University of Toronto
Faculty of Arts and Science
April 2010 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. [20 marks] Several programming languages include a **for** statement of the form:

```
for v := e1 , e2 , e3 , ... en do
      /* body of for loop */
end for
```

Where v is a previously declared scalar variable, and each of the e_i is a scalar expression. If there are n expressions in the list, the loop body is executed n times with the control variable v taking on successive values e_1, e_2 ... e_n . For example the loop:

```
for prime := 2 , 3 , 5 , 7 , 11 do
      put prime , " is prime"
end for
```

would print the first five prime numbers.

a) Describe the static semantic analysis checking that will be required for this statement in the general case.

b) Design a code generation template for this statement, i.e. show the code that might be generated for this statement in the general case.

2. [10 marks] The grammar below is LL(k) for some value of k.
Determine k . Justify your answer.

```
1        Z  →  X
2        X  →  Y
3           →  b  Y  a
4        Y  →  c
5           →  c  a
```

3. [20 marks] Show the Abstract Syntax Tree that would be generated for the program fragment in the project language shown below:

```
1     boolean function F ( integer i, integer j )
2         begin
3             boolean p
4             p  <= true
5             integer k, l
6             k  <= i
7             l  <= j
8             while k  >= 0 and l  >= 0 do
9                 p  <= k  > l
10                k  <= k - 2
11                l  <= l - 3
12            end
13            return ( p )
14        end
```

4. [10 marks] Describe the programming language implementation issues raised by each of the programming language features listed below

a) Allowing variables, types, constants, procedures and functions to be declared after they are used

b) An unrestricted **goto** statement that can branch to any visible label including out of loops and out of functions and procedures.

c) Allowing function/procedure overloading, i.e. allowing more than one than one function/procedure with the same name as long as they have different parameter lists

d) Allowing the size of arrays to be determined by *run time* expressions.

e) Allowing user defined overloading of existing operators (as in C++)

5. [10 marks] The programming language used in the course project has been extended with two new statements:

return from name
return (expression) from name

These statements have the same meaning as the original return statements *except* that *name* is the name of the function or procedure being returned from. These statements cause a return from the most recent (i.e. closest in the chain of outstanding calls) invocation of *name*.

a) discuss the implementation issues raised by these statements

b) what static semantic checking should be done on these statements?

c) What dynamic semantic checking should be done on these statements?

d) Describe how these statements might be implemented at runtime. You don't have to show detailed instruction sequences, but you should discuss the runtime algorithms that will be used.

6. [10 marks] Let P, Q, R and S be boolean variables and I, J, K be integer variables. Show the branching code that would be generated for the boolean expression:

P or (Q and I < J or K > I + J) or not (R and S or P)

7. [20 marks] Describe the classical optimizations that a good optimizing compiler would perform on the fragment of C code shown below. You may assume that int and float use 4 bytes of storage and that double uses 8 bytes. You can show only the final results of the optimization if you make it clear exactly what optimizations have been performed.

```
10      float A[ 100 ] , C[ 200 ] ;
11      double B[ 100 ] [ 200 ] ;
12      int I , J , K ;

        /* Assume the A, B and C are given values here */

50      for( I = 0 ; I < 200 ; I++ )
51          B[ 0 ][ I ] = C[ I ] * C[ I ] ;
52      B[ 0 ][ 0 ] = A[ 0 ] ;
53      for( I = 1 ; I < 100 ; I++ )
54          B[ I ][ 0 ] = A[ I ] * A[ I - 1 ] ;
55      for( I = 1 ; I < 100 ; I++ )
56          for( J = 1 ; J < 200 ; J++ )
57              B[ I ][ J ] = B[ I - 1][ J ] * B[ I ][ J -1 ] ;
```

8.[20 marks] A switch expression is an extension of the conditional expression in C and C++ . switch expressions may be used anywhere that an expression can be used. It has the general form:

switch $expn_s$ ($labels_1$: $expn_1$ **,** $labels_2$: $expn_2$ **, . . . ,** $lables_n$: $expn_n$)

Where $lables_k$ is a comma separated list of scalar constants (integer, char, enum). One $labels_k$ can be empty denoting the default case. The values in the labels lists must be disjoint. $expn_s$ is the control expression. It is evaluated once and its value is matched against the values of each list of labels. If a match is found (e.g. $labels_m$) then the value of the construct is the value of the corresponding expression (e.g. $expn_m$). If there is no match and there is a default, then the value of the default expression is the value of the construct. If there is no match to any labels and no default is present then in the tradition of C the result is undefined and the value of the construct is random garbage.

a) Describe the static semantic checks that should be performed on this construct.

b) Write a recursive descent parser for this construct. You may assume the existence of other parser functions as long as you document your assumptions in your answer. Assume that your switch parser will be called when the expression parser recognizes the reserved word **switch**

Examples:

isPrime = **switch** p (2,3,5,7,11 : 1 , 1,2,4,6,8,9,10,12 : 0 , : 0)

message = **switch** eCode + 1 (0 : "" , 1 : "Warning" , 2 : "Error" , : "Internal Error")