University of Toronto
Faculty of Arts and Science
April/May Examinations 2002
CSC488H1S / CSC2107H1S
Duration – 2 hours (120 minutes)

120 marks total, 7 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.

OPEN BOOK ALL written aids, books and notes are allowed.
ALL non-programmable calculators allowed. NO other electronic aids allowed.

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

1. [20 marks] Describe the symbol and type table entries that a typical compiler might make for these declarations:

```
 1      #define  NSTUDENTS   ( 50 )
 2      #define  MAXNAME     ( 30 )
 3      #define  NUGASSIGN   (  6 )
 4      #define  NGRASSIGN   (  5 )
 5      typedef enum{ undergrad, graduate, special, audit } studentType ;
 6      typedef unsigned char  smallInt ;
 7      struct {
 8           char        name[ MAXNAME ] ;
 9           char        studentNumber[ 10 ];
10           smallInt    team ;
11           studentType sType ;
12           union {
13                 smallInt  ugAssignments[ NUGASSIGN ], ugMidTerm, ugFinal ;
14                 smallInt  grAssignments[ NGRASSIGN ], grProject, grFinal ;
15                 smallInt  spProject, spFinal ;
16            } marks ;
17            float rawMark, courseMark ;
18      } class[ NSTUDENTS ] ;
```

2. [15 marks] In some functional languages (e.g. Lisp) the association between a *free variable* in a function and its location is determined *dynamically* by tracing back along the current call chain. An example:

```
function addX( int Y ) : int
     return  X + Y ;
end addX
```

X is a free variable in addX since it is not declared as a local variable in the function. When addX is called, the value for X is determined by tracing back along the chain of function calls leading up to the call of addX until a function with a local variable named X is found.
Describe a set of compile time and run time mechanisms sufficient to implement this dynamic binding of free variables to values.

3. [20 marks] Describe the optimizations that a good optimizing compiler would perform on the C function given below. You can show *only* the final result if you describe *very clearly* exactly which optimizations have been performed.

```
#define  N   (100)
. . .
void G( double A[N][N], double B[N], double X[N] ){
    double W[N][N+1] , M;
    int I, J, K ;
    for( I = 0 ; I < N ; I++ ) {
       for( J = 0 ; J < N ; J++ )
           W[I][J] = A[I][J] ;
       W[I][N] = B[I]
    }
    for( I = 0 ; I < N ; I++ )
       for( J = 0 ; J < N ; J++ )
           if( J != I ) {
               M = W[J][I] / W[I][I] ;
               for( K = I ; K <= N ; K++ )
                   W[J][K] -= M * W[I][K] ;
           }
    for( I = 0 ; I < N ; I++ )
        X[I] = W[I,N] / W[I][I] ;
}
```

4.[15 marks] Assume that you wanted to improve the CSC488S project compiler by providing *source program* line numbers in *runtime* error message. For example:

```
Execution Error - Divide by Zero on line 23 of the program.
```

Describe the changes in the project compiler

a) scanner
b) parser
c) AST data structure
d) code generation
e) run time environment
f) machine interpreter

that would be necessary to implement runtime line numbers.

5. [15 marks] A programming language designer has decided to allow arrays with dynamically (i.e. runtime) determined size to occur as fields within record structures. The designer has imposed the following restriction on the use of this language feature:

> Record fields for which the size of the field is determined at run time can only occur (directly or indirectly) as the *last* field(s) in a record structure.

Example

```
    type bigRec :
        record
              int X, Y, Z ;
              string(24) S, T ;
              array 0 .. M of float V ;
              array 1 .. N of string(P) U ;
        end record
```

Where M, N and P are run time expressions.

a) Describe (for the general case) how you would allocate storage for records containing dynamically sized fields.

b) Describe (for the general case) the code you would need to generate to support allocation of and access to dynamically sized records fields.

c Discuss the implications of the programming language designers restriction on the implementation of this language feature.

6. [15 marks] There are several possible choices for implementing variable length characters strings (perhaps with a declared maximum length):

null terminated A string is a sequence of characters accessed by a pointer to the first character. The end of a string is marked by a character containing a special value, usually zero.

explicit length A string is represented as an initial field (byte of halfword) containing the current length of the string, followed by storage for the string value. Strings are accessed via a pointer to this initial field. In some implementations a second field at the start of the string gives the declared maximum length of the string.

Discuss the advantages *and* disadvantages of each choice with regard to the ease of implementation of typical string operations like assignment, concatenation, substring, string search, parameter passing. Assume that your choice is not constrained by the programming language or by requirements to interface with other languages.

7. [20 marks] An expression E is *very busy* if regardless of which path we take through the control flow graph of a program, the value of E will be used before it is killed. Example, $A+3$ is *very busy* in:

```
    . . .
    if( expr ) {
        V = A + 3 ;
        R = K + 3 ;
    }else{
        Z = A + 3 ;
        K = 5 ;
        L = K + 3 ;
    }
```

The data flow equations for computing very busy expressions are

$$in[B] = used[B] \cup (out[B] - killed[B])$$

$$out[B] = \bigcap_{S \,\varepsilon\, successors[B]} in[S]$$

a) Is *very busy expressions* a forward-flow or a backward-flow problem?

b) Define $in[B]$, $out[B]$, $used[B]$ and $killed[B]$ for computing very busy expressions.

c) Divide the program fragment shown below into basic blocks and compute $in[B]$ and $out[B]$ for each basic block.

```
        . . .
        X = 5 ;
        Y = 10 ;
        if( E1 )
            if( E2 )
                A = X * Y ;
            else {
                B = 3 ;
                V = X * Y ;
                X = 1 ;
            }
        else {
           Y = 2 ;
           A = X * Y ;
        }
```