tree/main)
38 ==16662==
39 ==16662== LEAK SUMMARY:
40 ==16662==    definitely lost: 1,520 bytes in 38 blocks
41 ==16662==    indirectly lost: 0 bytes in 0 blocks
42 ==16662==    possibly lost: 0 bytes in 0 blocks
43 ==16662==    still reachable: 16,458 bytes in 3 blocks
44 ==16662==    suppressed: 0 bytes in 0 blocks

```

可以看到泄露的内存由yyalloc分配，这部分的代码实际上来源于bison和flex生成的代码。