University of Toronto

Faculty of Arts and Science

April/May Examinations 1999

CSC488S CSC2107S

Duration - 2 hours - 120 minutes

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

Part I - 3 Questions 60 marks, Part II - 6 Questions 60 marks. 120 marks total.
If more than 3/6 questions are answered, only the first 3/6 will be marked.
You must receive a mark of 35% or greater on this exam to pass the course.

WRITE LEGIBLY. Unreadable answers are not answers.
State clearly any assumptions that you have to make to answer a question.

Conventions for all questions:

In grammars, uppercase letters are nonterminal symbols and lower case letters are terminal symbols. λ is the empty string.
Line/rule reference numbers on the left side of programs and grammars are provided for ease of reference only and are not part of the program or grammar.

Part I - Answer ANY 3 Questions in Part I

1. [20 marks] Show a translation for the program below into the machine language for the machine that was used in the course project.

```
 1    func ack ( M : int , N : int ) : int
 2        {   var  outVal : int
 3            if  M = 0 then
 4                outVal = N + 1
 5            else
 6                outVal =  {   var inVal : int
 7                              if N = 0 then
 8                                  inVal = ack( M - 1 , 1 )
 9                              else
10                                  inVal = ack( M - 1 , ack( M , N - 1))
11                              result  inVal
12                          }
13            result  outVal
14        }
 ...  ...
25    put  ack( 3, 5 )
```

2. [20 marks] The C programming language has a mechanism called *setjump/longjmp* that is intended to provide a way to do deep error returns from nested function calls. The relevant C definitions are:

```
#include <setjmp.h>
int setjmp( jmp_buf env );
void longjmp( jmp_buf env , int val ) ;
```

Where *jmp_buf* is defined in `setjmp.h`. The setjmp function stores the current executing enviroment in its argument *env*. In this mode setjmp returns zero. The longjmp function causes control to return to the environment specified by its argument *env*. In this case setjump appears to return the non-zero value *val* passed back by longjmp. An example of typical usage:

```
jmp_buf outofhere ;
...
if( ( retVal = setjmp( outofhere )) != 0 ) {
        /* longjmp has returned here */
        if( retVal == 95 || retVal == 98 )
                printf("Mr. Gates I presume?\n")
        ...
}
...
void probeOS( ... ) {
        ...
        if( BorgSighted )
                longjmp( outofhere , 98 ) ;
        ...
}
```

a) [10 marks] Describe the run-time management issues related to providing this type of mechanism in a programming language. What kind of information will need to be stored in the jmp_buf? Justify each piece of information that you include.

b) [10 marks] Describe a run time implementation of long jumps.
What should be done when setjmp is called?
What should be done when longjmp is called?

3. [20 marks] Consider the C program fragment shown below.

```
double A[20,18], B[360] ;
int J, K ;
. . .   /* Assume A is initialized here */
for( J = 0 ; J < 20 ; J++ )
        for( K = 0 ; K < 18 ; K++ ) {
                B[ 18*J + K ] = 3.0 * A[ J ][ K ] * A[ J ][ K ]
                        - 12.0 * A[ J ][ K ] + 23.0 ;
        }
```

Assume that an optimizing compiler unrolls the innermost loop (i.e. the loop on K) 4 times. Describe the transformations that an optimizing compiler would perform on this program fragment. Assume the type double takes 8 bytes of storage. You can show only the final optimized version of the fragment **if** you make it **very clear** what optimizations have been performed.

4. [20 marks] Non-procedural languages like APL, Snobol and Icon allow *variables* to be created dynamically. For example in Icon, a character string value can be created (e.g by concatenation, by substring selection or by input from a file) and then the *value* of the string can be used as the *name* of a variable. If the variable already exists then the name is the name of the existing variable otherwise a new variable is created and initialized with a null value. Dynamic variables have global scope.

