暨南大学本科实验报告专用纸

课程名称 编译原理 成绩评定

实验项目名称 Tiny C 语言编译程序实验二 Parser.c 指导教师 余芳

实验项目编号 03 实验地点 N116

学生姓名 温钊迪 学号 2016051487

学院 信息科学技术学院 系 计算机科学系 专业 计算机科学与技术

实验时间 2019 年 05 月 20 日

一、 实验目的

填写各语法符号的递归下降子程序，完成语法分析器 parse.c。

二、 实验要求

1) 用递归下降分析法，为每个语法符号编写子程序。进入每个子程序前已读入一个新 Token。
2) 一个语法结构的内部表示形式为语法树，数据结构是 globals.h 中的 treeNode。在做语法分析的同时建立语法结构的内部表示——语法树。

三、 源代码

（1）parser.c 文件

```
/****************************************************/
/* File: parse.c                                    */
/* The parser implementation for the TINY compiler  */
/****************************************************/

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

static TokenType token; /* holds current token */

/* function prototypes for recursive calls */
static TreeNode * stmt_sequence(void);
```

```
static TreeNode * statement(void);
static TreeNode * if_stmt(void);
static TreeNode * repeat_stmt(void);
static TreeNode * assign_stmt(void);
static TreeNode * read_stmt(void);
static TreeNode * write_stmt(void);
static TreeNode * exp(void);
static TreeNode * simple_exp(void);
static TreeNode * term(void);
static TreeNode * factor(void);

static void syntaxError(char * message)
{ fprintf(listing,"\n>>> ");
   fprintf(listing,"Syntax error at line %d: %s",lineno,message);
   Error = TRUE;
}

static void match(TokenType expected)
{ if (token == expected) token = getToken();
   else {
      syntaxError("unexpected token -> ");
      printToken(token,tokenString);
      fprintf(listing,"           ");
   }
}

TreeNode * stmt_sequence(void)
{ TreeNode * t = statement();
   TreeNode * p = t;
   while ((token!=ENDFILE) && (token!=END) &&
             (token!=ELSE) && (token!=UNTIL))
   { TreeNode * q;
      match(SEMI);
      q = statement();
      if (q!=NULL) {
         if (t==NULL) t = p = q;
         else /* now p cannot be NULL either */
         { p->sibling = q;
            p = q;
         }
      }
   }
   return t;
}
```

```
TreeNode * statement(void)
{ TreeNode * t = NULL;
   switch (token) {
      case IF : t = if_stmt(); break;
      case REPEAT : t = repeat_stmt(); break;
      case ID : t = assign_stmt(); break;
      case READ : t = read_stmt(); break;
      case WRITE : t = write_stmt(); break;
      default : syntaxError("unexpected token -> ");
                printToken(token,tokenString);
                token = getToken();
                break;
   } /* end case */
   return t;
}

TreeNode * if_stmt(void)
{
      TreeNode *t = newStmtNode(IfK);
      match(IF);
      if(t!=NULL)
      {
         t->child[0]=exp();
         match(THEN);
         t->child[1]= stmt_sequence();
         if(token==ELSE)
         {
            match(ELSE);
            t->child[2] = stmt_sequence();
         }
      }
      match(END);
   return    t;
}

TreeNode * repeat_stmt(void)
{
       TreeNode *t = newStmtNode(RepeatK);
       match(REPEAT);
       if(t!=NULL)
       {
            t->child[0] = stmt_sequence();
            match(UNTIL);
```

```
            t->child[1] = exp();
        }
    return t;
}

TreeNode * assign_stmt(void)
{
    TreeNode *t = newStmtNode(AssignK);
    if(t!=NULL && token==ID)
    {
        t->attr.name = copyString(tokenString);
    }
    match(ID);
    match(ASSIGN);
    if(t!=NULL)
    {
        t->child[0] = simple_exp();
    }
    return    t;
}

TreeNode * read_stmt(void)
{
   TreeNode *t = newStmtNode(ReadK);
    match(READ);
    if(t!=NULL && token==ID)
    {
        t->attr.name = copyString(tokenString);
    }
    match(ID);
    return    t;
}

TreeNode * write_stmt(void)
{
   TreeNode *t = newStmtNode(WriteK);
    match(WRITE);
    if(t!=NULL)
        t->child[0]=exp();
    return t;

}

TreeNode * exp(void)
```

```
{ TreeNode * t = simple_exp();
   if ((token==LT)||(token==EQ)) {
      TreeNode * p = newExpNode(OpK);
      if (p!=NULL) {
         p->child[0] = t;
         p->attr.op = token;
         t = p;
      }
      match(token);
      if (t!=NULL)
         t->child[1] = simple_exp();
   }
   return t;
}

TreeNode * simple_exp(void)
{
    TreeNode*t    = term();
     if(token==PLUS || token==MINUS)
     {
          TreeNode *p = newExpNode(OpK);
          if(p)
          {
               p->child[0]= t;
               p->attr.op=token;
               t = p;
          }
          match(token);
          if(t)
               t->child[1] = simple_exp();
          return t;
     }
}

TreeNode * term(void)
{
    TreeNode *t = factor();
     while (token==TIMES || token==OVER)
     {
          TreeNode *p = newExpNode(OpK);
          if(p)
          {
               p->child[0] = t;
               p->attr.op = token;
```

```
                t = p;
            }
            match(token);
            if(t)
                t->child[1] = simple_exp();

        }
        return t;
}

TreeNode * factor(void)
{
    TreeNode *t = NULL;
     switch (token){
            case NUM:
                t = newExpNode(ConstK);
                if(t)
                    t->attr.val= atoi(tokenString);
                match(NUM);
                break;
            case ID:
                t = newExpNode(IdK);
                if(t) t->attr.name = copyString(tokenString);
                match(ID);
                break;
            case LPAREN:
                match(LPAREN);
                t = simple_exp();
                match(RPAREN);
                break;
            default:
                syntaxError("unexpected token ->");
                printToken(token,tokenString);
                token = getToken();
                break;
        }
        return t;
}

/*************************************/
/* the primary function of the parser      */
/*************************************/
/* Function parse returns the newly
  * constructed syntax tree
```

```
 */
TreeNode * parse(void)
{
      TreeNode * t;
   token = getToken();
   t = stmt_sequence();
   if (token!=ENDFILE)
      syntaxError("Code ends before file\n");
   return t;
}
```

