**编译原理**

班级：20计算机科学与技术

姓名：陈伟剑 学号：2020333503081

实验三 基于YACC的TINY语法分析器的构建

1. 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。

1. 实验方案

**输入**

使用教材附录中的案例，并添加了一点内容

{ 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;

read y;

repeat

y := y + x;

x := x + 1;

write x

until x = 10;

write fact { output factorial of x }

end

**输出**

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

Read: y

Repeat

Assign to: y

Op: +

Id: y

Id: x

Assign to: x

Op: +

Id: x

Const: 1

Write

Id: x

Op: =

Id: x

Const: 10

Write

Id: fact

**lex(词法分析)、yacc(语法分析)之间的数据传递**

①lex和yacc底层的运行过程

查资料([1][2])，了解到yacc工具产生的分析器在执行过程中，会调用yylex()来读入下一个token，yylex()会返回一个单词符号，并将相关的属性值存入全局变量yylval。通过bison -d tiny.y 命令可以生成tiny.tab.h，这个文件中存储了先前写好的tiny语法下各种token的定义。

接下来的两步用于创建lex和yacc之间的数据传递。

②修改tiny.y

教材附录中已经给出了一个tiny.y文件。删除文件中的yylex(void)函数。

③修改生成的tiny.tab.c文件

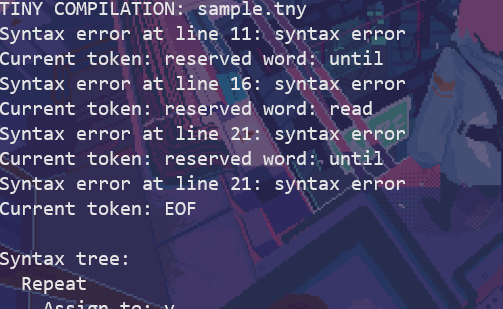
将第1304行的yychar=YYLEX; 修改为yychar=getToken();

至此就能将lex扫描到的token传到yacc中。

1. 分析表parsing table问题

理论和设计

语法分析会用到LR分析表，这个表可以通过bison -v tiny.y命令生成，文件名为tiny.output。对于一个状态或输入的token，若表中没有给出相关的动作，则调用yyerror()函数报错，报错效果如图。以上过程通过内部的yyparse()函数实现。



1. 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**

%token **ERROR**

**%%** /\* Grammar for TINY \*/

program     : stmt\_seq

                 { savedTree = **$1**;}

            ;

stmt\_seq    : stmt\_seq SEMI stmt

                 { YYSTYPE t = **$1**;

                   if (t != NULL)

                   { while (t->sibling != NULL)

                        t = t->sibling;

                     t->sibling = **$3**;

**$$** = **$1**; }

                     else $$ = $3;

                 }

            | stmt  { **$$** = **$1**; }

            ;

stmt        : if\_stmt { **$$** = **$1**; }

            | repeat\_stmt { **$$** = **$1**; }

            | assign\_stmt { **$$** = **$1**; }

            | read\_stmt { **$$** = **$1**; }

            | write\_stmt { **$$** = **$1**; }

            | error  { **$$** = NULL; }

            ;

if\_stmt     : IF exp THEN stmt\_seq END

                 { **$$** = newStmtNode(IfK);

**$$**->child[0] = **$2**;

**$$**->child[1] = **$4**;

                 }

            | IF exp THEN stmt\_seq ELSE stmt\_seq END

                 { **$$** = newStmtNode(IfK);

**$$**->child[0] = **$2**;

**$$**->child[1] = **$4**;

**$$**->child[2] = **$6**;

                 }

            ;

repeat\_stmt : REPEAT stmt\_seq UNTIL exp

                 { **$$** = newStmtNode(RepeatK);

**$$**->child[0] = **$2**;

**$$**->child[1] = **$4**;

                 }

            ;

assign\_stmt : ID { savedName = copyString(tokenString);

                   savedLineNo = lineno; }

              ASSIGN exp

                 { **$$** = newStmtNode(AssignK);

**$$**->child[0] = **$4**;

**$$**->attr.name = savedName;

**$$**->lineno = savedLineNo;

                 }

            ;

read\_stmt   : READ ID

                 { **$$** = newStmtNode(ReadK);

**$$**->attr.name =

                     copyString(tokenString);

                 }

            ;

write\_stmt  : WRITE exp

                 { **$$** = newStmtNode(WriteK);

