華南師範大學

本科学生实验（实践）报告

院　　系：计 算 机 学 院

实验课程：编译原理

实验项目：TINY 扩充语言的语法分析

指导老师：黄煜廉

开课时间：2019 ～ 2020 年度第 2 学期

专　　业：计算机科学与技术

班　　级：2017 级 4 班

学　　生：曾德明

学　　号：20172131138

华南师范大学教务处

目录

一、实验项目

TINY 扩充语言的语法分析。具体如下：

扩充的语法规则有：实现 while、do while、for 语句、乘方运算符号以及+=运算符号，具体文法规则自行构造。

可参考：P97 及 P136 的文法规则。

(1) While-stmt --> while exp do stmt-sequence endwhile

(2) Dowhile-stmt-->do stmt-sequence while(exp);

(3) for-stmt-->for identifier:=simple-exp to simple-exp do stmt-sequence enddo 步长递增 1

(4) for-stmt-->for identifier:=simple-exp downto simple-exp do stmt-sequence enddo 步长递减 1

(5) 大于>比较运算符号以及求余计算式子的文法规则请自行组织。

(6) 把 TINY 语言原有的 if 语句书写格式

if_stmt-->if exp then stmt-sequence end | | if exp then stmt-sequence else stmt-sequence end

改写为：

if_stmt-->if(exp) stmt-sequence else stmt-sequence | if(exp) stmt-sequence

二、实验要求

（一）要提供一个源程序编辑界面，以让用户输入源程序（可保存、打开源程序）

（二）可由用户选择是否生成语法树，并可查看所生成的语法树

（三）可应该书写完善的软件文档。

三、需求分析

（一）需要能打开指定的 Tiny 源代码文件并提供一个窗口显示 Tiny 源代码文件内容；

（二）需要提供一个窗口输入/编辑 Tiny 源代码文件内容；

（三）需要提供对于指定的 Tiny 源代码文件的语法树的生成和显示功能。

四、概要设计

（一）修改教材附录 B 的相关文件

修改教材附录 B 中的相关文件 globals.h、parse.h、scan.h、util.h、main.c、parse.c、scan.c、util.c，共八个文件，使得其可以实现对于 Tiny 扩充语言的语法分析，生成对应的语法树；scan.cpp 用于扫描程序完成词法分析、parse.cpp 用于完成语法分析、util.cpp 用于完成结果的显示，以及 main.cpp 作为主函数实现交互，因此主要修改这几个函数的部分内容即可；

（二）把项目更改为 c++项目并更改 main.c

为了减少移植到 QT 的工作，首先把 c 项目转化为 c++项目，同时更改 main 函数的读取数据方式，改为从文件读取；

```
// 获取当前文件路径
QByteArray ba = getFileName().toUtf8();
char * filename = ba.data();   // 把文件名转化为 char *
source = fopen(filename,"r");
```

（三）导入 QT 并实现可视化功能

使用 QT 绘制出可视化界面，并把原本的 DOS 黑框程序移植到 QT 项目之中；

五、详细设计（核心模块的代码）

（一）修改教材附录 B 的相关文件

globals.h、parse.h、scan.h、util.h、main.c、parse.c、scan.c、util.c

1. globals.h

（1）修改支持的保留字个数

```
/* MAXRESERVED = the number of reserved words(从 8 个添加到 15 个) */
#define MAXRESERVED 15
```

（2）在 typedef enum{} TokenType 结构中增加 WHILE,DO,TO,DOWNTO,FOR,ENDDO,ENDWHILE 以及 POW, ADDTO

```
typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words(WHILE,DO,TO,DOWNTO,FOR,ENDDO,ENDWHILE) */

IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,WHILE,DO,TO,DOWNTO,FOR,ENDDO,ENDWH
ILE,
    /* multicharacter tokens */
    ID,NUM,
    /* special symbols(POW,ADDTO) */

ASSIGN,EQ,LT,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,SEMI,POW,ADDEQUAL,ADDTO,
   } TokenType;
```

