University of Toronto

CSC488S/CSC2107S Compilers & Interpreters **Winter 2003/2004**
Midterm Test [15% of final mark] **March 4, 2004**

Total marks 100 - Total time is 50 minutes. Two pages, 5 questions. Answer all questions.
Instructions: This midterm is open book, open notes. Non-programmable calculators allowed. No electronic communication devices allowed.
The line numbers in example programs and grammars are for reference only and are not a part of the programs or grammars. Ellipsis `...` indicates omitted, correct text.
If you need to make any assumptions in order to answer a question, state the assumptions clearly in your answer book.

1. [20 marks] The programming language FORTH has a simple lexical structure:

1. There must be one or more space characters or newlines between every pair of lexical tokens.
2. *Names* are any sequence for consecutive characters not containing spaces or newlines. Names may be operators or variables, the FORTH interpreter distinguishes them by table lookup.
3. *Numeric values* are names that contain only the decimal digits 0 .. 9
4. *Strings* start with the character sequence `."` and continue to the next `"`

Describe a high-level design for a lexical analyzer for the FORTH language.

2. [20 marks] Given the array declaration in C:

```
1               struct{  int index ;
2                        double  Xaxis, Yaxis ;
3                        char  key ;
4                     }  items[ 10 ] ;
```

Assume the alignment constraints and sizes char: 8, int: 32 and double: 64 .
Show how the array *items* would be laid out in memory.

3. [20 marks] In the PL/I programming language, the naming of procedure formal parameters in the procedure header is **decoupled** from the declaration of the parameters in the procedure body. For example (using C syntax):

```
 1          procedure P( A , B , C ) {
              ...
17            float C ;
              ...
23            int A[] ;
              ...
37            struct { int X ; double Y } B ;
              ...
45        }  /* end P */
```

Describe the symbol table and semantic analysis issues that result from this decoupling.

4. [20 marks] One of the four grammars shown below is LL(1), one is LALR(1) but not LL(1) and two are neither. Identify each grammar.
Justify your answers. Answers without justification will get partial credit.
λ is the empty string. Lower case letters are terminal symbols.

1

S	$\rightarrow$	A B c
S	$\rightarrow$	A B D
A	$\rightarrow$	a
B	$\rightarrow$	b B
B	$\rightarrow$	λ
D	$\rightarrow$	c D
D	$\rightarrow$	λ

2

S	$\rightarrow$	A B c
S	$\rightarrow$	A b d
A	$\rightarrow$	A a
A	$\rightarrow$	a
B	$\rightarrow$	b B
B	$\rightarrow$	λ

3

S	$\rightarrow$	A b
S	$\rightarrow$	B a
A	$\rightarrow$	a A
A	$\rightarrow$	λ
B	$\rightarrow$	C b
B	$\rightarrow$	b

4

S	$\rightarrow$	a S e
S	$\rightarrow$	B C
B	$\rightarrow$	b B e
B	$\rightarrow$	b B e d
B	$\rightarrow$	C
C	$\rightarrow$	c C e
C	$\rightarrow$	d

5.[20 marks] Several modern programming languages (e.g. C++, Java, Ada) allow class method overloading. The usual approach is to allow the definition of several different functions/procedures with the same name (e.g. class Constructors) as long as the functions can be distinguished because they have distinct parameter lists.

a) describe in detail the symbol table mechanisms that would be required to support this form of overloading.

b) describe the semantic analysis checks that need to be applied to a function/procedure declaration.

c) describe an algorithm (using the symbol table mechanisms from part a) that could be applied to a function/procedure call that would resolve use of an overloaded function/procedure to the correct instance.

Example:

```
100         void P( int A )  {    /* 1st declaration of P */
            ...
120         }
121         void P( char A , float X ) {    /* 2nd declaration of P */
            ...
180         }
            ...
200         P( 100 ) ;            /* call on 1st P */
201         P( 'W', 23.0  ) ;     /* call on 2nd P */
202         P(  100 , 23.0 ) ;    /* ERROR no matching P */
```