**$$**->child[0] = **$2**;

                 }

            ;

exp         : simple\_exp LT simple\_exp

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = LT;

                 }

            | simple\_exp EQ simple\_exp

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = EQ;

                 }

            | simple\_exp { **$$** = **$1**; }

            ;

simple\_exp  : simple\_exp PLUS term

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = PLUS;

                 }

            | simple\_exp MINUS term

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = MINUS;

                 }

            | term { **$$** = **$1**; }

            ;

term        : term TIMES factor

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = TIMES;

                 }

            | term OVER factor

                 { **$$** = newExpNode(OpK);

**$$**->child[0] = **$1**;

**$$**->child[1] = **$3**;

**$$**->attr.op = OVER;

                 }

            | factor { **$$** = **$1**; }

            ;

factor      : LPAREN exp RPAREN

                 { **$$** = **$2**; }

            | NUM

                 { **$$** = newExpNode(ConstK);

**$$**->attr.val = atoi(tokenString);

                 }

            | ID { **$$** = newExpNode(IdK);

**$$**->attr.name =

                         copyString(tokenString);

                 }

            | error { **$$** = NULL; }

            ;

**%%**

int yyerror(char \* message)

{ fprintf(listing,"Syntax error at line %d: %s\n",lineno,message);

  fprintf(listing,"Current token: ");

  printToken(yychar,tokenString);

  Error = TRUE;

  return 0;

}

/\* yylex calls getToken to make Yacc/Bison output

 \* compatible with ealier versions of the TINY scanner

此处的yylex(void)函数已删除

 \* /

TreeNode \* parse(void)

{ yyparse();

  return savedTree;

}

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);

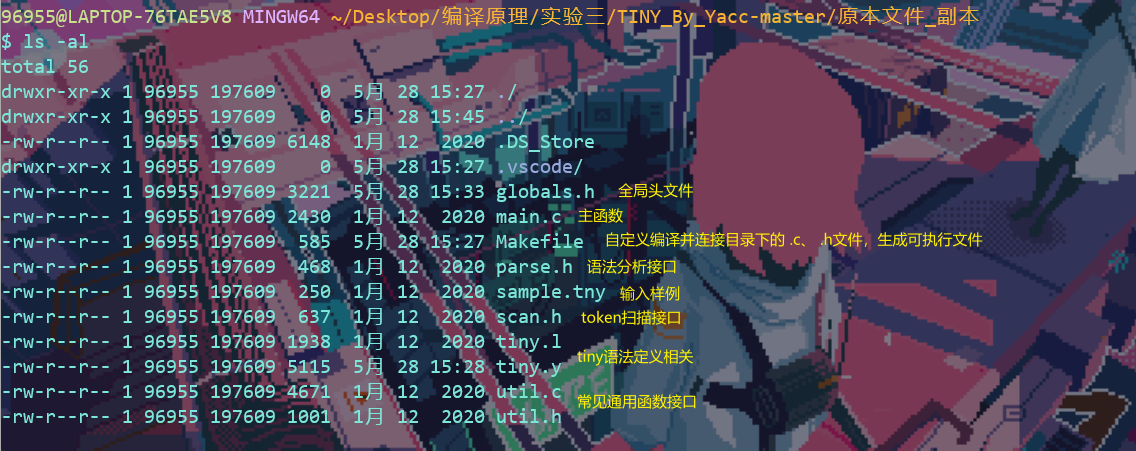
  }

  return currentToken;

