**编译原理**

班级：23计算机科学与计算机

姓名：盲灯 学号：

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

针对输入：

**{ 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

1. Token类型的定义与共享

使词法分析器与语法分析器对token类型具有一致理解。

（1）.在globals.h中定义一个TokenType枚举类型，列出所有终结符，如IF, READ, ID, NUM 等。

(2).在tiny.y文件中使用%token指令定义这些token由Bison自动生成tiny.tab.h文件。

(3）.所有源文件如tiny.l, tiny.y, scan.c等通过#include "globals.h"和#include "tiny.tab.h" 实现token类型共享。

（4）.注意避免重复定义,需要统一使用tiny.tab.h中的token宏，或保证globals.h与其一致。

2. Token 值的传递方式通过yylval变量

将词法分析器识别出的token的“具体值”（如常数、变量名）传递给语法分析器。

(1).在tiny.y中定义YYSTYPE类型，指定yylval的类型,例如：#define YYSTYPE TreeNode \*。

(2).在tiny.l的规则中，为带值token如NUM, ID设置yylval：

{number} { yylval = newExpNode(ConstK); yylval->attr.val = atoi(yytext); return NUM; }

{identifier} { yylval = newExpNode(IdK); yylval->attr.name = copyString(yytext); return ID; }

(3).在tiny.y中使用$1, $2, $$等变量在语义动作中接收这些值并构建语法树。

3. 辅助信息如行号和token字符串的传递

记录当前 token 的字符串值如tokenString[]和源代码中的行号lineno，用于调试和错误信息。

(1).在 globals.h 中声明相关变量：

extern int lineno;

extern char tokenString[MAXTOKENLEN+1];

extern FILE\* source;

extern FILE\* listing;

extern int TraceScan;

(2).在main.c中对上述变量进行定义初始化：

int lineno = 0;

int TraceScan = TRUE;

FILE\* source;

FILE\* listing;

(3).在scan.c中封装getToken()函数，调用yylex()并复制yytext到tokenString，用于语法分析和调试输出：

TokenType getToken(void) {

TokenType token = yylex();

strncpy(tokenString, yytext, MAXTOKENLEN);

tokenString[MAXTOKENLEN] = '\0';

if (TraceScan) printToken(token, tokenString);

return token;

}

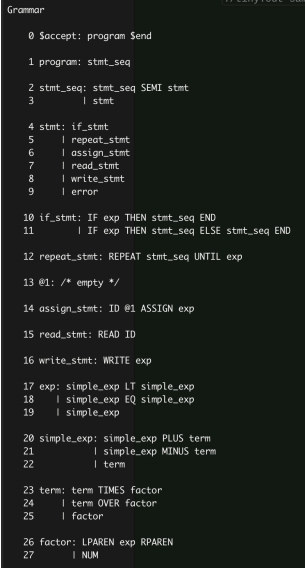
(4).在tiny.y中手动将yychar = yylex(); 修改为yychar = getToken();，以实现封装后的 token 读取和调试功能。

1. 分析表parsing table问题

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bison -v tiny.y获得parsing table



1. 内容和步骤

1、针对TINY语言给出 yacc的y文件的代码

%{

*#define YYPARSER*

*#include "globals.h"*

*#include "util.h"*

*#include "scan.h"*

*#include "parse.h"*

*#define YYSTYPE TreeNode \**

static char \* savedName;

static int savedLineNo;

static TreeNode \* savedTree;

extern int getToken(void);  // Flex中定义的词法函数

%}

%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

%%

program     : stmt\_seq                { savedTree = $1; }

            ;

stmt\_seq    : stmt\_seq SEMI stmt      { $$ = makeSibling($1, $3); }

            | stmt                    { $$ = $1; }

            ;

stmt        : if\_stmt

            | repeat\_stmt

            | assign\_stmt

            | read\_stmt

            | write\_stmt

            ;

if\_stmt     : IF exp THEN stmt\_seq END

              { $$ = newStmtNode(IfK);

                $$->child[0] = $2;

                $$->child[1] = $4; }

            ;

