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 语言编译程序实验三 Analyze.c 指导教师 余芳

实验项目编号 04 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 06 月 07 日

一、 实验目的

填写符号表条目插入子程序，完成符号表生成器 symtab.c；及类型检查子程序，完成语义分析器 analyze.c。。

二、 实验要求

1) 符号表采用散列表组织，散列函数选取如下函数：

```
#define SHIFT 4
static int hash ( char * key )      /* 哈希函数 */
{ int temp = 0;
  int i = 0;
  while (key[i] != '\0')
  { temp = ((temp << SHIFT) + key[i]) % SIZE;
    ++i;
  }
  return temp;
}    // SIZE 的值是 211
```

2)符号表条目的数据结构是 symtab.c 中的 BucketList，由名字、名字在程序中出现的位置组成的链以及地址构成。

```
typedef struct LineListRec          /* 名字在程序中出现的位置组成的链 */
   { int lineno;
     struct LineListRec * next;
   } * LineList;

typedef struct BucketListRec        /* 符号表条目的结构 */
   { char * name;
```

```
    LineList lines;
    int memloc ;          /* 记录名字在程序中是第几个出现的 */
    struct BucketListRec * next;
  } * BucketList;

static BucketList hashTable[SIZE];      /* 哈希表 */
```

3)

生成符号表由 analyze.c 中的子程序 buildSymtab(TreeNode * syntaxTree) 完成，它先根遍历在语法阶段建立的语法树——traverse(syntaxTree, insertNode, nullProc)。其中 nullProc 为一个什么事也不做的子程序。这样，由语法树的树根至叶子将各名字插入到符号表中。