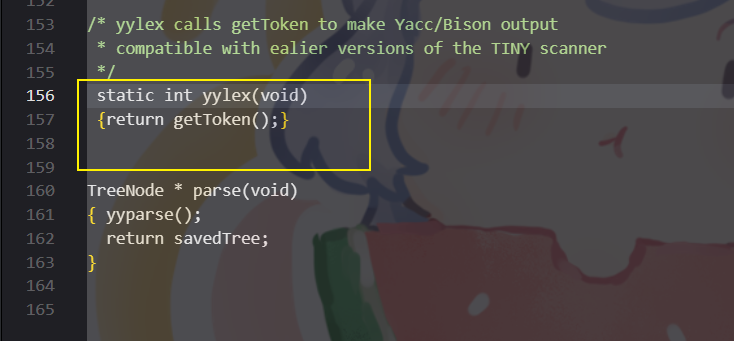
}

3、实验具体步骤

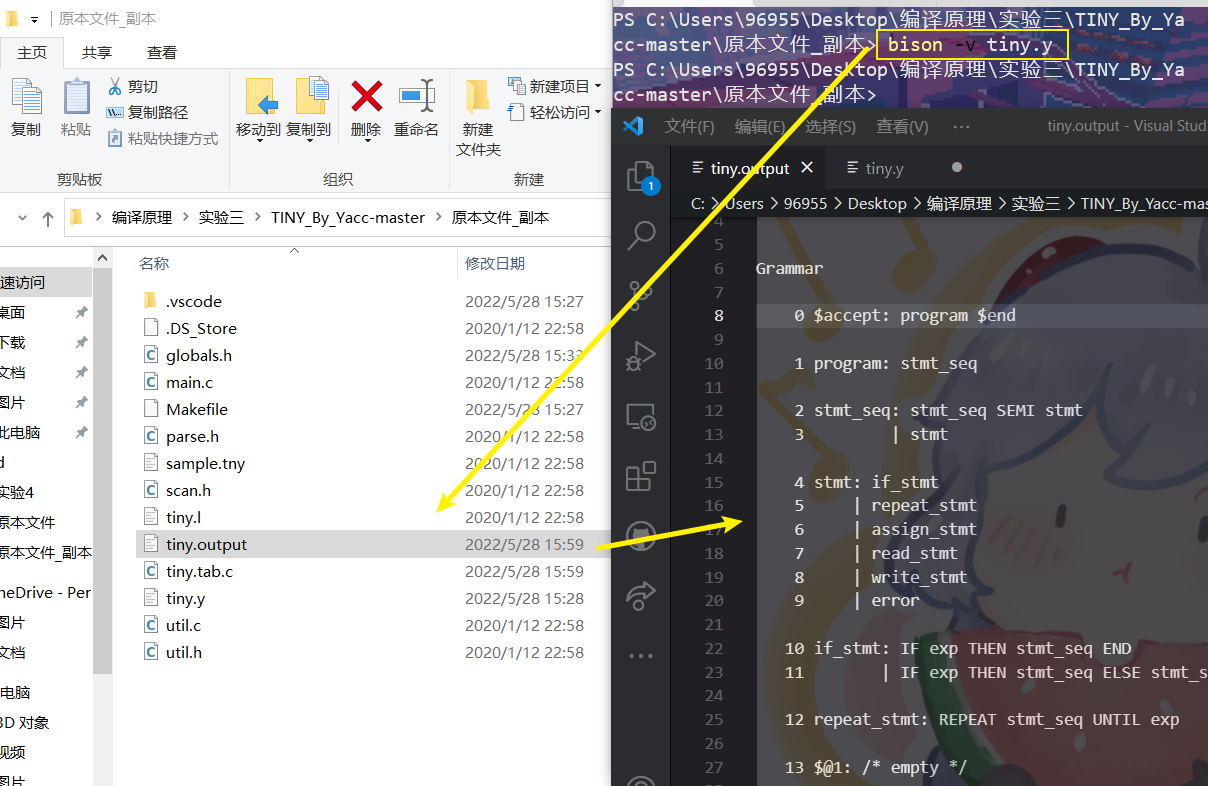
项目目录如下



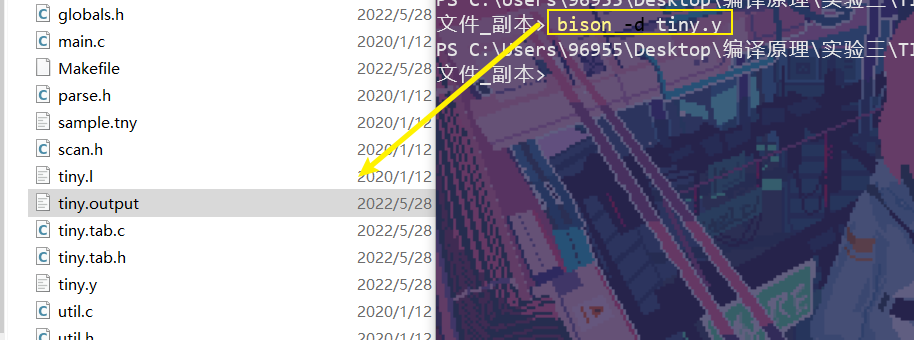
①删除tiny.y文件中的yylex(void)函数，如下



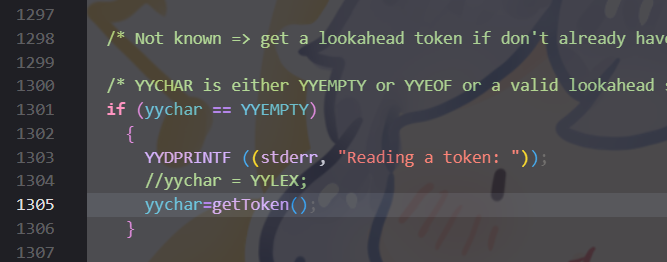
