一、实验目的

在这一阶段的实验中，同学们要完成Decaf编译器的语法分析工作，即用bison工具生成一个语法分析程序，对词法分析输出的单词符号串（终结符串）进行自底向上的分析，并依次输出用来进行归约的语法规则。

二、Decaf语言的语法

参见文档Decaf\_language.rtf。请大家注意，文档中给出的BNF描述并不能直接被YACC识别，而是需要进行适当的改写。请大家在做实验前，仔细研究一下我们给出的例子。

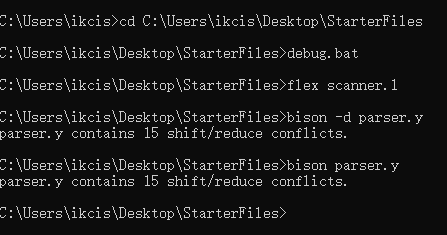
三、实验分析

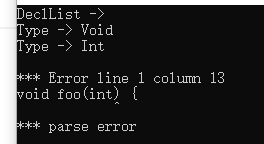
1、实验的主要过程是把C语言语法规则的巴科斯范式转化成YACC所能识别的文法，再判断实验1结果产生的token来是否符合文法。YACC使用BNF定义语法，能处理上下文无关文法，是向前查看、从左到右、最右(LALR(1))的自底向上分析方法。Bison是GNU版的YACC，与 YACC兼容，并且在YACC之上进行了改进。

2、实验中主要的难点是解决二义性和冲突。消除二义性有两条规则：(1)出现移进/归约冲突时，进行移进；(2)出现归约/归约冲突时，用先出现的规则进行归约。用结合性和优先级解决冲突，如果使用了％prec子句，则优先级和结合性由％prec子句决定；对于无优先级和结合性的规则，用规则1、2解决；对于有优先级和结合行的规则，用如下的规则解决：出现移进/归约冲突时，输入符号的优先级大于规则的优先级则移进，若输入符号的优先级小于规则的优先级则归约，若二者的优先级相同，左结合则归约，右结合则移进，无结合则出错。

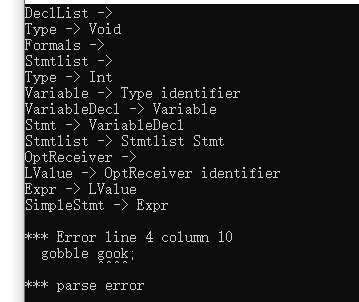
3、一旦出现规约/规约冲突，bison总会选择靠前的产生式；而一旦出现移入/规约冲突，则bison总会选择移入。%left、%right、%nonassoc可以对终结符的结合性进行规定，其中%left表示左结合，%right表示右结合，%nonassoc表示不可结合。

