University of Toronto
Faculty of Arts and Science
April 2013 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 5 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] Given the declarations:

integer I , J , A[100]
boolean P, Q, R [50]

Show the branching code (i.e quadruples) that would be generated for the boolean expression:

! P & (Q | A[I] > 12) & R[J] != Q | (P & ! Q)

2. [10 marks] Assume that the **exit** construct in the project language was replaced by a more general version:

exit *expression*
exit *expression* **when** *booleanExpression*

Where *expression* is an arbitrary integer expression i.e. the number of loops to exit is determined at runtime.
Describe how you would implement this enhanced version of the exit statement.

3. [10 marks] In programming languages that include *value-result* parameters the mode of parameter passing is often confused. In particular value-result parameters are sometimes used in a way that suggests that reference mode was intended. An indication of this problem is an assignment to a value-result parameter before the value of the parameter has been used. Show how data flow analysis could be used to identify value-result parameters that are defined (i.e. assigned to) before they are used.

4. [10 marks] The representation of information is evolving toward the use of meta languages like XML as a standard encoding format. Suppose that this evolution has gone forward so that even text files containing source programs are now encoded in XML. For example, the program fragment:

```
if ( X != 13 ) {
   Y = 12 ;
   Z = X -1 ;
}
else
   X++ ;
```

Might be encoded as

```
<RESERVED>if</RESERVED> <PARENTHESIS> <IDENTIFIER>X</IDENTIFIER>
<OPERATOR>!=</OPERATOR> <INTEGER>13</INTEGER> </PARENTHESIS>
<BRACKET> <IDENTIFIER>Y</IDENTIFIER> <OPERATOR>=</OPERATOR>
<INTEGER>12</INTEGER> <OPERATOR>;</OPERATOR> <IDENTIFIER>Z</IDENTIFIER>
<OPERATOR>=</OPERATOR> <IDENTIFIER>X</IDENTIFIER> <OPERATOR>-</OPERATOR>
<INTEGER>1</INTEGER> <OPERATOR>;</OPERATOR> </BRACKET>
<RESERVED>else</RESERVED> <IDENTIFIER>X</IDENTIFIER>
<OPERATOR>++</OPERATOR> <OPERATOR>;</OPERATOR>
```

Describe how you would design a compiler scanner to process source programs that are encoded in this way.
Emphasize what would be *different* from the traditional scanner that was discussed in lecture.

5. [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 *recursive descent* parsing function for this **for** statement to be used as part of an Algol 60 compiler. You may assume:

- your parser is called when the statement level parser encounters the reserved word **for**
- that there is a function **boolean function** variable ( )
  that recognizes variables and returns true if it is successful.
- that there is a function **boolean function** expression ( )
  that recognizes expressions and returns true if it is successful
- that there is a function **boolean function** statement ( )
  that recognizes statements and returns true if it is successful.
- that the next incoming lexical token is in a variable named *nextToken* and that the function **boolean function** getToken() advances the lexical input stream by one lexical token. It returns true if it was successful.

6. [20 marks] Describe the machine independent optimizations that a good optimizing compiler would perform on the C function listed below. You can show only the final result if you describe *very clearly* exactly which optimizations have been performed. Assume variables of type **double** are stored in 8 bytes.

```
 1    #define N ( 100 )
      ...
20    void G ( double A[ N ][ N ] , double B[ N ] , double X[ N ] ) {
21        double W[ N ][ N+1 ] , M ;
22        int I , J , K ;
23        for( I = 0 ; I < N ; I ++ ) {
24            for( J = 0 ; J < N ; J ++ )
25                W[ I ][ J ] = A[ I ][ J ] ;
26            W[ I ][ N ] = B[ I ] ;
27        }
28        for( I = 0 ; I < N ; I ++ )
29            for( J = 0 ; J < N ; J ++ )
30                if ( J != I ) {
31                    M = W[ J ][ I ] / W[ I ][ I ] ;
32                    for( K = 1 ; K <= N ; K ++ )
33                        W[ J ][ K ] -= M * W[ I ][ K ] ;
34                }
35        for( I = 0 ; I < N ; I ++ )
36            X[ I ] = W[ I ][ N ] / W[ I ][ I ] ;
37    }
```

7. [20 marks] Numerical analysts often take advantage of symmetries in a problem to make more efficient use of memory to store large amounts of data. One well known technique is the use of upper triangular storage for symmetric matrices.

A matrix $A[N,N]$ is *symmetric* if and only if $\forall I,J \quad A[I,J] = A[J,I]$
Since the information above and below the diagonal is the same, it is only necessary to store the diagonal and the upper triangular region above the diagonal.

Assume you are implementing a Numerical Analyst friendly extension to C in which upper triangular matrices are built in data types, e.g. a programmer might declare:

doubleupper A[100][100] ;

Design an implementation of upper triangular matrices
a) show how the matrices are laid out in memory.
b) based on your memory layout, give an algorithm for calculating the address of an arbitrary array element $B[E_1][E_2]$ where B is an MxM doubleupper matrix and E_1 and E_2 are integer expressions.

8.[20 marks] A proposal has been made to extend the course project language to allow arrays to be passed as parameters to procedures and functions. The syntax changes are:

```
1    parameters:   type parametername ,
2                  type arrayname [  ] ,
3                  parameters , parameters

4    arguments:    expression ,
5                  arrayname ,
6                  arguments, arguments
```

Describe how this extension could be implemented
- what new semantic analysis checks would be required ?
- Show code templates (i.e. pseudo machine instructions) to describe
    - how arrays would be passed as arguments
    - how parameter arrays are accessed inside a function/procedure

Example

```
 1    begin
 2        integer Sum( integer src[  ] , integer from , integer to )
 3            begin
 4                integer index , total
 5                total := 0
 6                index := from
 7                loop
 8                    exit when index > to
 9                    total := total + src[ index ]
10                    index := index + 1
11                pool
12                result total
13                end % Sum
14
15        integer A[ -100 .. 100 ] , S
16        % Assume A gets a value here
17        S := Sum( A , 0 , 100 )
18    end
```