（3）在 typedef enum {} StmtKind 结构中增加语句类型 WhileK, DoWhileK, ForK, AddtoK

```
typedef enum {IfK,RepeatK,AssignK,ReadK,WriteK,WhileK,DoWhileK,ForK,AddtoK}
StmtKind; // add WhileK, DoWhileK, ForK,AddtoK
```

2. parse.h

无需修改，只需引用。

3. util.h

无需修改，只需引用。

4. scan.h

为了实现多次生成，把 scan.c 中的三个变量移入 scan.h：

```
extern int linepos; /* current position in LineBuf */
extern int bufsize; /* current size of buffer string */
extern int EOF_flag; /* corrects ungetNextChar behavior on EOF */
```

6. parse.c

（1）增加静态成员函数定义

```
// new
static TreeNode * while_stmt(void);
static TreeNode * dowhile_stmt(void);
static TreeNode * for_stmt(void);
```

（2）分别实现上面三个函数

```
// while_stmt(While-stmt --> while  exp  do  stmt-sequence  endwhile)
TreeNode * while_stmt(void)
{
    TreeNode * t = newStmtNode(WhileK);
    match(WHILE);
    if (t != NULL) t->child[0] = exp();
    match(DO);
```

```
    if (t != NULL) t->child[1] = stmt_sequence();
    match(ENDWHILE);
    return t;
}

// dowhile_stmt(Dowhile-stmt-->do  stmt-sequence  while(exp))
TreeNode * dowhile_stmt(void)
{
    TreeNode * t = newStmtNode(DoWhileK);
    match(DO);
    if (t != NULL) t->child[0] = stmt_sequence();
    match(WHILE);
    match(LPAREN); // (
    if (t != NULL) t->child[1] = exp();
    match(RPAREN); // )
    return t;
}

// for_stmt
// for-stmt-->for identifier:=simple-exp  to  simple-exp  do  stmt-sequence
enddo
// for-stmt-->for identifier:=simple-exp  downto  simple-exp  do
stmt-sequence enddo
TreeNode * for_stmt(void)
{
    TreeNode * t = newStmtNode(ForK);
    match(FOR);
    if ((t != NULL) && (token == ID))
    {
        t->attr.name = copyString(tokenString);
    }
    match(ID);
    match(ASSIGN);
    if (t != NULL)
    {
        t->child[0] = simple_exp();
    }
    if (token == TO)
    { // +1
        match(TO);
    }
    if (token == DOWNTO)
    { // -1
        match(DOWNTO);
```

```
    }
    if (t != NULL)
    {
        t->child[1] = simple_exp();
    }
    match(DO);
    if (t != NULL)
    {
        t->child[2] = stmt_sequence();
    }
    match(ENDDO);
    return t;
}
```

（3）更改现有的 if_stmt 函数

```
// if_stmt(if_stmt-->if(exp) stmt-sequence else stmt-sequence | if(exp)
stmt-sequence)
TreeNode * if_stmt(void)
{
    TreeNode * t = newStmtNode(IfK);
    match(IF);
    match(LPAREN);
    if (t != NULL) t->child[0] = exp();
    match(RPAREN);
    if (t != NULL) t->child[1] = stmt_sequence();
    if (token == ELSE)
    {
        match(ELSE);
        if (t != NULL) t->child[2] = stmt_sequence();
    }
    return t;
}
```

（4）根据观察可得，+=的实现类似于:=赋值符号，根据文法，statement 如果 match 到 ID 的话，会直接跳到 assign_stmt 函数进行赋值符号判断，所以可更改函数来同时实现+=

```
TreeNode * assign_stmt(void)
{
    TreeNode * t = newStmtNode(AssignK);
    if ((t!=NULL) && (token==ID))
        t->attr.name = copyString(tokenString);
    match(ID);
    //match(ASSIGN);
    // +=类比:=
    if(token == ASSIGN) {
        match(ASSIGN);
    }
    else {
        t->kind.stmt = AddtoK;
        match(ADDTO);
    }
    if (t!=NULL)
        t->child[0] = exp();
    return t;
}
```

