University of Toronto
Faculty of Arts and Science
April/May Examinations 2003
CSC488H1S / CSC2107H1S
Duration – 2 hours (120 minutes)

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.

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 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. [15 marks] Show the branching code that would be generated for the Boolean expression in the while statement on lines 20 and 21 below. You can invent any branch or compare instructions you need as long as you describe them fully

```
10    char ch  ;
      . . .
20    while(  'a' <= ch && ch <= 'z'  ||  'A' <= ch && ch <= 'Z'
21           ||  '0' <= ch && ch <= '9'  || ch == '_' ) {
         ...
30    }
```

2. [15 marks] Convert the grammar below into an LL(1) grammar.
Show the director sets for your revised grammar.

```
1    S  →  S  r  Y
2       →  r  X
3       →  X
4    X  →  X  s  Y
5       →  Y  u  Y
6       →  Y
7    Y  →  u  Y
8       →  t
```

3. [15 marks] Describe the optimizations that a good optimizing compiler might perform on the fragments of C code listed below.

```
10      int R[ 100 ] , I, RM , J ;
        . . .
35      int lz( int X[] , int N ){
36          if( N < 0 )
37             return  -1 ;
38          else
39             if( X[N] == 0 )
40                return N ;
41             else
42                return lz( X , N - 1 );
43      }
        . . .
70      RM = lz( R , 99 ) ;
71      for( I = 99  ; RM >= 0 ; I = RM  ) {
72         printf("%d\n", RM );
73         for( J = RM+1 ; J <= 99 ; J++ )
74              R[ J - 1 ] = R[ J ] * R[ J ] ;
75         RM = lz( R , RM - 1 ) ;
76      }
77      I = 99 ;
        . . .
```

4.[15 marks] Use the simple storage mapping Algorithm 1 described in the lecture notes to show how the C structure given below might be laid out in memory.

```
 1        struct  {
 2             char  C ;
 3             struct {
 4                 float  X ;
 5                 double Y ;
 6             } Z ;
 7             int  I, J ;
 8             struct {
 9                 char   D ;
10                 short  E ;
11                 double F ;
12            } G ;
13            int K ;
14        } A ;
```

You may assume the bit sizes and alignments for the basic data types that are given in the table below.

Type	Size	Align	Type	Size	Align	Type	Size	Align
char	8	8	int	32	32	short	16	16
float	32	32	double	64	64			

5. [15 marks]A programming language with a rich set of character string manipulating primitives has the following characteristics:

- String variables must be declared. The declaration for a string variable specifies a maximum length for the variable. The actual length of a string variable ranges from zero (null string) up to the maximum declared length.
- The maximum length may be a compile time constant or a run time expression. In the case of a run time expression the maximum allowed length of the string is determined when the declaration is encountered during program execution.
- The string processing functions provided in the language include the usual set of operations on strings: length, substring, concatenation, indexing, etc. You can use the C string functions as a model. Strings can be passed as parameters to routines and returned by functions.

a) Describe a complete strategy for implementing storage for character string variables in this language. Discuss how strings with a maximum length determined at run time should be handled.

b) One unique feature of this language is that it allows the substring operation to be used on the *left* side of an assignment statement. For example:

substr(S , E1 , E2) = ”Hello World” ;

Assigns the string constant ”Hello World” to the substring of the string variable S starting at the character position given by the value of the expression E1 , and continuing for the number of characters determined by the value of the expression E2.

For the general case, discuss the implementation issues raised by allowing substring to be used on the left side of an assignment statement. What run time checks would be necessary to ensure the correct execution of this construct?

6. [15 marks] The CSC488S resident computer architect is about to redesign the pseudo machine processor used in the course project. What four machine instructions would you like to see added to the machine to make code generation easier?

7. [15 marks]Some programming languages (e.g. Modula-3) allow default values to be provided for routine parameters when the routine is declared. For example:

procedureP(A : **float** = 3.14 , B : **int** = 23 , invert : **boolean** = **false** ,
msg : **string**(*) = ”Hello World”) ;
. . .
end P ;

A routine call in such a language would have some notation (e.g. consecutive commas) to indicate that there is no actual parameter in a given call corresponding to some formal parameter. For example, the call:

P(2.78 , , , ”Goodbye Cruel World”);

would cause P to be called with default values for the parameters B and invert .
Describe the programming language implementation issues raised by this parameter mechanism. Describe the semantic analysis and code generation mechanisms that would be required to support this construct.

8. [15 marks] The C programming language allows the size of fields in a structure to be specified with *bit resolution*. The general form of a structure field definition is:

unsigned *name* : *bits*

Where *name* is the name of the field, and *bits* is a constant giving the size of the field *in bits*. The C standard allows the compiler to impose an implementation defined maximum on the size of a field (usually word size). The implementation can layout fields so that they don’t cross storage unit boundaries by adding unnamed fill in the structure. The implementation also decides what *storage unit* it uses in laying out structures. Most implementations use a memory word, but halfword or even byte is allowed by the standard. The special value zero for the *bits* specifier forces the next field in the structure to start on a storage unit boundary. The *address of* operator (&) cannot be applied to bit sized fields. Example:

```
unsigned int  X ;
struct {
       unsigned int A : 7 ;
       unsigned int B : 13 ;
       unsigned int C : 12 ;
} Y ;
```

a) Describe the impact that the presence of bit fields will have on the compiler processing of structures.

b) Assume the compiler is using 32-bit words as the storage unit for structures. Describe the kind of code that a compiler would have to emit to support the assignment statement:

```
X = Y.B ;
```

based on the structure declared above.

c) Using the same assumptions as the previous part, Describe the code that a compiler would have to emit to support the assignment statement:

```
Y.B = X ;
```