Mid Term Test Solution

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.

   a) Need to capture type of block from type of final expression for type checking.
   b) Need to be able to create/hide symbol table entries for the block scope in the middle of expression processing.
   c) This feature does not introduce any syntax problems. An interesting question is: what is the meaning of a `return` statement if it occurs in `statements` rather than at the end? Is it another way to exit a scope expression or is it a return from an enclosing procedure/function?

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.

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

   Applying the structure mapping algorithm to structType results in this layout

<table>
<thead>
<tr>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
</tbody>
</table>

   The total size of structType is 208 bits (26 bytes).
   However to make every element of the array A align properly, a fill of 48 bits (6 bytes) is required between elements of A. The size of an element of A is therefore 256 bits (32 bytes).
   Address of A[23]: `@ A[23] = Abase + (23 - 0) * 32 = Abase + 736`
3. [10 marks] The Eiffel programming language is case insensitive for identifiers and reserved words. e.g.

test Test TestT Test TEST are all the same identifier
distinguish DISTINGUISH DISTINGUISH else ELSE Else ELSE eLsE are all the same reserved word

Describe the lexical and syntactic processing necessary to implement this feature.

During lexical analysis map all identifiers and reserved words to lower (or upper) case. A good compiler would record the original case in a bitmap (1 bit per character) for all identifiers (at least) so error messages could print identifiers as the programmer wrote them. Syntax analysis would only need to deal with the lower case set of reserved words.

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

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 \texttt{mul} shown below is from class \texttt{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 ;
    }
}
```

**Lines**

1. Verify that \texttt{nullPointerException} is defined and is of a throwable exception type.
2. Check that \texttt{mul} with this signature has not been previously defined.
3. For each reference to \texttt{Poly} Verify that \texttt{Poly} is defined and is a class.
4. For each reference to \texttt{q.deg} Verify that \texttt{q} is defined and has accessible field \texttt{deg}
5. For each reference to \texttt{q.trms} Verify that \texttt{q} is defined and has accessible field \texttt{trms} that is a one dimensional array
6. For each reference to \texttt{r.trms} Verify that \texttt{r} is defined and has accessible field \texttt{trms} that is a one dimensional array
7. For each reference to \texttt{deg} Verify that \texttt{deg} is of an integer type
8. For each reference to \texttt{trms} Verify that \texttt{trms} is a field of this
9. For each reference to \texttt{this} Verify that \texttt{this} is a field of this
10. For each reference of the \texttt{==} operator Verify that its operands can be compared
11. For each reference of the \texttt{&&} operator Verify that both operands are of a Boolean type
12. For each instance of a \texttt{return} statement Verify that the expression being returned is valid for the \texttt{mul} function
13. For each instance of the \texttt{+} operator Verify that both operands are of an acceptable numeric type
14. For each instance of the \texttt{++} operator Verify that both operands are of an acceptable numeric type
15. For each instance of the \texttt{new} operator Verify that its operand is a class and that a suitable constructor exists.
16. For each instance of the \texttt{=} operator (assignment) Verify that the assignment is valid
17. For each instance of the \texttt{<=} operator verify that its operands can be compared.
18. Verify that the operands of the \texttt{||} operator are of Boolean type
19. For each instance of the \texttt{[]} operator, verify that the subscript is valid.
20. For each reference to \texttt{i} Verify that \texttt{i} is defined and accessible
21. For each reference to \texttt{j} Verify that \texttt{j} is defined and accessible
22. Check that \texttt{r} has not been previously defined.