四、实验数据

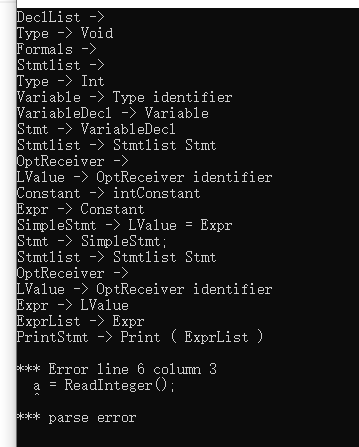




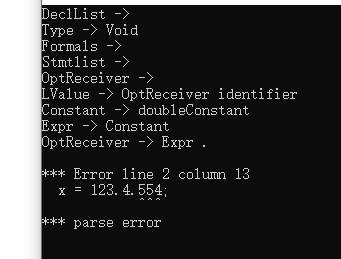
Bad1：与out一致



Bad2：与out一致



Bad3：与out一致

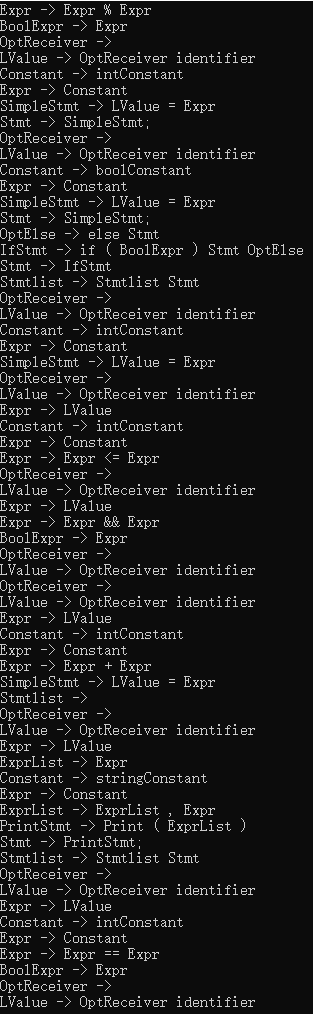


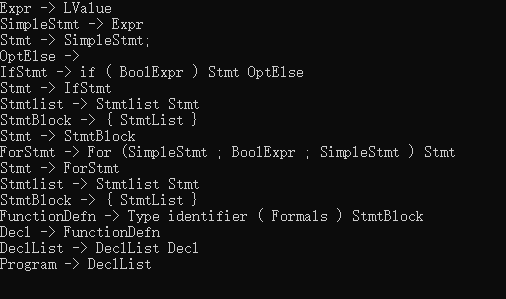
Bad4：与out一致



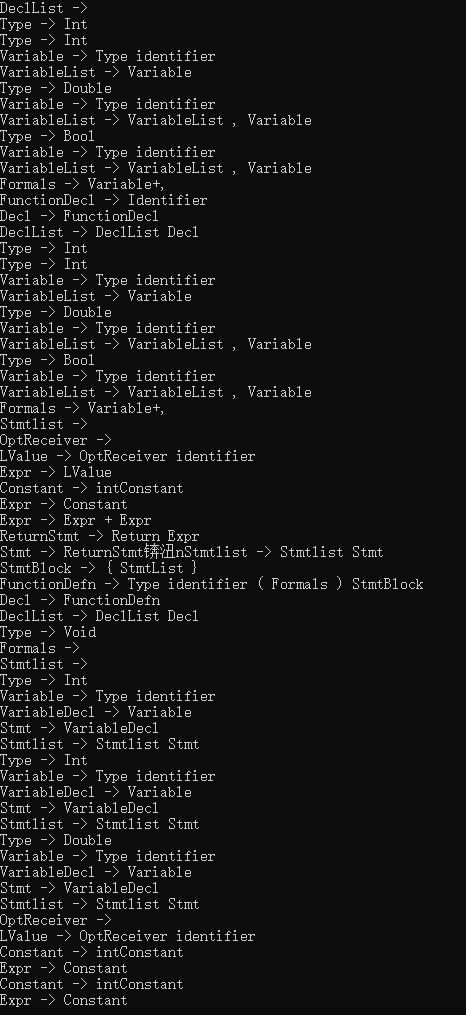
Class：与out一致

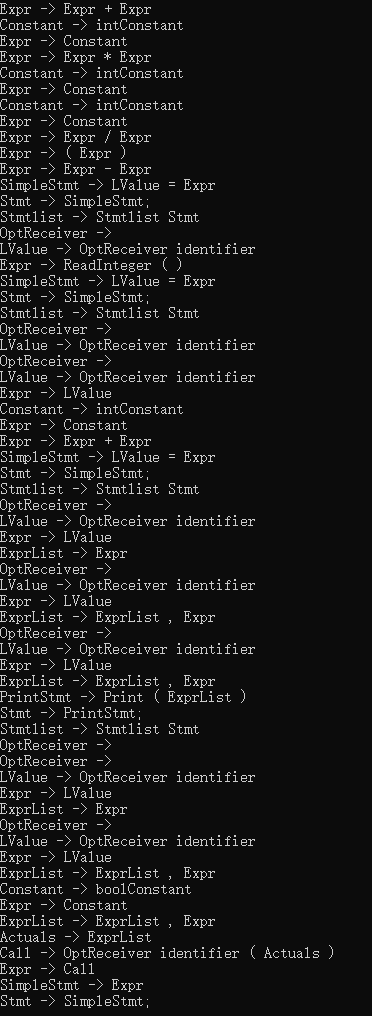


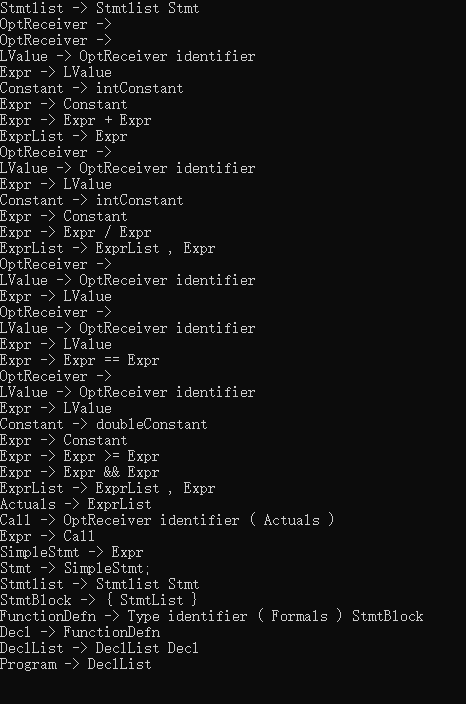




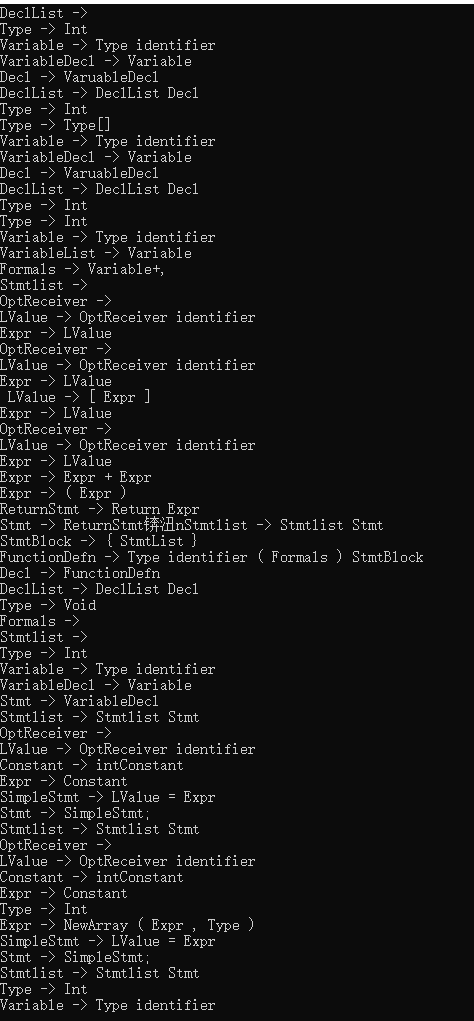
Control：与out一致

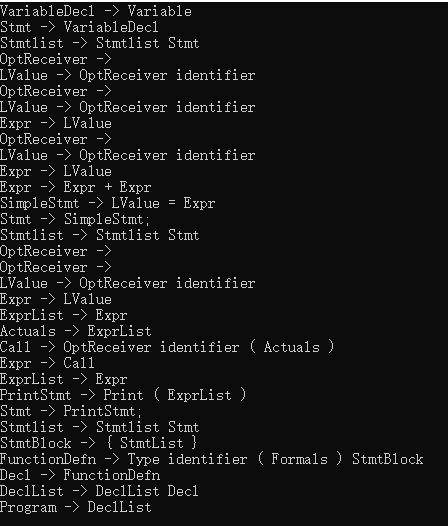






Expressions：与out一致





Functions：与out一致

五、实验心得

1、编写的parser.y文件存在15处移进/修约冲突，说明仍存在二义性和冲突。根据实验所给的sample，和自己所写的样例，没能找到冲突的来源，待解决。

2、bison在对语句进行分析时，会有一个递归的过程，当它分析到这个非终结符时，会递归到下一个语句，左值是上一式产生递归的终止符的那个语句。此时bison会把前一个非终止符先压入内部堆栈，接着切换到一个新的状态，这个状态会对应着刚刚压入内部堆栈的状态。

3、当移进终结符a和归约产生式A->a相冲突时，若产生式A->a的优先级高于a，或者两者优先级相同但产生式左结合时，则按A->a归约，否则移进a。产生式的优先级和结合性与产生式右部最右面的终结符的优先级和结合性相同。

4、%left，%right，%nonassoc的优先级关系则是以他们定义出现的顺序决定的，先定义的优先级低，最后定义的优先级最高，同时定义的优先级相同。

5、实验的大部分时间都在根据sample的out反推文法规则，一方面是因为文法的层层嵌套容易弄乱思路，另一方面是因为平常的学习好像只学会了要这么做，但是不明白为什么这么做，概念与概念间的逻辑很混乱。

六、实验文件

Parser.l

%{

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <process.h>

#include "scanner.h"

