University of Toronto

Faculty of Arts and Science

April/May Examinations 2001

CSC488H1S CSC2107H1S

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 - 4 Questions 40 marks, Part II - 4 Questions 80 marks. 120 marks total.
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.
The notation ... stands for correct code that has been omitted for clarity.

Part I - Answer ANY 4 Questions in Part I

IF MORE THAN 4 QUESTIONS ARE ANSWERED, ONLY THE FIRST 4 WILL BE COUNTED.

1. [10 marks] Describe the optimizations that a good optimizing compiler would perform on the C function shown below. You may show only the final result of the optimization **if** you **very clearly** describe all of the optimizations that have been performed. Assume byte addressing and that int and float variables occupy 4 bytes of storage.

```
 1    void shellsort ( float A[], int L , int R ) {
 2        int I , H ;
 3        for ( H = 1 ; H <= ( R -1 )/9 ; H = 3*H + 1 ) ;
 4        for ( ; H > 0 ; H /= 3 )
 5            for ( I = L + H ; I <= R ; I++ ) {
 6                int J = I ;
 7                while( J >= L + H && A[ I ] < A[ J - H ] ){
 8                    A[ J ] = A[ J - H ] ;
 9                    J -= H ;
10                }
11                A[ J ] = V ;
12            }
13    }
```

2. [10 marks] Construct the director sets for the LL(1) grammar described below

1	S	=	’a’	S	,
2			’b’	S	’c’ ,
3			U	T	
4	T	=	’a’	S ,	
5			λ		
6	U	=	’d’	V	
7	V	=	’b’	S	’c’ ,
8			λ		

3. [10 marks] Discuss the implications that each of the following programming language features has for the design of a compiler that implements the feature.

a) variables may be declared after they are used.
b) the lower and upper bounds of arrays can be run-time expressions
c) procedure and function declarations may be nested.
d) non-local go to statements are allowed.

4. [10 marks] If you could add two new instructions (of similar complexity to the existing instructions) to the pseudo-machine used in the course project in order to make the task of compiling to the machine simpler, what would those instructions be? Justify your choice.

5. [10 marks] Corresponding elements of a *symmetric* 2-dimensional array have the same value above and below the main diagonal (.i.e. $\forall$ I $\forall$J A[I , J] = A[J , I]).

Design a space optimizing array subscripting scheme for symmetric arrays that allows only the upper half of the array to be actually stored in memory.
Show how array storage is laid out by the compiler.
Show the general case of an array subscript calculation, i.e. how is a reference to an array element A[E_1 , E_2] transformed into the memory address of the array element?

Part II - Answer ANY 4 Questions in Part II

IF MORE THAN 4 QUESTIONS ARE ANSWERED, ONLY THE FIRST 4 WILL BE COUNTED.

6. [20 marks] Assume you have the misfortune to be writing a compiler for a language that has **key words** instead of **reserved words**. This means that words like **if** , **return** , **while** that determine the structure of the program can also be used by the programmer as ordinary names for variables, constants and types. Discuss the problems that keywords cause for the compiler designer. Sketch a possible scanner/parser solution to this problem.

7. [20 marks] Several programming languages include an iteration statement of the form

```
1       for variable = expn1 , expn2 , ... , expnn do
2             . . .
3       end for
```

Where *variable* is a scalar variable and each $expn_i$ is a scalar expression that can be assigned to the variable. The example shown will loop *n* times with the variable taking on successively the values $expn_1$ up to $expn_n$. Example:

```
1       for V = 1 , N - 2 , ackermann( 3, 2 ) , V / 7 do
2             printf("V = %d\n", V );
3       end for
```

In the style of the lecture slides, design a parse-tree directed translation for this statement into the quadruples used in the lecture slides.

a) design a plausible set of AST tree nodes to represent the statement
b) design a scheme for translating this statement into quadruples
c) show the translation that is done for each tree node that you designed in a).

8. [20 marks] The programming language used in the course project has been augmented with two new statements.

return from *name*
result of *name* **is** *expression*

These statements have the same meaning as the return and result statements in the language **except** that *name* is the name of the function or procedure that is being returned from. These statements cause a return from the most recent (i.e. closest in the chain of outstanding calls) invocation of *name*.

a) discuss the implementation issues raised by these new statements.
b) what static semantic checking should be done on these statements?
c) what dynamic semantic checking should be done on these statements?
d) describe how these statements could be implemented at runtime. You don't have to show detailed instructions sequences, but you should discuss the runtime algorithms that will be used.

9. [20 marks] Several steps from the LL(1) table generation algorithm are listed below. For each step describe in words the purpose of the table entries built by the step

3c if $A \rightarrow a\ \beta$ is a production
    row *a*, col *a* $\leftarrow$ REPLACE(β) NEXT }

3d if $A \rightarrow B\ \alpha$ is a production and $B\ \alpha$ is not nullable
for each *b* in first(B α)
    row *a*, col *b* $\leftarrow$ REPLACE(B α)

3e if $A \rightarrow B\ \alpha$ is a production and $B\ \alpha$ is nullable
for each *b* in first($B\ \alpha$) $\cup$ follow(*A*)
    row *a*, col *b* $\leftarrow$ REPLACE($B α)

3f if $A \rightarrow \lambda$ is a production
for each *b* in follow(*A*)
    row *a*, col *b* $\leftarrow$ POP

10. [20 marks] Several programming languages allow *associative arrays* as a builtin data type. A typical definition might be

- the subscript of an associative array can be an arbitrary (possibly non-integer) expression, e.g. A["Hello World"] or A[3.141592653].
- the use of an array subscript expression that has never been used before for that array causes the spontaneous creation of an array element with that subscript.
- once an array element comes into existence, it remains available until the entire array is deleted (e.g. by leaving the scope in which the array was declared.
- an associative array can be used in the same ways that an ordinary array can be used.
- some form of iterator is available to enumerate all of the elements in the associative array, for example: **for** v **in** A **do**

Describe a complete implementation strategy for associative arrays. Include a description of runtime storage management and the runtime algorithms used to process a reference to an array element.