(2)在安装完 tcc 后

①在 windows 的 shell 下运行 tcc main.c util.c parse.c scan.c

```
PS F:\大学资料\大三下\编译原理\编译原理作业\实验\3\TinyC-master\src> tcc main.c util.c parse.c scan.c
parse.c:210: ';' expected
PS F:\大学资料\大三下\编译原理\编译原理作业\实验\3\TinyC-master\src> tcc main.c util.c parse.c scan.c
PS F:\大学资料\大三下\编译原理\编译原理作业\实验\3\TinyC-master\src> ls


    目录: F:\大学资料\大三下\编译原理\编译原理作业\实验\3\TinyC-master\src


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2019.04.24     01:35                cmake-build-debug
------        2019.04.24     01:35           3079 ANALYZE.C
------        2019.04.24     01:35            540 ANALYZE.H
------        2019.04.24     01:35           6759 cgen.c
------        2019.04.24     01:35            660 cgen.h
------        2019.04.24     01:35            101 CMakeLists.txt
------        2019.04.24     01:35           2843 GLOBALS.H
-a----        2019.05.26     22:04           2248 MAIN.C
-a----        2019.05.26     22:13           8704 main.exe
-a----        2019.05.26     22:13           5830 PARSE.C
------        2019.04.24     01:35            372 PARSE.H
------        2019.04.24     01:35            263 SAMPLE.TNY
------        2019.04.24     01:35           6348 SCAN.C
------        2019.04.24     01:35            547 SCAN.H
------        2019.04.24     01:35           2938 SYMTAB.C
------        2019.04.24     01:35            847 SYMTAB.H
------        2019.04.24     01:35           4736 UTIL.C
------        2019.04.24     01:35            925 UTIL.H
```

运行得到 main.exe

②然后运行.\main.exe .\SAMPLE.TNY

```
PS F:\大学资料\大三下\编译原理\编译原理作业\实验\3\TinyC-master\src> .\main.exe .\SAMPLE.TNY

TINY COMPILATION: .\SAMPLE.TNY
        5: reserved word: read
        5: ID, name= x
        5: ;
        6: reserved word: if
        6: NUM, val= 0
        6: <
        6: ID, name= x
        6: reserved word: then
        7: ID, name= fact
        7: :=
        7: NUM, val= 1
        7: ;
        8: reserved word: repeat
        9: ID, name= fact
        9: :=
        9: ID, name= fact
        9: *
        9: ID, name= x
        9: ;
        10: ID, name= x
        10: :=
        10: ID, name= x
        10: -
        10: NUM, val= 1
        11: reserved word: until
        11: ID, name= x
        11: =
        11: NUM, val= 0
        11: ;
        12: reserved word: write
        12: ID, name= fact
        13: reserved word: end
        14: EOF
```