%}

%start Program

%token T\_Void T\_Bool T\_Int T\_Double T\_String T\_Class

%token T\_LessEqual T\_GreaterEqual T\_Equal T\_NotEqual

%token T\_And T\_Or T\_Null T\_Extends T\_This

%token T\_While T\_For T\_If T\_Else T\_Return T\_Break

%token T\_Identifier

%token T\_StringConstant T\_IntConstant T\_DoubleConstant T\_BoolConstant

%token T\_New T\_NewArray T\_Print T\_ReadInteger T\_ReadLine

%union

{

int integerConstant;

int boolConstant;

const char \*stringConstant;

double doubleConstant;

char identifier[128];

}

%right '='

%left '+' '-' '\*' '/' '%'

%nonassoc LOWER\_THEN\_ELSE

%nonassoc T\_Else

%left '<' '>'

%left '.'

%left '(' ')'

%left '{' '}'

%left '[' ']'

%right '^'

%left T\_And T\_Or

%left T\_LessEqual T\_GreaterEqual T\_Equal T\_NotEqual

%%

Program : DeclList

{ printf("Program -> DeclList\n"); @1 };

DeclList : DeclList Decl

{ printf("DeclList -> DeclList Decl\n");}

|

{ printf("DeclList ->\n");};

Decl : VariableDecl

{ printf("Decl -> VaruableDecl\n");}

| ClassDefn

{ printf("Decl -> ClassDefn\n");}

| FunctionDefn

{ printf("Decl -> FunctionDefn\n");}

| FunctionDecl

{ printf("Decl -> FunctionDecl\n");};

VariableDecl : Variable ';'

{ printf("VariableDecl -> Variable\n");};

VariableList : VariableList ',' Variable

{ printf("VariableList -> VariableList , Variable\n");}

| Variable

{ printf("VariableList -> Variable\n");};

Variable : Type T\_Identifier

{ printf("Variable -> Type identifier\n");};

Type : T\_Int

{ printf("Type -> Int\n");}

| T\_Double

{ printf("Type -> Double\n");}

| T\_Bool

{ printf("Type -> Bool\n");}

| T\_String

{ printf("Type -> String\n");}

| T\_Void

{ printf("Type -> Void\n");}

| T\_Class T\_Identifier

{ printf("Type -> class identifier\n");}

| Type '[' ']'

{ printf("Type -> Type[]\n");};

FunctionDecl : Type T\_Identifier '(' Formals ')' ';'

{ printf("FunctionDecl -> Identifier\n");};

FunctionDefn : Type T\_Identifier '(' Formals ')' StmtBlock

{ printf("FunctionDefn -> Type identifier ( Formals ) StmtBlock\n");};

Formals : VariableList

{ printf("Formals -> Variable+,\n");}

|

{ printf("Formals ->\n");};

ClassDefn : T\_Class T\_Identifier OptExtends '{' Fieldlist '}'

{ printf("ClassDefn -> class identifier OptExtends { FieldList }\n");};

OptExtends : T\_Extends T\_Identifier

{ printf("OptExtends -> extends identifier\n"); }

|

{ printf("OptExtends ->\n");};

Fieldlist : Fieldlist Field

{ printf("Fieldlist -> Fieldlist Field\n");}

|

{ printf("Fieldlist ->\n");};

Field : VariableDecl

{ printf("Field -> VariableDecl\n");}

| FunctionDecl

{ printf("Field -> FunctionDecl\n");}

| FunctionDefn

{ printf("Field -> FunctionDefn\n");};

StmtBlock : '{' Stmtlist '}'

{ printf("StmtBlock -> { StmtList }\n");};

Stmtlist : Stmtlist Stmt

{ printf("Stmtlist -> Stmtlist Stmt\n");}

|

{ printf("Stmtlist ->\n");};

Stmt : VariableDecl

{ printf("Stmt -> VariableDecl\n");}

| SimpleStmt ';'

{ printf("Stmt -> SimpleStmt;\n");}

| IfStmt

{ printf("Stmt -> IfStmt\n");}

| WhileStmt

{ printf("Stmt -> WhileStmt\n");}

| ForStmt

{ printf("Stmt -> ForStmt\n");}

| ReturnStmt ';'

{ printf("Stmt -> ReturnStmt；\n");}

| PrintStmt ';'

{ printf("Stmt -> PrintStmt;\n");}

| StmtBlock

{ printf("Stmt -> StmtBlock\n");};

SimpleStmt : LValue '=' Expr

{ printf("SimpleStmt -> LValue = Expr\n");}

| Expr

{ printf("SimpleStmt -> Expr\n");}

|

{ printf("SimpleStmt -> \n");};

LValue : OptReceiver T\_Identifier

{ printf("LValue -> OptReceiver identifier\n");}

| Expr '[' Expr ']'

{ printf(" LValue -> [ Expr ]\n");};

OptReceiver : Expr '.'