（5）增加 stmt_sequence 文法的 follow 集，添加增加条件(token != ENDWHILE) && (token != WHILE)&&(token != ENDDO)

```
TreeNode * stmt_sequence(void)
{
    TreeNode * t = statement();
    TreeNode * p = t;
    // (token != ENDWHILE) && (token != WHILE)&&(token != ENDDO)
    while ((token != ENDFILE) && (token != END) &&
        (token != ELSE) && (token != UNTIL)&&
        (token != ENDWHILE) && (token != WHILE)&&
        (token != ENDDO))
    {
        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;
}
```

（5）在 TreeNode * statement(void)函数中 switch (token)模块下增加状态

```
TreeNode * statement(void)
{
    TreeNode * t = NULL;
    // 在 TreeNode * statement(void)函数中 switch (token)模块下增加状态
DO,WHILE,FOR
    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;
    // new
    case WHILE:
        t = while_stmt();
```

```
            break;
        case DO:
            t = dowhile_stmt();
            break;
        case FOR:
            t = for_stmt();
            break;
        default :
            syntaxError("unexpected token -> ");
            printToken(token,tokenString);
            token = getToken();
            break;
        } /* end case */
        return t;
}
```

（6）在 TreeNode * term(void)函数中的 while 条件增加代码

```
// 在 TreeNode * term(void)函数中的 while 条件增加代码（pow 乘方）
    while ((token==TIMES)||(token==OVER)||(token==POW))
```

7. scan.c

（1）在 static struct {} reservedWords[MAXRESERVED]中增加保留字映射关系

```
/* lookup table of reserved words(添加保留字映射) */
static struct
{
    char* str;
    TokenType tok;
} reservedWords[MAXRESERVED]
= {{"if",IF},{"then",THEN},{"else",ELSE},{"end",END},
    {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
    {"write",WRITE},{"do", DO}, {"while", WHILE},
    {"to", TO}, {"downto", DOWNTO}, {"for", FOR},
    {"enddo", ENDDO}, {"endwhile", ENDWHILE}
};
```

（2） 在 TokenType getToken(void)函数中的 switch (c)模块下增加状态

```
case '^':
  currentToken = POW;  // add 乘方
  break;
```

（3） 在 typedef enum {} StateType 中增加状态 INADDTO

```
/* states in scanner DFA(添加状态 INADDTO) */
typedef enum
{ START,INASSIGN,INCOMMENT,INNUM,INID,DONE,INADDTO }
StateType;
```

（4） 在 TokenType getToken(void)函数中的 switch (c)模块下更改状态

```
case '+':
     //currentToken = PLUS;
     state = INADDTO; // 如果读到+，先不 match，跳转到状态 INADDTO
     break;
```

（5） 在 TokenType getToken(void)函数中的 case INCOMMENT

模块下增加状态

```
     // +=
    case INADDTO:
        state = DONE;
        if(c == '=')
            currentToken = ADDTO; // 如果+后接=
        else
            currentToken = PLUS;  // 如果只是个+
        break;
```

8. util.c

（1） 在 void printToken(TokenType token, const char* tokenString)函数中 switch (token)模块下增加状态

```
// 新增
case WHILE:
case DO:
case TO:
case DOWNTO:
case FOR:
case ENDDO:
case ENDWHILE:
    fprintf(listing,"reserved word: %s\n",tokenString);
    break;
// +=
case ADDTO:
    fprintf(listing,"+=\n");
    break;
case POW: // add ^
    fprintf(listing,"^\n");
    break;
```

（2） 在 void printTree(TreeNode * tree)函数中 switch()模块下增加状态

```
// +=
case AddtoK:
    fprintf(listing, "Add to: %s\n", tree->attr.name);
    break;
// new
case WhileK:
    fprintf(listing, "While\n");
    break;
case DoWhileK:
    fprintf(listing, "Do While\n");
    break;
case ForK:
    fprintf(listing, "For\n");
    break;
```

8. main.c

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

/* allocate global variables */
int lineno = 0;
FILE * source;
FILE * listing;
FILE * code;

