# 编译原理

班级：17 计算机科学与计算机

姓名：黄吉昊　　学号：2017330300135

# 实验三　基于 YACC 的 TINY 语法分析器的构建

## 一、　实验要求

运用 YACC，针对 TINY 语言，构造一个语法分析器。给出实验方案，实施并描述结果。

## 二、　实验方案

利用教材《编译原理与实践》课后的附录 B 中的例子进行设计

输入：

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

输出：

```
TINY COMPILATION: sample.tny

Syntax tree:
  Read: x
  If
    Op: <
```

```
    Const: 0
    Id: x
  Assign to: fact
    Const: 1
  Repeat
    Assign to: fact
      Op: *
        Id: fact
        Id: x
    Assign to: x
      Op: -
        Id: x
        Const: 1
    Op: =
      Id: x
      Const: 0
  Write
    Id: fact
```

lex 和 yacc 阶段数据传递方法

**第一步 查找资料**

查找资料可知 yacc 产生的子程序在申请读入下一个单词时会调用 yylex()。yylex()返回一个单词符号，并将相关的属性值存入全局量 yylval

为了联用 lex 和 yacc，需要在运行 yacc 程序时加选项-d，以产生文件 y. tab. h，其中会包含在 yacc 描述文件中（由% tokens 定义）的所有单词种别。文件 y. tab. h 将被包含在 lex 描述文件中。

**第二步 对于 tiny.y 进行修改**

在附录所给的 tiny.y 文件进行修改，将其中的 static int yylex 方法去掉

注：我使用的是 Linux 系统

```
/* yylex calls getToken to make
 * compatible with ealier versi
 */
static int yylex(void)
{ return getToken(); }

TreeNode * parse(void)
{ yyparse();
  return savedTree;
}
```

**第三步 对于 yacc 后生成的 y.tab.c 文件进行修改**

对于其中的 yychar=yylex;修改成 yychar=getToken()

```
  if (yychar == YYEMPTY)
    {
      YYDPRINTF ((stderr, "Reading a token: "));
      yychar = YYLEX;
    }
```

就此，将使用 lex 方法读到的 token 送入 yacc 产生的分析子程序中，实现了两者的使用

## 三、 分析表 parsing table 问题

理论和设计

parsing table 在实验方案中的作用

YACC 输出的语法分析函数 int yyparse()在对一个输入文件进行分析时，如果面对某一状态或输入单词在**分析表**中找不到相应的动作，则调用函数 yyerror()报错

利用 bison -v 命令获得 parsing table

```
./tiny.out sam
Grammar

    0 $accept: program $end

    1 program: stmt_seq

    2 stmt_seq: stmt_seq SEMI stmt
    3         | stmt

    4 stmt: if_stmt
    5     | repeat_stmt
    6     | assign_stmt
    7     | read_stmt
    8     | write_stmt
    9     | error

   10 if_stmt: IF exp THEN stmt_seq END
   11        | IF exp THEN stmt_seq ELSE stmt_seq END

   12 repeat_stmt: REPEAT stmt_seq UNTIL exp

   13 @1: /* empty */

   14 assign_stmt: ID @1 ASSIGN exp

   15 read_stmt: READ ID

   16 write_stmt: WRITE exp

   17 exp: simple_exp LT simple_exp
   18    | simple_exp EQ simple_exp
   19    | simple_exp

   20 simple_exp: simple_exp PLUS term
   21           | simple_exp MINUS term
   22           | term

   23 term: term TIMES factor
   24     | term OVER factor
   25     | factor

   26 factor: LPAREN exp RPAREN
   27       | NUM
```

