CS39003 Compiler Lab
Assignment 9

Intermediate Code Generation

Submission Deadline: 17th Nov   Marks: 30

Consider the following context-free grammar. It is a modified form of the grammar provided in Assignment 8.

Expression	Symbols in Grammar
and	AND
:=	ASSIGN
:	COLON
,	COMMA
def	DEF
else	ELSE
/	DIV
End	END
boolean	BOOLEAN
=	EQ
float	FLOAT
>=	GE
global	GLOBAL
>	GT
if	IF
int	INT
(	LEFT_PAREN
<=	LE
<	LT
-	MINUS
%	MOD
*	MULT
<>	NE
not	NOT
null	NUL
or	OR
+	PLUS
print	PRINT
read	READ
return	RETURN
)	RIGHT_PAREN
;	SEMICOLON
while	WHILE

Non-terminals

prog declList decl typeList varList var sizeListO sizeList type stmtListO stmt assignmentStmt dotId readStmt printStmt ifStmt elsePart whileStmt returnStmt expO ID bExp relOp exp

Start symbol

```
prog
```

Production Rules

```
prog --> GLOBAL declList stmtListO END
declList --> decl declList
         --> epsilon
decl --> DEF typeList END
typeList --> typeList SEMICOLON varList COLON type
         --> varList COLON type
varList --> var COMMA varList
        --> var
var --> ID
type --> INT
     --> FLOAT
     --> BOOLEAN
     --> NULL
stmtListO --> stmtList
          --> epsilon
stmtList --> stmtList SEMICOLON stmt SEMICOLON
         --> stmt
stmt --> assignmentStmt
     --> readStmt
     --> printStmt
     --> ifStmt
     --> whileStmt
assignmentStmt --> ID ASSIGN exp
readStmt --> READ FORMAT exp
printStmt --> PRINT FORMAT exp
ifStmt --> IF bExp COLON stmtList elsePart END
elsePart --> ELSE stmtList
         --> epsilon
whileStmt --> WHILE bExp COLON stmtList END
expO --> exp
     --> epsilon
Exp --> bExp OR bExp
     --> bExp AND bExp
    --> NOT bExp
    --> LEFT_PAREN bExp RIGHT_PAREN
    --> exp relOP exp
relOP --> EQ
      --> LE
      --> LT
      --> GE
      --> GT
      --> NE
exp --> exp PLUS exp
```

```
    --> exp MINUS exp
    --> exp MULT exp
    --> exp DIV exp
    --> exp MOD exp
    --> exp DOT exp
    --> LEFT_PAREN exp RIGHT_PAREN
    --> ID
    --> INT_CONST
    --> FLOAT_CONST
```

Operator precedence for the generated language is: {+–} < {* / %} < { . }

Write a three-address code generator for the programming language corresponding to the grammar mentioned above using *Lex* and *YACC*. The code of the source file is split into tokens (Lex) and the hierarchical structure of the program is established (YACC). The text file containing the equivalent three-address quadruple representation for a given source code should be reported. The "semantic action" section of the YACC specification file should be appropriately used for the generation of the equivalent three-address code.

Few important points to note

1. The declaration of the variables should be handled by creating a symbol table which should incorporate the type information of individual identifies and literals. The temporary variables should also be inserted in the symbol table.
2. In case of type mismatch, translator must perform a type conversion following the specified rules. The rule is, "If an expression consists of float and integer variables/constants, the expression must be evaluated in float." That implies, integers must be converted to float. For any other kind of type mismatch, report error. The **explicit** type conversion must be done.
3. If you are unable to handle the aforementioned type conversion, in that case you must report error if ANY kind of type mismatch occurs in expressions (simpler version). However, this will deduct some credit.

Symbol table format

ID Name (lexeme)	Type	Offset (relative address)

You have to submit three source files: `<group-no>.9.l`, `<group-no>.9.y` and `Makefile` in a tar archive. The `Makefile` should produce an executable file (`tac`), which produces symbol table (`symtab.txt`) and a text file (`output.txt`) containing the quadruple representation of the statements of the input source code (`input.txt`) if it is executed using the command *`./tac < input.txt`*

APPENDIX

Few sample three-address instructions

- **x := y binop z** where **binop** is one of: +, -, *, /, **and**, **or**
- **goto L**
- **if x goto L, ifFalse x goto L** where x is a Boolean variable.
- **if x relop y goto L** where **relop** is one of <**,** <=**,** =**,** >=**,** >**,** <>
- **param x** precedes call operation, passing a parameter
- **call p** procedure call, no return value
- **x := funcall f** function call, **x** is assigned the value returned by **f**
- **t1=(float) t2** if t2 is an integer, it converts t2 to a float t1.

Input code 1

```
global
def
        a:int;
        b:int;
        sum:float;
        xpos:float;
        ypos:float;
        temp: float
end

a:=1;
sum:=0.0;
xpos:=2.331;
temp:=1.5;
ypos:=sum+temp;
read %d b;
```

```
if b = 0 :
        print %f xpos;
else
        print %f ypos;
end
while a < b:
        a:= a * 2;
        sum := sum + temp;
end
print %f sum;
end
```

Input code 2

```
global
def
        a:int;
        b:int;
        sum:float;
        xpos:float;
        ypos:float
end

a:=1;
sum:=0.0;
xpos:=2.331;
ypos:=sum+a;
read %d b;

if b = 0 :
        print %f xpos;
else
        print %f ypos;
end
while a < b:
        a:= a * 2;
        sum := sum + 1;
end
print %f sum;
end
```

Input code 3

```
global
def
        a:int;
        b:int;
        sum:float;
        xpos:float;
        ypos:float;
        temp: boolean
end

a:=1;
sum:=0.0;
xpos:=2.331;
temp=1;
ypos:=sum+a+temp;
read %d b;

if b = 0 :
        print %f xpos;
else
        print %f ypos;
end
while a < b:
        a:= a * 2;
        sum := sum + 1;
end
print %f sum;
end
```