{ printf("OptReceiver -> Expr .\n");}

|

{ printf("OptReceiver ->\n");};

Call : OptReceiver T\_Identifier '(' Actuals ')'

{ printf("Call -> OptReceiver identifier ( Actuals )\n");};

Actuals : ExprList

{ printf("Actuals -> ExprList\n");}

|

{ printf("Actuals ->\n");};

ExprList : ExprList ',' Expr

{ printf("ExprList -> ExprList , Expr\n"); }

| Expr

{ printf("ExprList -> Expr\n"); };

ForStmt : T\_For '(' SimpleStmt ';' BoolExpr ';' SimpleStmt ')' Stmt

{ printf("ForStmt -> For (SimpleStmt ; BoolExpr ; SimpleStmt ) Stmt\n");};

WhileStmt : T\_While '(' BoolExpr ')' Stmt

{ printf("WhileStmt -> While ( BoolExpr ) Stmt\n");};

IfStmt : T\_If '(' BoolExpr ')' Stmt OptElse

{ printf("IfStmt -> if ( BoolExpr ) Stmt OptElse\n");};

OptElse : T\_Else Stmt

{ printf("OptElse -> else Stmt\n"); }

| %prec LOWER\_THEN\_ELSE

{ printf("OptElse ->\n"); };

ReturnStmt : T\_Return

{ printf("ReturnStmt -> Return\n");}

| T\_Return Expr

{ printf("ReturnStmt -> Return Expr\n");};

PrintStmt : T\_Print '(' ExprList ')'

{ printf("PrintStmt -> Print ( ExprList )\n");};

BoolExpr : Expr

{ printf("BoolExpr -> Expr\n");};

Expr : Constant

{ printf("Expr -> Constant\n");}

| LValue

{ printf("Expr -> LValue\n");}

| T\_This

{ printf("Expr -> this\n");}

| Call

{ printf("Expr -> Call\n");}

| '(' Expr ')'

{ printf("Expr -> ( Expr )\n");}

| Expr '+' Expr

{ printf("Expr -> Expr + Expr\n");}

| Expr '-' Expr

{ printf("Expr -> Expr - Expr\n");}

| Expr '\*' Expr

{ printf("Expr -> Expr \* Expr\n");}

| Expr '/' Expr

{ printf("Expr -> Expr / Expr\n");}

| Expr '%' Expr

{ printf("Expr -> Expr %% Expr\n");}

| '-' Expr

{ printf("Expr -> - Expr\n");}

| Expr '<' Expr

{ printf("Expr -> Expr < Expr\n");}

| Expr T\_LessEqual Expr

{ printf("Expr -> Expr <= Expr\n");}

| Expr '>' Expr

{ printf("Expr -> Expr > Expr\n");}

| Expr T\_GreaterEqual Expr

{ printf("Expr -> Expr >= Expr\n");}

| Expr T\_Equal Expr

{ printf("Expr -> Expr == Expr\n");}

| Expr T\_NotEqual Expr

{ printf("Expr -> Expr != Expr\n");}

| Expr T\_And Expr

{ printf("Expr -> Expr && Expr\n");}

| Expr T\_Or Expr

{ printf("Expr -> Expr || Expr\n");}

| '!' Expr

{ printf("Expr -> ! Expr\n");}

| T\_ReadInteger '(' ')'

{ printf("Expr -> ReadInteger ( )\n");}

| T\_ReadLine '(' ')'

{ printf("Expr -> ReadLine ( )\n");}

| T\_New '(' T\_Identifier ')'

{ printf("Expr -> New ( identifier )\n");}

| T\_NewArray '(' Expr ',' Type ')'

{ printf("Expr -> NewArray ( Expr , Type )\n");};

Constant : T\_IntConstant

{ printf("Constant -> intConstant\n");}

| T\_DoubleConstant

{ printf("Constant -> doubleConstant\n");}

| T\_BoolConstant

{ printf("Constant -> boolConstant\n");}

| T\_StringConstant

{ printf("Constant -> stringConstant\n");}

| T\_Null

{ printf("Constant -> null\n");};

%%

void Inityyparse(void)

{

}