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
int Error = FALSE;

int main( int argc, char * argv[] )
{
    TreeNode * syntaxTree;
    char pgm[120],ch[120];
    int n, flag = 1;
    while(flag)
    {
        switch(flag)
        {
        case 1:
            {
                printf("-------TINY analyis-------\n");
                printf("--open file: \n");
                printf("1.WhileSample.tiny\n");
                printf("2.DoWhileSample.tiny\n");
                printf("3.ForSample.tiny\n");
                printf("4.TestSample.tiny\n");
                printf("5.exit program \n");
                printf("\n");printf("\n");
```

```
printf("please select (1-5): ");
scanf("%d",&n);
while(n<1||n>5)
{
    printf("-----wrong input, please input again: \n");
    scanf("%d",&n);
}
switch(n)
{
case 1: // 打开文件 WhileSample.tiny
    {
        source=fopen("WhileSample.tiny","r");
        strcpy(pgm,"WhileSample.tiny"); // 文件名
        if (source == NULL)
        {
            fprintf(stderr, "File %s not found\n", pgm);
            exit(1);
        }
        break;
    }
case 2: // 打开文件 DoWhileSample.tiny
    {
        source=fopen("DoWhileSample.tiny","r");
        strcpy(pgm,"DoWhileSample.tiny"); // 文件名
        if (source == NULL)
        {
            fprintf(stderr, "File %s not found\n", pgm);
            exit(1);
        }
        break;
    }
case 3: // 打开文件 ForSample.tiny
    {
        source=fopen("ForSample.tiny","r");
        strcpy(pgm,"ForSample.tiny"); // 文件名
        if (source == NULL)
        {
            fprintf(stderr, "File %s not found\n", pgm);
            exit(1);
        }
        break;
    }
case 4: // 打开文件 TestSample.tiny
    {
```

```
                source=fopen("TestSample.tiny","r");
                strcpy(pgm,"TestSample.tiny"); // 文件名
                if (source == NULL)
                {
                    fprintf(stderr, "File %s not found\n", pgm);
                    exit(1);
                }
                break;
            }
        case 5: // 结束程序
        default:
            fclose(source);//关闭文件
            exit(0);
        }
    }
case 2:
    {
        printf("*******load source code succeed!*******\n");
        printf("*******1.check source code\n"); // 查看源文件
        printf("*******2.check grammer tree\n"); // 查看语法树
        printf("*******3.return\n"); // 返回上一级
        printf("please select (1-3): ");
        scanf("%d",&n);
        while(n<1||n>3)
        {
            printf("-----wrong input, please input again: \n");
            scanf("%d",&n);
        }
        if(n==3)
        {
            flag = 1;
            fclose(source);
            break;
        }
        else
        {
            flag = 2;
            switch(n)
            {
            case 1: // 打开源程序
                {
                    while(!feof(source))
                    {
                        if(fgets(ch,120,source)!=0)
```

```
                                printf("%s",ch);
                            }
                            fseek(source,0L,0);
                            break;//读取结束，将文件指针指向头位置
                        }
                    case 2: // 查看生成树
                        {
                            //listing = fopen("test.txt", "w");
                            listing = stdout;
                            fprintf(listing, "\nTINY COMPILATION: %s\n\n", pgm);
                            // 进行重置
                            linepos = 0; /* current position in LineBuf */
                            bufsize = 0; /* current size of buffer string */
                            EOF_flag = FALSE; /* corrects ungetNextChar behavior
on EOF */
                            syntaxTree = parse(); // 生成
                            fprintf(listing, "Syntax tree:\n\n");
                            printTree(syntaxTree);
                            fseek(source,0L,0);
                            //fclose(listing);
                            break;//读取结束，将文件指针指向头位置
                        }
                    }
                }
            }
        case 3:
            {
                printf("\n");
                printf("*****1.return\n");
                printf("*****2.exit program\n");
                printf("please select (1-2): ");
                scanf("%d",&n);
                while(n<1||n>2)
                {
                    printf("*****wrong input, please input again: \n");
                    scanf("%d",&n);
                }
                if(n==1)
                    flag=2;
                else if(n==2)
                    flag=0;
            }
        }
    }
```

