University of Toronto
Faculty of Arts and Science
April 2012 Examinations
CSC488H1S / CSC2107HS
Duration – 2 hours (120 minutes)
OPEN BOOK ALL written aids, books and notes are allowed.
ALL non-programmable calculators allowed.
NO other electronic aids allowed.

120 marks total, 8 Questions on 4 Pages. ANSWER ALL QUESTIONS
Write all answers in the Exam book.
You must receive a mark of 35% or greater on this final exam to pass the course.

WRITE LEGIBLY Unreadable answers cannot be marked.
Line/rule reference numbers on the left side of of programs and grammars are provided for ease of reference only and are not part of the program or grammar .
The notation **. . .** stands for correct code that has been omitted for brevity
State clearly any assumptions that you have to make to answer a question.

1. [10 marks] Discuss the compiler implementation issues/benefits for each of these programming language restrictions.

a) All array bounds must be compile time constants.

b) Procedure and function declarations may not be nested.

c) All parameters are passed by value.

d) Set types can contain at most 32 distinct values.

e) **goto** statements may not branch out of functions or procedures.

2. [10 marks] Assume P, Q R and S are Boolean variables and I and J are integer variables. Show the branching code that would be generated for the Boolean expression:

P **and not** (Q **or** I <= J and R **not**= Q) **or** Q **and not** S

3. [20 marks] The **for** statement in the Algol 60 programming language has the syntax:

```
1    forStatement  →  for variable ':='  forList  do statement
2    forList       →  forSpec
3                  →  forList ','  forSpec
4    forSpec       →  expression
5                  →  expression  while  expression
6                  →  expression  step  expression  until  expression
```

For example the program fragment:

```
1    for i := 3 , 7 ,
2        11 step 1 until 16 ,
3        i ÷ 2 while i >= 1 ,
4        2 step i until 32 do
5    print( i );
```

will print 3 7 11 12 13 14 15 16 8 4 2 1 2 4 8 16 32

Design a translation scheme for this general form of **for** statement (similar to the translation scheme for the C **for** statement in the lecture slides). Show how each of the *forSpec*s will be implemented in the general case. Show how *forList*s will be implemented in the general case.

4. [20 marks] The Turing programming language has a **forward** declaration to allow procedure and function headers to be declared before the actual declaration of the procedure or function. The purpose of this declaration is to make it easier to write mutually recursive procedures and functions. Example:

```
forward procedure P ( i : Integer , p : Boolean )
forward function F ( n : Integer ) : Integer
/* Actual declarations of P and F will occur later. */
```

Design a set of semantic analysis checks that will verify that this construct is being used correctly.
a) what happens at the **forward** declaration?
b) what happens at the actual declaration ?
c) are any other checks required?
You can describe your answer as a set of **Sxy** operations similar to those used in the semantic analysis handout. Be very clear about what each **Sxy** does and where it is used.

5. [20 marks] Describe the classical optimizations that a good optimizing compiler would perform on the following code fragment. Assume the size of a **double** is 8 bytes with mod 8 alignment. You may show only the final optimized version if you are very clear about what optimizations have been performed.

```
 1    var A[ 0 : 50  , 0: 50  , 0 : 50 ]  , B[ 0 : 50  , 0 : 50  , 0 : 50 ]  : double
 2    var i , j , k  , n  : integer
 .
 .    /* Assume A and B get values here */
 .
30    n := 48
31    for i := 1 to n do
32        for j := n to 1 by -1 do
33            for k := 1 to n + 1 do
34                A[ i , j , k ] := A[ i , j - 1  , k - 1 ] + A[ i - 1  , j  , k ]
35                B[ i , j - 1  , k ] := A[ i , j - 1  , k -1 ] * 2.0
36                A[ i , j , k + 1 ] := B[ i  , j , k ] + 1.0
37            endfor
38        endfor
39    endfor
```

6. [10 marks] Assume you want to add *run time* subscript checking to your implementation of the course project language. Describe how you would do this. What changes would be required during semantic analysis? Give a code generation template for subscripting with subscript range checking.

7. [10 marks] A proposal has been made to change the syntax of the course project language to allow declarations to be intermixed with statements, but still require strict *declaration before use* for declared identifiers. Something like:

```
1    superStatement :    statement ,
2                        declaration ,
3                        superStatement statement ,
4                        superStatement declaration ;
5    scope :             '{' superStatement '}' ,
6                        '{'     '}' ;
```

How would this change affect semantic analysis? Would it introduce any major problems?

8.[20 marks] Describe the semantic analysis checks that a good Java compiler would perform on the Java code shown below.

```
 1    public class GeneratePrimeNumbersExample {
 2         public static void main(String[] args) {
 3                 //define limit
 4                 int limit = 100;
 5                 System.out.println("Prime numbers between 1 and " + limit);
 6                 //loop through the numbers one by one
 7                 for(int i=1; i < 100; i++){
 8                      boolean isPrime = true;
 9                      //check to see if the number is prime
10                      for(int j=2; j < i ; j++){
11                           if(i % j == 0){
12                                isPrime = false;
13                                break;
14                           }
15                      }
16                      // print the number
17                      if(isPrime)
18                             System.out.print(i + " ");
19                 }
20         }
21    }
```