University of Toronto
Faculty of Arts and Science
April/May Examinations 2004
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 .
State clearly any assumptions that you have to make to answer a question.

1. [15 marks] The grammar below describes a simplified version of declarations in C.

1	declaration	→	typeSpecifier declarators ';'
2	declarators	→	declareAndInit
3		→	declarators ',' declareAndInit
4	typeSpecifier	→	'int"
5		→	'long'
6		→	'long' 'int'
7	declareAndInit	→	declarator optInitializer
8	declarator	→	IDENTIFIER
9		→	declarator '(' ')'
10		→	declarator '[' ']'
11		→	declarator '[' EXPRESSION ']'
12	optInitializer	→	initializer
13		→	λ
14	initializer	→	'=' EXPRESSION
15		→	'=' '{' initializerList '}'
16	initializerList	→	EXPRESSION
17		→	initializerList ',' EXPRESSION
18		→	'{' initializerList '}'

In this grammar, terminal symbols are enclosed in single quote marks. IDENTIFIER and EXPRESSION are also terminal symbols in this grammar. λ is the empty string.

Convert this grammar to an LL(1) grammar.

2. [20 marks] Describe the semantic analysis checks that a competent compiler would make on the following fragment of Java.

```
 1    public class OrderedList {

 5    private boolean empty ;
 6    private OrderedList  left , right ;
 7    private Comparable  val  ;

80    public void addElement( Comparable EL ) throws nullPointerException,
81          DuplicateException , ClassCastException
82    {
83        if( val == null ) throw new NullPointerException( "addElement" ) ;
84        if( empty ) {
85            left = new OrderedList( ) ;
86            right = new OrderedList( ) ;
87            val = EL ;
88            empty = false ;
89            return ;
90        }
91        int n = EL.compareTo( val ) ;
92        if( n == 0 ) throw new DuplicateException( "addElement" ) ;
93        if( n < 0  )
94             left.addElement( val ) ;
95        else
96             right.addElement( val ) ;
97    }

400 }
```

3. [15 marks] In the Postscript programming language, arrays may be created dynamically using the [ ] operators. Arrays in Postscript are *heterogeneous*. This means that the individual elements stored in an array can each be of a different type (e.g. an integer, a real, a boolean, a string, a name (reference to a variable), the body of a function, a dictionary (a table like structure), or even another array.
Examples:

```
[  3, 3.14 , true  ]
% 3 elements:  integer, real, boolean

[ /ABC , 2.78, 23, {DUP ADD POP} , [ 3 , 4, 5 ] ]
% 5 elements: variable, real, integer, function , array
```

Describe a set of run time mechanisms and data structures that would be sufficient to support this form of dynamic, heterogeneous array. You should assume that you also need to support strict run time subscript range checking for indexing into arrays. Checking on the appropriate *use* of an array element is done by the Postscript interpreter and is not part of this question.

4.[15 marks] A proposal has been made to extend a Turing/Pascal like programming language to allow arrays and strings with run time determined sizes *in* record type declarations. For example:

```
1       type flexRecord
2           record
3               int A ;
4               real B [ E1 ] ;
5               boolean C ;
6               string D ( E2 ) ;
7               int E [ 12 ] ;
8           end ;
```

Where *E1* and *E2* are expressions that are evaluated at runtime. The language designers have asked for your opinion as an ace complier writer.

a) What implementation difficulties do you see in this proposal?

b) There are two alternatives for when the expressions in a type declaration like this get evaluated:

b.1 Once when the type declaration is encountered during execution

b.2 Each time that the type is used to create a variable.

Which alternative would be more difficult to implement? Why?

c) What language restrictions would you suggest to make this construct easier to implement?

5. [15 marks] A lexical analyzer for Fortran has to deal with several kinds of tokens:

identifiers	a non-empty sequence of letters and digits starting with a letter
integer constants	$\mathcal{D}$
real constants	$\mathcal{D}$ **.**$\mathcal{D}$ $\mathcal{D}$**.** $\mathcal{D}$**.**$\mathcal{D}$
	$\mathcal{D}$ E$\mathcal{D}$ $\mathcal{D}$**.**E$\mathcal{D}$ $\mathcal{D}$**.**$\mathcal{D}$ E$\mathcal{D}$ **.**$\mathcal{D}$ E$\mathcal{D}$
boolean constants	**.**FALSE**.** **.**TRUE**.**
boolean operators	**.**AND**.** **.**OR**.** **.**NOT**.**
comparison operators	**.**LT**.** **.**LE**.** **.**EQ**.** **.**NE**.** **.**GE**.** **.**GT**.**

Where $\mathcal{D}$ stands for a non-empty sequence of decimal digits and E is used to mark the exponent part of real numbers (e.g. `1.0E10`). Integer and real constants may be used in the same expression with automatic conversion as required. Fortran is a *blank insensitive* language, except inside strings, the presence or absence of blanks *anywhere* doesn't change the meaning of the program.

Discuss the *problems* that this *combination of lexical tokens* will cause in the design of a lexical analyzer for Fortran. You **do not** have to give the design of a lexical analyzer.

6. [15 marks] Describe how the C struct and union declarations shown below would be laid out in memory using the structure mapping algorithm described in lecture..

```
1  typedef  union{                | 10  struct Mstruct  {
2             long A  ;           | 11        short  N ;
3             float B1, B2 ;      | 12        long   P ;
4             char C[ 17  ]  ;    | 13        /* U from typedef on left */
5             struct {            | 14        U      Q[ 2 ]  ;
6                 short   D1 ;    | 15        int R[ 3 ]  ;
7                 double  D2 ;    | 16        double S ;
8             } D ;               | 17      }  M ;
9          }  U ;                 | 18
```

You should assume the sizes and alignment constraints:

type	size	align	type	size	align	type	size	align
char	8	8	short	16	16	int	32	32
long	64	32	float	32	32	double	64	64

7. [10 marks] Describe the symbol and type table entries that a typical compiler might make for the data structures in the previous question

8. [15 marks] Describe the optimizations that a good optimizing compiler would perform on the fragment of C code shown below. You may assume that int and floats take 4 bytes of storage. You can show only the final results of the optimization if you make it clear what optimizations have been performed.

```
10       float A[ 100 ] , C[ 200 ] ;
11       double B[ 100 ][ 200 ]  ;
12       int   I , J , K ;
13       /* Assume that 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 ] ;
```