三、 源代码

（1）symtab.c 文件

```
/****************************************************/
/* File: symtab.c                                   */
/* Symbol table implementation for the TINY compiler*/
/* (allows only one symbol table)                   */
/* Symbol table is implemented as a chained         */
/* hash table                                       */
/****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "symtab.h"

/* SIZE is the size of the hash table */
#define SIZE 211

/* SHIFT is the power of two used as multiplier
   in hash function  */
#define SHIFT 4

/* the hash function */
static int hash ( char * key )
{ int temp = 0;
  int i = 0;
  while (key[i] != '\0')
  { temp = ((temp << SHIFT) + key[i]) % SIZE;
    ++i;
```

```
    }
    return temp;
}

/* the list of line numbers of the source
 * code in which a variable is referenced
 */
typedef struct LineListRec
     { int lineno;
       struct LineListRec * next;
     } * LineList;

/* The record in the bucket lists for
 * each variable, including name,
 * assigned memory location, and
 * the list of line numbers in which
 * it appears in the source code
 */
typedef struct BucketListRec
     { char * name;
       LineList lines;
       int memloc ; /* memory location for variable */
       struct BucketListRec * next;
     } * BucketList;

/* the hash table */
static BucketList hashTable[SIZE];

/* Procedure st_insert inserts line numbers and
 * memory locations into the symbol table
 * loc = memory location is inserted only the
 * first time, otherwise ignored
 */
void st_insert( char * name, int lineno, int loc )
{
    /*  请完成符号表条目插入子程序。
        先找，找不到说明定义了一个新名字，则插入新条目，默认插在其对应哈希表项的表
头；
        找到说明仅对名字进行引用，则将行号加入到该名字的行链中    */
    int h = hash(name);
    BucketList l = hashTable[h];
    while ((l!=NULL)&&(strcmp(name,l->name)!=0))
          l = l->next;
    if(l==NULL)
```

```
    {
        l = (BucketList)malloc(sizeof(struct BucketListRec));
        l->name = name;
        l->lines = (LineList)malloc(sizeof(struct    LineListRec));
        l->lines->lineno = lineno;
        l->memloc = loc;
        l->lines->next = NULL;
        l->next = hashTable[h];
        hashTable[h] = l;

    } else{
       LineList    t = l->lines;
       while (t->next!=NULL)
            t = t->next;
       t->next=(LineList)malloc(sizeof(struct LineListRec));
       t->next->lineno = lineno;
       t->next->next = NULL;
    }
} /* st_insert */

/* Function st_lookup returns the memory
 * location of a variable or -1 if not found
 */
int st_lookup ( char * name )
{ int h = hash(name);
   BucketList l =    hashTable[h];
   while ((l != NULL) && (strcmp(name,l->name) != 0))
      l = l->next;
   if (l == NULL) return -1;
   else return l->memloc;
}

/* Procedure printSymTab prints a formatted
 * listing of the symbol table contents
 * to the listing file
 */
void printSymTab(FILE * listing)
{ int i;
   fprintf(listing,"Variable Name    Location      Line Numbers\n");
   fprintf(listing,"-------------    --------      ------------\n");
   for (i=0;i<SIZE;++i)
   { if (hashTable[i] != NULL)
      { BucketList l = hashTable[i];
         while (l != NULL)
```

```
        { LineList t = l->lines;
          fprintf(listing,"%-14s ",l->name);
          fprintf(listing,"%-8d  ",l->memloc);
          while (t != NULL)
          { fprintf(listing,"%4d ",t->lineno);
            t = t->next;
          }
          fprintf(listing,"\n");
          l = l->next;
        }
      }
    }
} /* printSymTab */



/****************************************************/
/* File: analyze.c                                  */
/* Semantic analyzer implementation                 */
/* for the TINY compiler                            */
/****************************************************/

#include "globals.h"
#include "symtab.h"
#include "analyze.h"

/* counter for variable memory locations */
static int location = 0;

/* Procedure traverse is a generic recursive
 * syntax tree traversal routine:
 * it applies preProc in preorder and postProc
 * in postorder to tree pointed to by t
 */
static void traverse( TreeNode * t,
                      void (* preProc) (TreeNode *),
                      void (* postProc) (TreeNode *) )
{ if (t != NULL)
  { preProc(t);
    { int i;
      for (i=0; i < MAXCHILDREN; i++)
        traverse(t->child[i],preProc,postProc);
    }
    postProc(t);
```

```
      traverse(t->sibling,preProc,postProc);
    }
}

/* nullProc is a do-nothing procedure to
 * generate preorder-only or postorder-only
 * traversals from traverse
 */
static void nullProc(TreeNode * t)
{ if (t==NULL) return;
   else return;
}

/* Procedure insertNode inserts
 * identifiers stored in t into
 * the symbol table
 */
static void insertNode( TreeNode * t)
{ switch (t->nodekind)
   { case StmtK:
        switch (t->kind.stmt)
        { case AssignK:
          case ReadK:
            if (st_lookup(t->attr.name) == -1)
            /* not yet in table, so treat as new definition */
              st_insert(t->attr.name,t->lineno,location++);
            else
            /* already in table, so ignore location,
               add line number of use only */
              st_insert(t->attr.name,t->lineno,0);
            break;
          default:
            break;
        }
        break;
      case ExpK:
        switch (t->kind.exp)
        { case IdK:
            if (st_lookup(t->attr.name) == -1)
            /* not yet in table, so treat as new definition */
              st_insert(t->attr.name,t->lineno,location++);
            else
            /* already in table, so ignore location,
               add line number of use only */
```

```
                st_insert(t->attr.name,t->lineno,0);
                break;
            default:
                break;
        }
        break;
    default:
        break;
  }
}

/* Function buildSymtab constructs the symbol
 * table by preorder traversal of the syntax tree
 */
void buildSymtab(TreeNode * syntaxTree)
{ traverse(syntaxTree,insertNode,nullProc);
  if (TraceAnalyze)
  { fprintf(listing,"\nSymbol table:\n\n");
    printSymTab(listing);
  }
}

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

/* Procedure checkNode performs
 * type checking at a single tree node
 */
static void checkNode(TreeNode * t)
{ switch (t->nodekind)
  { case ExpK:                    /*  此处请填写完整  */
    switch (t->kind.stmt)
    {
        case ExpK:
            switch (t->kind.exp)
            {
                case OpK:
                    if((t->child[0]->type!=Integer)||
                            (t->child[1]->type!=Integer))
                        typeError(t,"Op applied to non-integer");
                    if((t->attr.op ==EQ)||(t->attr.op==LT))
                        t->type = Boolean;
```

```
                    else
                        t->type = Integer;
                    break;
                case ConstK:
                case IdK:
                    t->type = Integer;
                    break;
                default:
                    break;
            }
            break;
    }
  case StmtK:                    /*  此处请填写完整  */
  switch (t->kind.stmt)
  {
      case IfK:
          if(t->child[0]->type==Integer)
              typeError(t->child[0],"If test is not Boolean");
          break;
      case AssignK:
          if(t->child[0]->type!=Integer)
              typeError(t->child[0],"assignment of non-integer value");
          break;
      case WriteK:
          if(t->child[0]->type!=Integer)
              typeError(t->child[0],"write of non=integer value");
          break;
      case RepeatK:
          if(t->child[1]->type==Integer)
              typeError(t->child[1],"repeat test is not Boolean");
          break;
      default:
          break;
  }
          break;
      default:
          break;
  }
}

/* Procedure typeCheck performs type checking
 * by a postorder syntax tree traversal
 */
void typeCheck(TreeNode * syntaxTree)
```

{ traverse(syntaxTree,nullProc,checkNode);
}
(2)在安装完 tcc 后
①在 windows 的 shell 下运行 tcc main.c util.c parse.c scan.c symtab.c analyze.c

```
PS F:\大学资料\大三下\编译原理\编译原理作业\实验\4\TinyC-master\src> tcc main.c util.c parse.c scan.c symtab.c analyze.c

PS F:\大学资料\大三下\编译原理\编译原理作业\实验\4\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\4\TinyC-master\src


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2019.06.09     20:52                cmake-build-debug
-a----        2019.06.09     21:36           4469 ANALYZE.C
-a----        2019.04.24     01:35            540 ANALYZE.H
-a----        2019.04.24     01:35           6759 cgen.c
-a----        2019.04.24     01:35            660 cgen.h
-a----        2019.04.24     01:35            101 CMakeLists.txt
-a----        2019.04.24     01:35           2843 GLOBALS.H
-a----        2019.05.26     22:04           2248 MAIN.C
-a----        2019.06.09     21:59          11264 main.exe
-a----        2019.05.26     22:13           5830 PARSE.C
-a----        2019.04.24     01:35            372 PARSE.H
-a----        2019.04.24     01:35            263 SAMPLE.TNY
-a----        2019.04.24     01:35           6348 SCAN.C
-a----        2019.04.24     01:35            547 SCAN.H
-a----        2019.06.09     21:52           3583 SYMTAB.C
-a----        2019.04.24     01:35            847 SYMTAB.H
-a----        2019.04.24     01:35           4736 UTIL.C
-a----        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\4\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
```