```
    fclose(source);
    fclose(listing);

    return 0;
}
```

（二）抽象出函数接口，移植入 QT

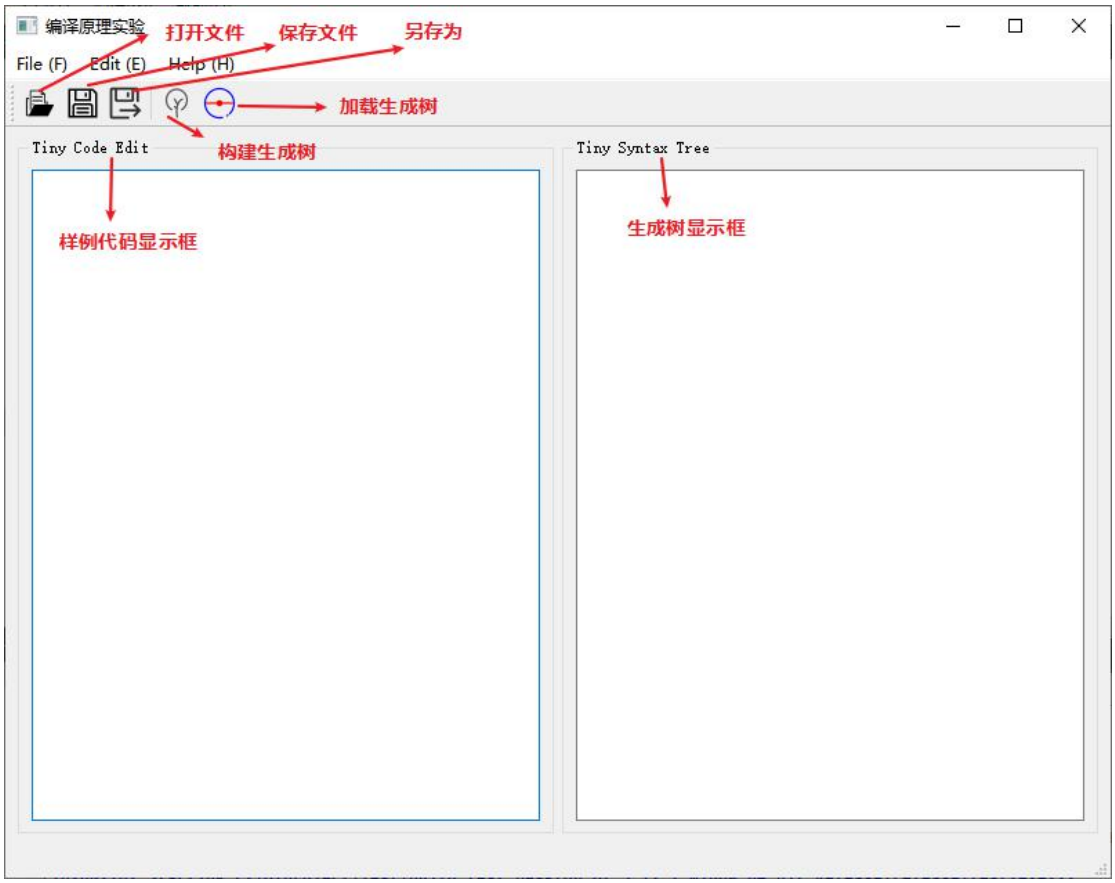
```
// 生成树
bool MainWindow::tinySyntaxTree()
{
    if(getFileName() == "")
    {
        return false;
    }
    else
    {
        TreeNode * syntaxTree;
        // 获取当前文件路径
        QByteArray ba = getFileName().toUtf8();
        char * filename = ba.data();   // 把文件名转化为 char *
        qDebug() << filename;
        QStringList list = getFileName().split("/"); // 把文件路径分割以获取最后的文件名
        // 需要一个转化文件名的过程(用于解决存储)
        QString openFilePath = getFileName();   // 测试文件路径
        qDebug() << "openFilePath-->"+openFilePath;
        QString treeName = openFilePath.insert(openFilePath.lastIndexOf('/')+1, "tree");   // 生成的树文件直接存储在测试用例文件的同级中， 并重命名为 tree xxx.tiny
        qDebug() << "treeName-->"+treeName;
        QByteArray bb = treeName.toUtf8();
        char * saveName = bb.data();
        source = fopen(filename,"r");
        if (source == NULL)
        {
            // todo 弹窗
            fprintf(stderr, "File %s not found\n", filename);
            return false;
            //exit(1);
        }
        listing = fopen(saveName, "w");
```

