University of Toronto

CSC488S/CSC2107S Compilers and Interpreters **Winter 2008/2009**

Mid Term Test (15% of course mark) **March 5, 2009.**

5 questions on **2** pages. **100 marks total. 50 minutes total**
Open Book and Notes, Non-programmable calculators allowed, NO other electronic aids allowed
Answer all questions. WRITE LEGIBLY!
If you need to make any additional assumptions to answer a question, be sure to state those assumptions in your test booklet.
The line numbers on the left side of programs are for reference only and not part of the program.

1. [20 marks] A proposal has been made to extend the CSC488S project language with *scope expressions* using the syntax:

scopeExpression: '**begin**' declarations statements '**return**' expression '**end**' **,**
'**begin**' statements '**return**' expression '**end**'

and adding

expression: scopeExpression **,**

A scope expression behaves like an implicit parameterless function. It can be used anywhere that an expression can be used. Example:

Z : = **begin var** Y : **int** Y = 0 **while**(X > 0) **do begin** X := X / 2 Y := Y + 1 **end return** Y **end**

a) Describe the semantic analysis issues raised by this proposal, i.e what checking is required.
b) Describe the symbol table issues raised by this proposal. How would they be resolved?
c) Describe any other issues raised by this proposal.

2. [20 marks] Show how storage would be mapped for the array of structures (A) declared below.
Show the subscript calculation for A[23]. Show your work.

```
1   typedef struct {
2           char b ;
3           struct {
4               int c ;
5               double d ;
6           } e ;
7           short f;
8   } structType ;

9   structType A[ 97 ] ;
```

Sizes and Alignments

Type	Size	Alignment
char	8	8
short	16	16
int	32	32
double	64	64

3. [10 marks] The Eiffel programming language is **case insensitive** for identifiers and reserved words. e.g.
test Test tEST TeSt TEST are all the same identifier
else ELSE Else elsE eLsE are all the same reserved word
Describe the lexical and syntactic processing necessary to implement this feature.

4. [25 marks] The grammar below describes a simplified version of declarations in C.

```
1    declaration:          decl_specifiers  init_declarators  ';'
2    decl_specifiers:      type_specifier  decl_specifier
3    decl_specifier:       type_specifier  decl_specifier  ,
4                          /* empty */
5    type_specifier:       'int'  ,
6                          'long'
7    init_declarators:     init_declarator  init_declarators  ,
8                          /* empty */
9    init_declarator:      declarator  initializer
10   declarator:           identifier  ,
11                         declarator  '('  ')'  ,
12                         declarator  '['  ']'
13   initializer:          '='  expression  ,
14                         '='  '{'  initializer_list  '}'  ,
15                         /* empty */
16   initializer_list:     expression  ,
17                         initializer_list  ','  initializer_list  ,
18                         '{'  initializer_list  '}'
```

You may assume that *identifier* , *expression* and all of the symbols enclosed in single quote marks are terminal symbols. `/* empty */` is a comment marking rules with empty right hand sides.

Convert this grammar to an LL(1) grammar. Show the director sets to prove the LL(1) property. You may assume any follow sets you need for other parts of C as long as you specify a sufficient condition (e.g. the follow set cannot contain X) for the grammar to be LL(1).

5. [25 marks] The function *mul* shown below is from class *Poly* that implements manipulation of polynomials in one variable.
List the type and usage checks that a Java compiler would perform on this function.

```
public class Poly {
    private int [ ] trms ;
    private int deg ;
    . . .
1     public Poly mul (Poly q) throws NullPointerException {
2         // returns the Poly this * q
3         if (( q.deg == 0 ) && q.trms[ 0 ] == 0 ) | |
4            ( deg == 0 ) && trms[ 0 ] == 0 ) return new Poly ;
5         Poly r = new Poly( deg + q.deg ) ;
6         r.trms[ deg+q.deg ] = 0 ;        // prepare to compute coeffs
7         for ( int i = 0 ; i <= deg ; i ++ )
8             for ( int j = 0 ; j <= q.deg ; j ++ )
9                 r.trms[ i + j ] = r.trms[ i + j ] + trms[ i ] * q.trms[ j ] ;
10        return r ;
11    }
```