repeat\_stmt : REPEAT stmt\_seq UNTIL exp

              { $$ = newStmtNode(RepeatK);

                $$->child[0] = $2;

                $$->child[1] = $4; }

            ;

assign\_stmt : ID ASSIGN exp

              { $$ = newStmtNode(AssignK);

                $$->attr.name = copyString(tokenString);

                $$->child[0] = $3; }

            ;

read\_stmt   : READ ID

              { $$ = newStmtNode(ReadK);

                $$->attr.name = copyString(tokenString); }

            ;

write\_stmt  : WRITE exp

              { $$ = newStmtNode(WriteK);

                $$->child[0] = $2; }

            ;

exp         : simple\_exp

            | simple\_exp EQ simple\_exp

              { $$ = newExpNode(OpK); $$->attr.op = EQ; $$->child[0] = $1; $$->child[1] = $3; }

            | simple\_exp LT simple\_exp

              { $$ = newExpNode(OpK); $$->attr.op = LT; $$->child[0] = $1; $$->child[1] = $3; }

            ;

simple\_exp  : term

            | simple\_exp PLUS term

              { $$ = newExpNode(OpK); $$->attr.op = PLUS; $$->child[0] = $1; $$->child[1] = $3; }

            | simple\_exp MINUS term

              { $$ = newExpNode(OpK); $$->attr.op = MINUS; $$->child[0] = $1; $$->child[1] = $3; }

            ;

term        : factor

            | term TIMES factor

              { $$ = newExpNode(OpK); $$->attr.op = TIMES; $$->child[0] = $1; $$->child[1] = $3; }

            | term OVER factor

              { $$ = newExpNode(OpK); $$->attr.op = OVER; $$->child[0] = $1; $$->child[1] = $3; }

            ;

factor      : LPAREN exp RPAREN        { $$ = $2; }

            | NUM                      { $$ = newExpNode(ConstK); $$->attr.val = atoi(tokenString); }

            | ID                       { $$ = newExpNode(IdK); $$->attr.name = copyString(tokenString); }

            ;

%%

int yyerror(char \* message) {

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

  return 0;

}

TreeNode \* parse(void) {

  yyparse();

  return savedTree;

}

2、给出.l文件的代码

%{

#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;

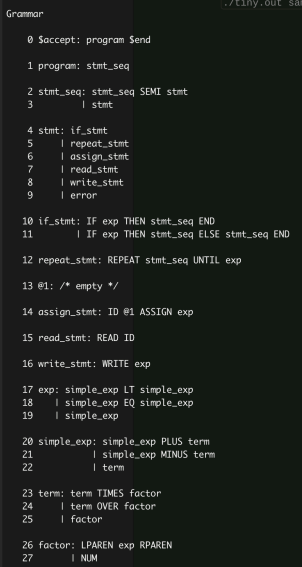
}

3、实验具体步骤

1. 所有的文件如图所示

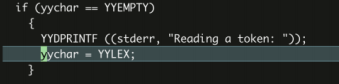


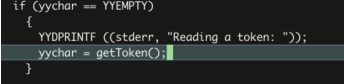
2.输入命令 bison -v tiny.y 生成 tiny.output LR 分析表，里面包含 LR(1) 状态、移进规约表、冲突等信息



3.在cmd中输入命令 lex tiny.l 和 yacc -d tiny.y，得到y.tab.c,y.tab.h和lex.yy.c这三个文件；lex.yy包含yylex()函数的实现，y.tab.c为语法分析器主代码，包含yyparse()实现，y.tab.h包含所有%token的符号宏定义，用于lexer调用