②输入命令bison -v tiny.y 生成tiny.ouput和tiny.tab.c。tiny.output中存储的是LR分析表



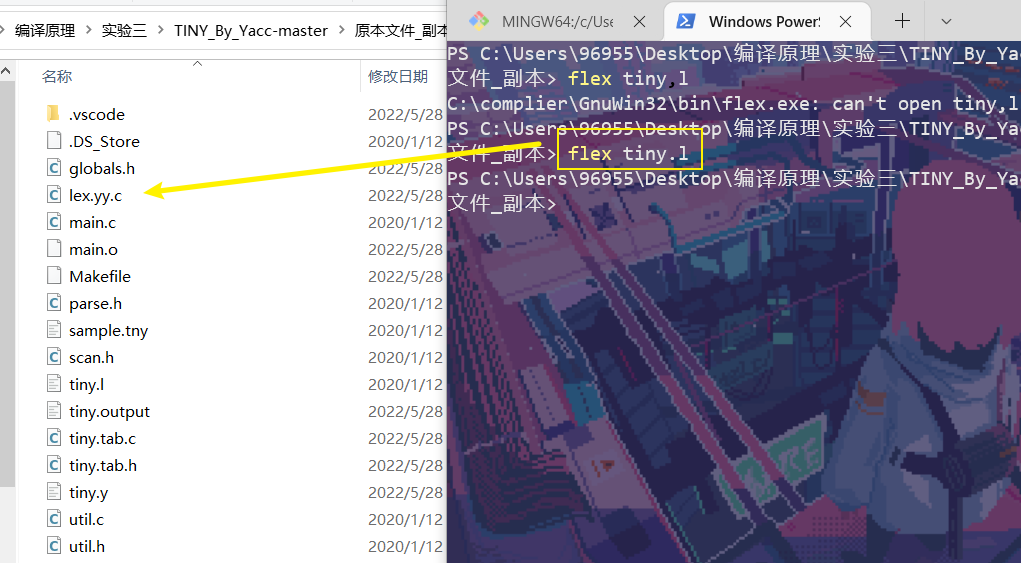
③输入命令 bison -d tiny.y 生成tiny.tab.h



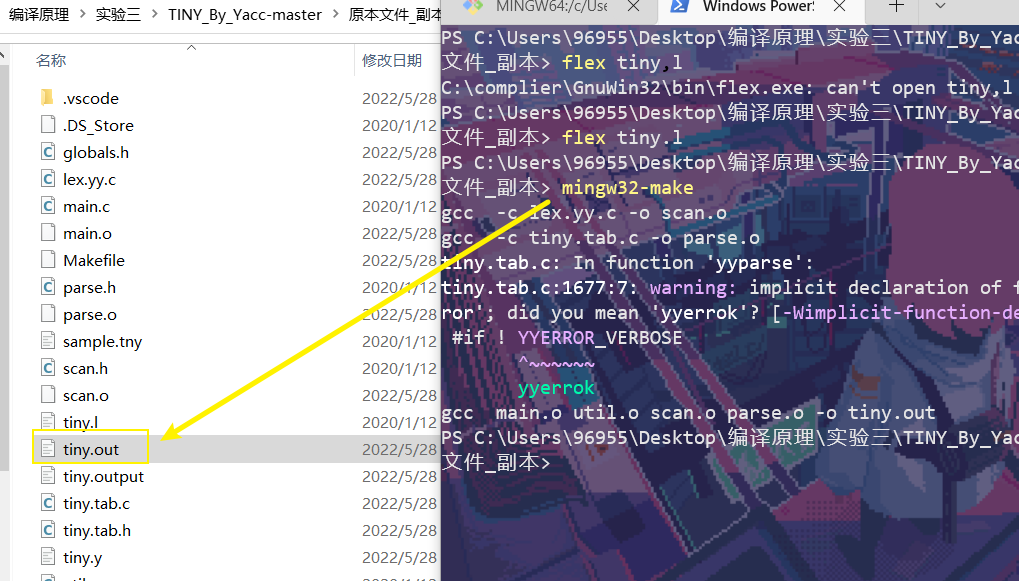
④在tiny.tab.c文件中，第1304行的yychar=YYLEX; 修改为yychar=getToken();