```
        fprintf(listing, "\nTINY COMPILATION: %s\n\n", saveName);
        // 进行重置
        linepos = 0; /* current position in LineBuf */
        bufsize = 0; /* current size of buffer string */
        EOF_flag = FALSE; /* corrects ungetNextChar behavior on EOF */
        syntaxTree = parse(); // 生成
        fprintf(listing, "Syntax tree:\n\n");
        printTree(syntaxTree);
        fseek(source,0L,0);
        fclose(source);
        fclose(listing);
        return true;
    }
}

// 加载生成树
bool MainWindow::loadtree()
{
    if(getFileName() == "")
    {
        return false;
    }
    else
    {
        QString openFilePath = getFileName();   // 测试文件路径
        QStringList list = getFileName().split("/");
        qDebug() << "openFilePath-->"+openFilePath;
        QString openTreeName =
openFilePath.insert(openFilePath.lastIndexOf('/')+1, "tree");
        qDebug() << "openTreeName-->"+openTreeName;
        QFile file(openTreeName);
        if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
        {
            QMessageBox::warning(this,"error","open file error!");
            return false;
        }
        else
        {
            if(!file.isReadable())
                QMessageBox::warning(this,"error","this file is not
readable!");
            else
            {
                QTextStream textStream(&file);
```

```
                while(!textStream.atEnd())
                {
                    treeBrowser->setTextColor(Qt::red);
                    treeBrowser->setPlainText(textStream.readAll());
                }
                treeBrowser->show();
            }
        }
        return true;
    }
}
```

（三）使用 QT 的 QToolbar，QTextBrowser，QTextEdit 等组件实现可视化



六、运行结果

（样例代码有些部分不符合 Tiny 语言语法，主要是分号的处理，本人对其进行了相应修改）

（一）Tiny 扩充语言的样例代码一：DoWhileSample.tiny

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

生成的语法树：

Syntax tree:

```
  Read: x
  If
    Op: <
      Const: 0
      Id: x
    Assign to: fact
      Const: 1
    Do While
      Assign to: fact
        Op: *
          Id: fact
          Id: x
      Assign to: x
        Op: -
          Id: x
```

```
            Const: 1
        Op: <
            Const: 0
            Id: x
    Write
        Id: fact
```

（二）Tiny 扩充语言的样例代码二：ForSample.tiny

```
{ Sample program
 in TINY language -
 computes factorial
}
read x; { input an integer }
if(0<x) { don't compute if x <= 0 }
    for fact := x downto 1 do
        fact := fact * x
    enddo;
    write fact { output factorial of x }
```

生成的语法树：

```
Syntax tree:

  Read: x
  If
    Op: <
      Const: 0
      Id: x
    For
      Id: x
      Const: 1
      Assign to: fact
        Op: *
          Id: fact
          Id: x
    Write
      Id: fact
```

（三）Tiny 扩充语言的样例代码三：WhileSample.tiny

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if(0<x) { don't compute if x <= 0 }
        fact := 1;
        while 0<x do
                fact := fact * x;
                x := x - 1
        endwhile;
        write fact { output factorial of x }
```

生成的语法树：

Syntax tree:

```
   Read: x
   If
      Op: <
         Const: 0
         Id: x
      Assign to: fact
         Const: 1
      While
         Op: <
            Const: 0
            Id: x
         Assign to: fact
            Op: *
               Id: fact
```

```
            Id: x
        Assign to: x
          Op: -
            Id: x
            Const: 1
    Write
      Id: fact
```

（四）Tiny 扩充语言的样例代码三：TestSample.tiny

```
{ Sample program
 in TINY language -
 computes factorial
}
read x; { input an integer }
if(0<x) { don't compute if x <= 0 }
    fact := 1;
    while 0<x do
        fact += 2;
        x := x - 1
    endwhile;
    write fact { output factorial of x }
```