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

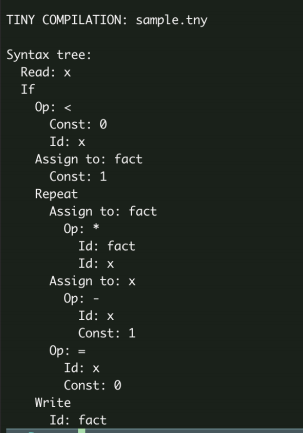




1. 输入 make 命令tiny.out: lex.yy.c y.tab.c main.c parse.c util.c scan.c

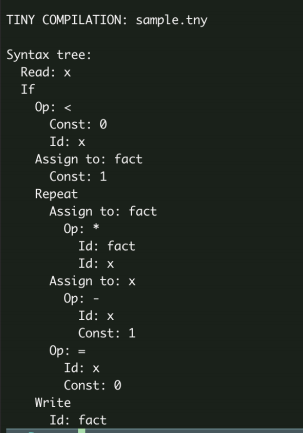
gcc -o tiny.out lex.yy.c y.tab.c main.c parse.c util.c scan.c -DYY\_NO\_UNISTD\_H进行构建 ，生成tiny.out可执行文件

1. 以sample.tny作为输入执行tiny.out文件，cmd中输如.\tiny.out < sample.tny，得到输出结果如下：

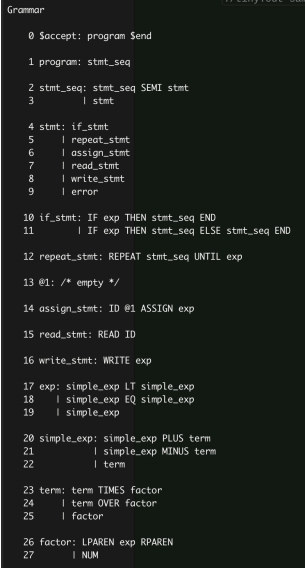


1. 实验结果:

1.输出结果如下：



2.分析表如下：



1. 实验结论:

1 、实验结论

本次实验成功构建了基于 YACC 的 TINY 语言语法分析器，能够对输入程序进行语法分析并输出对应的抽象语法树。同时，结合 Lex 工具完成了词法分析器的构建，实现了从 TINY 源程序到语法树的完整编译前端流程。通过 bison -v 成功生成并解析了 LR(1) 分析表，分析器能够正确识别语法结构、处理词法单元并输出结构清晰的语法树。

2、分析和总结

（1）借助Lex和YACC工具，只需按照其规定的描述格式编写规则，即可快速生成对应的词法分析器与语法分析器。这大大简化了编译器前端的开发工作，免去了手动设计状态机、书写语法分析算法等复杂过程。特别是在语义规则设计方面，YACC提供的嵌入式动作机制使得构建语法树更加高效可靠，适合教学初期和中等规模编译器的快速实现。

（2）手动编写词法和语法分析程序虽然工作量较大，但能更清晰地掌握编译器工作的底层流程，例如状态转移、Toke缓存、递归下降或自顶向下/自底向上的语法分析流程。在处理输入字符流、错误恢复机制以及语法树节点分配等方面都能获得更多实践经验，是深入理解编译原理的必要过程。

（3）使用Lex/YACC进行开发，更适合需要快速构建、可移植性强、结构清晰的标准化项目。而手写分析程序虽然复杂，但当需要实现非标准的语言结构、特殊的语法树格式、定制化的错误提示逻辑时，则可以提供更大的灵活性与自由度。两者各有侧重，应视实验目标而定。

3、实验中出现的冲突及解决过程描述

（1）词法分析器与语法分析器的 Token 冲突问题  
问题：在globals.h和 tiny.y中重复定义token类型，导致 YACC 报错“重定义符号”。  
解决：删去globals.h中对token的枚举定义，统一由tiny.tab.h自动生成并通过#include共享。

（2）词法分析函数调用冲突  
问题：YACC默认调用yylex()作为词法接口，而我们的词法分析函数是封装后的getToken()，导致分析器无法正确获取token。  
解决：在tiny.tab.c中将yychar = yylex(); 修改为yychar = getToken();，确保调用自定义词法封装函数。

（3）终端命令执行失败和文件缺失问题  
问题：命令行中使用gcc构建时提示找不到y.tab.c，实际是生成文件为tiny.tab.c。  
解决：统一命名规则，在Makefile中将目标文件替换为tiny.tab.c，并使用bison -d -o tiny.tab.c tiny.y 显式生成对应文件名。