Experiment #3 Generating Three-address Code for Assignment, Conditional Statements and functions

Assignment Due: 07/03, 11:59pm

- **To do this assignment, you *may* work in teams of two. Team members must be from the same lab group.**
- **Submit the assignment in piazza as a private message:**
    - **Title: Lab Experiment 03_roll01_roll02_(Odd/EVEN)**

Assignment Details:
In this assignment we build up on the solution to assignment 2 where you generated 3AC for assignment statements. Here we will look into conditional statements and function or subprogram calls for language X. You can find the informal reference manual of the programming language X with some example programs **here**. Please read the instructions carefully.

A. Grammar of Language X

You can find the CFG of the programming language X **here**. In this assignment you are to generate intermediate code (i.e., 3AC) of every type of statement supported by **X** as an output of your compiler. As you can see in production rule 3 below there are 3 different types of statements.

stmt → ϵ | selection | iteration | assignment

You will find detail explanation and example of selection, iteration and subprogram calls in **here** (page 4-7). They correspond to if-else, while loops and function calls. For this assignment you will have to take multiple lines of inputs. Your program cannot terminate if there is an error in a line. It should try to parse to the end of the code. *For that simply error recovery methods need to be added.*

As can be seen from rules 1 and 2 below a program in X is basically a collection of statements separated by a semicolon (;).

program → stmts |eof

stmts → stmt | stmts ; stmt

Finally from rule 9 defines subprogram calls

```
assignment → vars := exprs          // done in assignment 2
 |vars := subprogram := exprs       // takes in a list of expr as parameter
                                    //and returns a list of variables
 |:= subprogram := exprs            // takes in a list of expr as parameter
                                    //and does not return anything
 |vars := subprogram :=             //no parameters but returns variables
 |:= subprogram :=                  // no parameter and no arguments
```

So far in assignment 2 you have only worked with the first type of assignment statements namely vars := exprs. The next 4 rules above show the ways you can call a subprogram with or without parameter, return values. We will assume only pass by value parameter passing method. *This grammar does not specify subprogram definitions. We will add productions rules for that as an extension to this assignment.*

B. Three-address Code

Examples of Three-address code (often abbreviated to TAC or 3AC) segments corresponding to different types of statements, written in a C like language, are shown below. For the list of available 3AC code statements see the description of assignment 2.

Control Flow Statements

Code in C	3AC

```
if (x<y)
    z=x;
else
    z=y;
z=z*z
```
	```
	 t0 = x < y
	 if t0 goto L0
	 goto L1
	L0:
	 z=x
	 goto L2
	L1:
	 z=y
	```

	L2:
	t2 = z*z
	z= t1

while (x < y) {
x = x * 2;
}
y = x;
	L0:
	t0 = x < y
	IfFalse t0 goto L1
	x = x * 2
	Goto L0
	L1:
	y = x

SimpleFunction(137, x+2);	t0 = 137 t1 = x + 2 Param t0 Param t1 Call SimpleFn, 2 //two arguments

void myfun()
{
int b;
int a;
b = 3;
a = 12;
a = (b + 2)-(a*3)/6;
return a;
}
	myfun:
	BeginFunc
	t0 = 3
	b = t0
	t1 = 12
	a = t1
	t2 = 2
	t3 = b + t2
	t4 = 3
	t5 = a * t4
	t6 = 6
	t7 = t5 / t6
	t8 = t3 - t7
	a = t8
	return a
	EndFunc

C. Implementation:

You will have to write a LEX/FLEX and YACC/Bison specifications in this assignment. In YACC or Bison you have to put down the corresponding CFG productions of each line of the syntax description. You will have to decide which symbols/variables in the language specification you want to define in the lexical analyzer and return to the parser, and which ones you want to be matched in YACC/Bison. In order to generate 3ACs

(i) You may need to store the 3AC code generated at different grammar variables. The types of different grammar symbols must be appropriately declared. Try to adopt the syntax directed code generation schemes discussed in class.

(ii) For error in a specific line, it's not acceptable to have the program terminated. You have to implement simple error recovery mechanism, some related discussions can be found **here** and also **here**. (Initially you can skip this part)

Sample Input # 1
 	
if x < y ? x := y-1
:: x > y ? x := y+1
fi
	t0= x <y
	if t0 goto L0
	goto L1
	L0:
	  t1 = y-1
	  x = t1
	 goto L2
	L1:
	  t2= x >y
	  if t2 goto L3
	  goto L2
	L3:
	  x =y +1
	L2:

Sample Input # 2
 	
do x < y ? x := x+7.111
od
	L0:
	  t0= x<y
	  ifF t0 goto L1
	  t0 = x+7.111
	  x = t0
	  goto L0
	L1:

Sample Input # 3
 	
r, y := myfun := 3.14, 37, false
	t1 = 3.14
	t2 = 37
	t3 = false
	param t1
	param t2
	param t3
	call myfun, 3, 2
	return r
	return y

Sample Input # 4 i x < y ? x := y-1	Error here t1 = x < y Error here

fi
z=z*z
	t2 = y-1
	x= t2
	t0 = z * z
	z = t0