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

```plaintext
scopeExpression: 'begin' declarations statements 'return' expression 'end',
'begin' statements 'return' expression 'end'
```

and adding

```plaintext
expression: scopeExpression ,
```

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

```plaintext
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.

```plaintext
typedef struct {
    char b ;
    struct {
        int c ;
        double d ;
    } e ;
    short f ;
} structType ;
structType A[ 97 ] ;
```

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.

```plaintext
1 declaration: decl specifiers init declarators ;
2 decl specifiers: type specifier decl specifiers
3 decl specifier: type specifier decl specifiers ,
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.

```java
public class Poly {
    private int [] trms ;
    private int deg ;
    ...

    public Poly mul (Poly q) throws NullPointerException {
        // returns the Poly this * q
        if ((( q.deg == 0 ) && q.trms[ 0 ] == 0 ) ||
             ( deg == 0 ) && trms[ 0 ] == 0 ) return new Poly ;
        Poly r = new Poly( deg + q.deg ) ;
        r.trms[ deg+q.deg ] = 0 ; // prepare to compute coeffs
        for ( int i = 0 ; i <= deg ; i ++ )
            for ( int j = 0 ; j <= q.deg ; j ++ )
                r.trms[ i + j ] = r.trms[ i + j ] + trms[ i ] * q.trms[ j ] ;
        return r ;
    }
}
```

Page 2 of 2 pages.