⑤输入 flex tiny.l 生成yy.lex.c



⑥输入mingw32-make命令执行目录下的Makefile文件，生成tiny.out可执行文件(不同环境下这个执行命令可能不同)

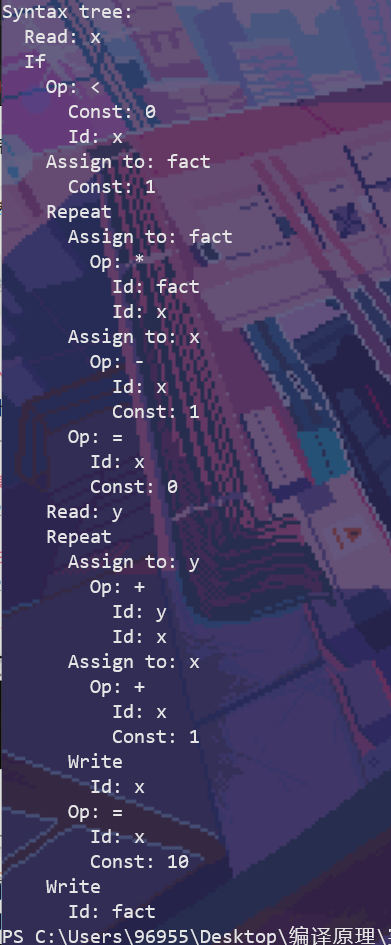


⑦由于实验环境是windows，不能执行.out文件，这里先将tiny.out文件名改为tiny.exe。然后在控制台输入 .\tiny sample.tny，这个命令可以执行tiny.exe文件(前面的.\代表信任该程序)，同时将sample.tny作为输入文件。



1. 实验结果:

截图



1. 实验结论:

1 、实验结论

整个实验过程中用到了lex和yacc两个工具，给定一个输入sample.tny，先用lex进行词法分析(和目录中的tiny.l、yy.lex.c相关)。然后将词法分析结果通过yacc进行语法分析，这个过程通过编译生成的tiny.exe程序来执行，语法分析所需要的LR分析表由tiny.output给出。

2、分析和总结

①工具

lex 和 yacc 两个工具已经封装好了基本的词法分析和语法分析功能，只需要依据规定编写和生成指定文件即可完成一个语法分析器。

②手写

手写需要从头开始考虑词法分析和语法分析的整个过程，需要花费更多时间进行分析器的设计和代码编写。由于lex和yacc已经比较完善，它们的内部程序所考虑的情况会更加全面且具体，因此短时间内手写完成的语法分析器的健壮性会比较弱。

3、实验中出现的冲突及解决过程描述

①生成的tiny.out文件无法执行[3]

查资料后发现.out文件是linux环境下的可执行文件，相当于windows环境下的.exe文件。不过这两种文件的内容是一致的，因此将文件后转改成.exe即可在windows下运行。

②bison 和 yacc[4]

两者是同一种工具，只不过bison是yacc的GNU版本，由于实验前在本机中安装的是bison，因此生成可执行文件的过程中使用的是bison命令而不是yacc命令。

③不能输入tiny直接执行tiny.exe文件

查阅了一下微软官方的控制台文档[5]，文档中对于这个问题的回答是将命令修改成.\tiny表示信任该可执行文件

参考资料：

[1] lex和yacc详解 <https://zhuanlan.zhihu.com/p/143867739>

[2] a guide to Lex & Yacc https://arcb.csc.ncsu.edu/~mueller/codeopt/codeopt00/y\_man.pdf

[3].out文件和.exe文件 <https://blog.csdn.net/weixin_39629075/article/details/110812050>

[4] Yacc和Bison <https://blog.csdn.net/gfdhjf/article/details/86654370>

[5]PowerShell执行新生成的可执行文件https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\_command\_precedence?view=powershell-7.2