## 四、 内容和步骤

1、针对 TINY 语言给出 yacc 的 y 文件的代码

```
/****************************************************/
/* File: tiny.y                                     */
/* The TINY Yacc/Bison specification file           */
/* Compiler Construction: Principles and Practice   */
/* Kenneth C. Louden                                */
/****************************************************/
%{
#define YYPARSER /* distinguishes Yacc output from other code files */

#include "globals.h"
#include "util.h"
#include "scan.h"
#include "parse.h"

#define YYSTYPE TreeNode *
static char * savedName; /* for use in assignments */
```

```
static int savedLineNo;  /* ditto */
static TreeNode * savedTree; /* stores syntax tree for later return */

%}

%token IF THEN ELSE END REPEAT UNTIL READ WRITE
%token ID NUM
%token ASSIGN EQ LT PLUS MINUS TIMES OVER LPAREN RPAREN SEMI
**************************************************/
/* File: tiny.y                                   */
/* The TINY Yacc/Bison specification file         */
/* Compiler Construction: Principles and Practice  */
/* Kenneth C. Louden                              */
/************************************************/
%{
#define YYPARSER /* distinguishes Yacc output from other code files */

#include "globals.h"
#include "util.h"
#include "scan.h"
#include "parse.h"

#define YYSTYPE TreeNode *
static char * savedName; /* for use in assignments */
static int savedLineNo;  /* ditto */
static TreeNode * savedTree; /* stores syntax tree for later return */

%}

%token IF THEN ELSE END REPEAT UNTIL READ WRITE
%token ID NUM
%token ASSIGN EQ LT PLUS MINUS TIMES OVER LPAREN RPAREN SEMI
%token ERROR

%% /* Grammar for TINY */
**************************************************/
/* File: tiny.y                                   */
/* The TINY Yacc/Bison specification file         */
/* Compiler Construction: Principles and Practice  */
/* Kenneth C. Louden                              */
/************************************************/
%{
#define YYPARSER /* distinguishes Yacc output from other code files */
```

```
#include "globals.h"
#include "util.h"
#include "scan.h"
#include "parse.h"

#define YYSTYPE TreeNode *
static char * savedName; /* for use in assignments */
static int savedLineNo;  /* ditto */
static TreeNode * savedTree; /* stores syntax tree for later return */

%}

%token IF THEN ELSE END REPEAT UNTIL READ WRITE
%token ID NUM
%token ASSIGN EQ LT PLUS MINUS TIMES OVER LPAREN RPAREN SEMI
%token ERROR

%% /* Grammar for TINY */

program     : stmt_seq
```

2、给出.l 文件的代码

```
/****************************************************/
/* File: tiny.l                                     */
/* Lex specification for TINY                       */
/* Compiler Construction: Principles and Practice   */
/* Kenneth C. Louden                                */
/****************************************************/

%{
#include "globals.h"
#include "util.h"
#include "scan.h"
/* lexeme of identifier or reserved word */
char tokenString[MAXTOKENLEN+1];
%}

%option noyywrap

digit       [0-9]
number      {digit}+
letter      [a-zA-Z]
identifier  {letter}+
```

```
newline    \n
whitespace  [ \t]+

%%

"if"          {return IF;}
"then"        {return THEN;}
"else"        {return ELSE;}
"end"         {return END;}
"repeat"      {return REPEAT;}
"until"       {return UNTIL;}
"read"        {return READ;}
"write"       {return WRITE;}
":="          {return ASSIGN;}
"="           {return EQ;}
"<"           {return LT;}
"+"           {return PLUS;}
"-"           {return MINUS;}
"*"           {return TIMES;}
"/"           {return OVER;}
"("           {return LPAREN;}
")"           {return RPAREN;}
";"           {return SEMI;}
{number}      {return NUM;}
{identifier}  {return ID;}
{newline}     {lineno++;}
{whitespace}  {/* skip whitespace */}
"{"           { char c;
                do
                { c = input();
                  if (c == EOF) break;
                  if (c == '\n') lineno++;
                } while (c != '}');
              }
.             {return ERROR;}

%%

TokenType getToken(void)
{ static int firstTime = TRUE;
  TokenType currentToken;
  if (firstTime)
  { firstTime = FALSE;
    lineno++;
```

```
    yyin = source;
    yyout = listing;
  }
  currentToken = yylex();
  strncpy(tokenString,yytext,MAXTOKENLEN);
  if (TraceScan) {
    fprintf(listing,"\t%d: ",lineno);
    printToken(currentToken,tokenString);
  }
  return currentToken;
}
```

3、实验具体步骤

1. 目前所有的文件

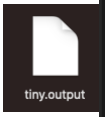
```
➜  P_yacc ls
Makefile   main.c     sample.tny tiny.l     util.c
globals.h  parse.h    scan.h     tiny.y     util.h
```

2.将 tiny.y 中的 yylex 方法去掉

```
/* yylex calls getToken to make Yacc/Bison out
 * compatible with ealier versions of the TINY
 */
static int yylex(void)
{ return getToken(); }
```

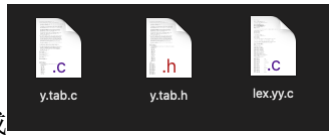
3.输入命令 bison -v tiny.y

生成 tiny.output　　LR 分析表




```
./tiny.out sam
Grammar

    0 $accept: program $end

    1 program: stmt_seq

    2 stmt_seq: stmt_seq SEMI stmt
    3         | stmt

    4 stmt: if_stmt
    5     | repeat_stmt
    6     | assign_stmt
    7     | read_stmt
    8     | write_stmt
    9     | error

   10 if_stmt: IF exp THEN stmt_seq END
   11        | IF exp THEN stmt_seq ELSE stmt_seq END

   12 repeat_stmt: REPEAT stmt_seq UNTIL exp

   13 @1: /* empty */

   14 assign_stmt: ID @1 ASSIGN exp

   15 read_stmt: READ ID

   16 write_stmt: WRITE exp

   17 exp: simple_exp LT simple_exp
   18    | simple_exp EQ simple_exp
   19    | simple_exp

   20 simple_exp: simple_exp PLUS term
   21           | simple_exp MINUS term
   22           | term

   23 term: term TIMES factor
   24     | term OVER factor
   25     | factor

   26 factor: LPAREN exp RPAREN
   27       | NUM
```