生成的语法树：

Syntax tree:

```
  Read: x
  If
    Op: <
      Const: 0
      Id: x
    Assign to: fact
```

Const: 1

While

Op: <

Const: 0

Id: x

Add to: fact

Const: 2

Assign to: x

Op: -

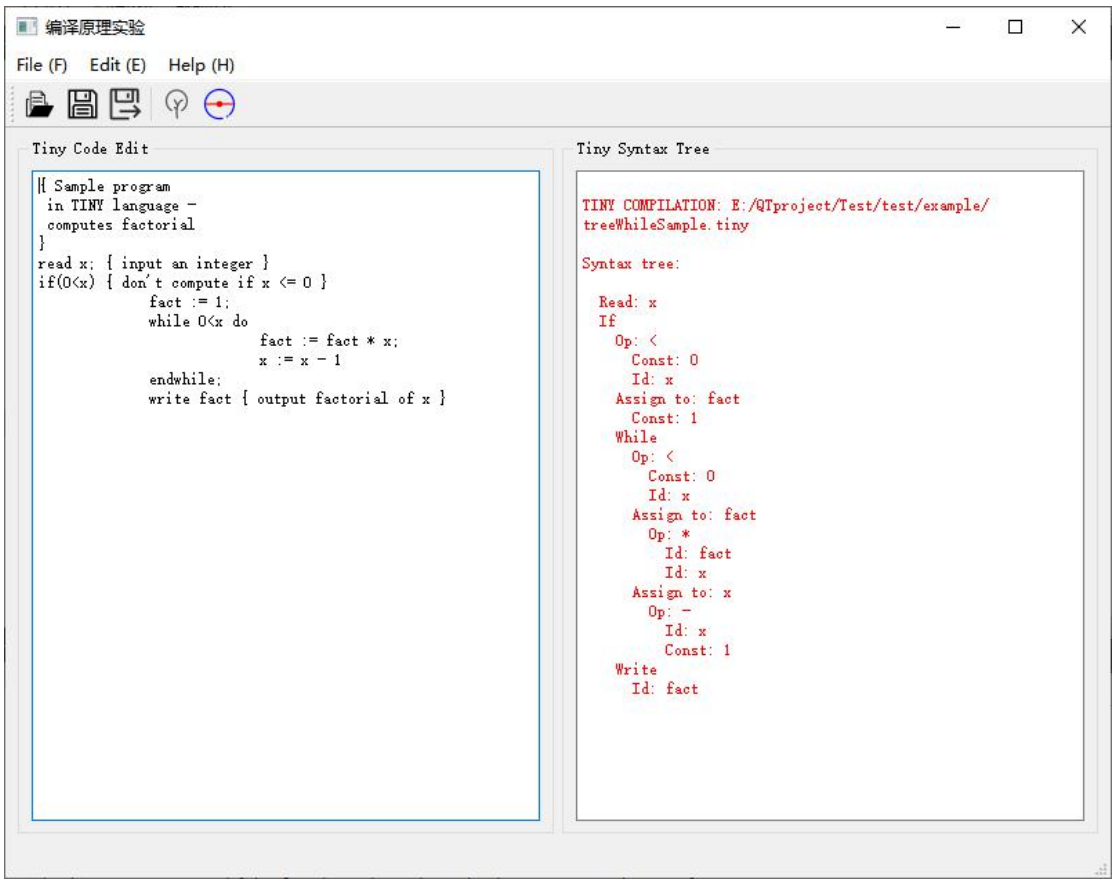
Id: x

Const: 1

Write

Id: fact

（五）GUI 界面



七、参考文献

（一） QT 官方文档

（二） 《编译原理及实践》

（三） 黄煜廉老师的 PPT

八、总结

这次的实验让我很好的了解了 Tiny 的文法和语法，同时也尝试使用了递归下降的文法分析方式去进行文法分析。做完实验后，我对于语法分析有了更多一些的了解，对于递归下降的分析方法也有了更多的理解，同时也学会了使用调试找出程序的问题所在。