Design a runtime mechanism to implement this kind dynamic creation a variables. Discuss at least each of the issues listed below.

a) how is the association between variable names and variable values maintained?

b) how should storage management for dynamically created variables be done? Can storage for a variable ever be freed?

c) What kind of dynamic semantic analysis will be required?

Part II - Answer ANY 6 Questions in Part II

IF MORE THAN 6 QUESTIONS ARE ANSWERED, ONLY THE FIRST 6 WILL BE COUNTED.

5. [10 marks] Consider the LL(1) grammar

1	A	$\rightarrow$	B	C	c	
2		$\rightarrow$	e	D	B	
3	B	$\rightarrow$	λ			
4		$\rightarrow$	b	C	D	E
5	C	$\rightarrow$	D	a	B	
6		$\rightarrow$	c	a		
7	D	$\rightarrow$	λ			
8		$\rightarrow$	d	D		
9	E	$\rightarrow$	e	A	f	
10		$\rightarrow$	c			

a) [5 marks] Give the first and follow sets for each non-terminal symbol.
b) [5 marks] Give the director sets for the grammar

6. [10 marks] For the declaration given below

```
union bigU {
    unsigned char uchar ;
    struct {
        int ordinal  ;
        double dNumb ;
    } stra ;
    struct {
        unsigned char cursor ;
        int xCoord, yCoord ;
        char * sptr ;
        double value ;
    } strb  ;
    short clink ;
} ;
```

Show how this structure would be laid out in memory using the space conserving *Algorithm 2* that was discussed in lecture. Assume the size and alignment constraints in the table below.

type	size	align	type	size	align
char *	32	32	unsigned char	8	8
double	64	64	short	16	16
int	32	32			

7. [10 marks] Assume that W, X, Y, Z are float variables and P, Q, R are int variables used as boolean variables. Show how the Boolean expression below would be translated into branching code.

```
(( W - X ) < 1E-5 ) && ! ( P || ( Q && !R)) || ( Y > Z && R )
```

8. [10 marks] Find all the nullable non-terminal symbols in the following grammar. Explain your answer.

1	S	→	a	B	D	
2		→	A	B		
3		→	D	A	C	
4		→	b			
5	A	→	S	C	B	
6		→	S	A	B	C
7		→	C	b	D	
8		→	c			
9		→	λ			
10	B	→	c	D		
11		→	d			
12		→	λ			
13	C	→	A	D	C	
14		→	c			
15	D	→	S	a	C	
16		→	S	C		
17		→	f	g		

9. [10 marks] Each of the programming language restrictions listed below arises because of some implementation issue. For each restriction, explain the underlying implementation issue.
   a) character strings may not be longer than 255 characters.
   b) record variables may not be compared using the relational operators < , <=, =, !=, > , >= .
   c) Function declarations may not be nested, i.e. the declaration of a function can not occur inside the declaration of a function.
   d) The target label of a goto statment must be in the some function or procedure as the goto statement itself.

10. [10 marks] The Turing language definition imposes very strict error checking requirements on pointer operations.
   1) An error message must be given if an dereference of a pointer results in a dereference of the null pointer.
   2) An error message must be given if any dereference of a pointer uses a pointer that is *dangling* (i.e. pointing to a block of storage that has been freed).

   Describe a set of mechanisms that will implement this error checking.

11. [10 marks] In the C preprocessor, the directives

```
#include  <filename.h>
#include  "filename.h"
```

can be used to incorporate a secondary file into the main file being compiled. The use of #include may be nested to an arbitrary depth.
Describe an implementation of #include that could be used in a C preprocessor, i.e. what should the preprocessor do to process a #include directive.

12. [10 marks] Describe the static semantic checks that a competent C compiler would perform on the statements shown below

```
...         ...
10          if( ch = getchar() != EOF )
11                  fprintf( stderr, "next char is %c", ch ).
12          else
13                  return ;
14          buffer[ bp++ ] = ch ;
15          for( K = 0 ; K < bp - 1 ; K++ )
16                  if( ch == buffer[ K ] ){
17                          bp -- ;
18                          break ;
19                  } ;
```