4.输入命令 lex tiny.l 和 yacc -d tiny.y

生成 

文件

5.修改 y.tab.c 将 yychar=yylex()修改成 yychar=getToken()

```
if (yychar == YYEMPTY)
  {
    YYDPRINTF ((stderr, "Reading a token: "));
    yychar = YYLEX;
  }
```

```
if (yychar == YYEMPTY)
  {
    YYDPRINTF ((stderr, "Reading a token: "));
    yychar = getToken();
  }
```

6.输入 make 命令进行构建

生成 tiny.out 可执行文件


7.以 sample.tny 作为输入执行 tiny.out 文件

```
➜  P_yacc ./tiny.out sample.tny

TINY COMPILATION: sample.tny

Syntax tree:
  Read: x
  If
    Op: <
      Const: 0
      Id: x
    Assign to: fact
      Const: 1
    Repeat
      Assign to: fact
        Op: *
          Id: fact
          Id: x
      Assign to: x
        Op: -
          Id: x
          Const: 1
      Op: =
        Id: x
        Const: 0
    Write
      Id: fact
➜  P_yacc
```

# 五、 实验结果:

1. 输出

```
➜  P_yacc ./tiny.out sample.tny

TINY COMPILATION: sample.tny

Syntax tree:
  Read: x
  If
    Op: <
      Const: 0
      Id: x
    Assign to: fact
      Const: 1
    Repeat
      Assign to: fact
        Op: *
          Id: fact
          Id: x
      Assign to: x
        Op: -
          Id: x
          Const: 1
      Op: =
        Id: x
        Const: 0
    Write
      Id: fact
➜  P_yacc
```

2、分析表

```
Grammar

    0 $accept: program $end

    1 program: stmt_seq

    2 stmt_seq: stmt_seq SEMI stmt
    3         | stmt

    4 stmt: if_stmt
    5     | repeat_stmt
    6     | assign_stmt
    7     | read_stmt
    8     | write_stmt
    9     | error

   10 if_stmt: IF exp THEN stmt_seq END
   11        | IF exp THEN stmt_seq ELSE stmt_seq END

   12 repeat_stmt: REPEAT stmt_seq UNTIL exp

   13 @1: /* empty */

   14 assign_stmt: ID @1 ASSIGN exp

   15 read_stmt: READ ID

   16 write_stmt: WRITE exp

   17 exp: simple_exp LT simple_exp
   18    | simple_exp EQ simple_exp
   19    | simple_exp

   20 simple_exp: simple_exp PLUS term
   21           | simple_exp MINUS term
   22           | term

   23 term: term TIMES factor
   24     | term OVER factor
   25     | factor

   26 factor: LPAREN exp RPAREN
   27       | NUM
```

# 六、 实验结论:

1 、实验结论

在这个实验中，通过 lex 和 yacc 工具，对于输入的内容，lex 进行词法分析，yacc 对于 lex 词法分析后的结果进行语法分析。两者相结合，通过调用 tiny.out 文件即可进行语法分析。通过 bison -v 命令，即可生成 tiny.out 分析表

2、分析和总结

（一）利用工具

利用工具的话可以在实现语法分析和词法分析的时候，只需要根据 lex 和 yacc 工具给出的格式进行编写就可以很轻松的完成词法分析和语法分析，不用考虑整个程序的流程，等琐碎的细节。

（二）手写编写程序

手写编程程序来进行词法分析和语法分析，可以较为全面的把控所有的细节，对于输入，输出，token 的定义，等等方面可以进行更加详细的设计。缺点就是，写起来比较耗时，需要完成一整个程序的编写

3、实验中出现的冲突及解决过程描述

（一）没有对 tiny.y 和 y.tab.c 中的文件进行处理直接进行 lex 和 yacc 之后 make

需要先对 tiny.y 的 yylex 方法去除

再进行 yacc -d tiny.y 生成 y.tab.c

再对 y.tab.c 中的 yychar=yylex 进行修改替换

（二）对于 yychar=yylex()的修改可能会改错位置

因为 tiny.y 中有许多个 yychar 字符，而 yychar=yylex()只有一处

需要用 vim 的:/yychar 进行一个一个的查找，避免找错

（三）tiny.y 中去掉 yylex 方法时可能没去干净

yylex 方法需要全部去除，包括函数的定义和实现内容

（四）必须在 lex 和 yacc 之后，生成 lex.yy.c 和 y.tab.c 和 y.tab.h 之后再进行 make