1. [10 marks] Assume you would like to make the project language a little more programmer friendly and allow all double character terminal symbols to have white space between them. For example, allow := and : = , allow <> and < >. Describe how you would implement this change in the compiler488 scanner and parser.

2. [10 marks] During the LL(1) table construction algorithm, how is a failure in the LL(1) property detected, i.e. what condition indicates that the grammar is not LL(1)?

3. [15 marks] In some programming languages comments start with the characters (* and end with the characters *) . Comments may be nested. For example:

```
(* Outer comment which contains
  (* this inner comment
    which ends here =>*)
  and the outer comment ends on the next line
*)
```

Describe the lexical analysis algorithms necessary to process nested comments.
4. [25 marks] The figure below illustrates three different ways of representing fixed length strings (i.e. the maximum length of the string is a constant known to the compiler).
   a) C-style null terminated strings.
   b) String with an explicit length field at the front.
   c) String with an explicit length field at the end.

   Where the length field contains the current length of the string, 0 stands for a byte containing zero. The arrow indicates the memory address used to access the string. The shaded part of each string represents characters that are unused in the current value of the string (i.e. the current value is shorter than maximum value)

   a) 
   ![Diagram of C-style null terminated strings]

   b) 
   ![Diagram of string with explicit length field at the front]

   c) 
   ![Diagram of string with explicit length field at the end]

   Evaluate and compare each of these representations in terms of ease of implementation of common string operations:

   a) string assignment

   b) obtaining the length of a string

   c) string concatenation, you can assume that tail concatenation (S := S + moreS) is the dominant form of concatenation.

   d) obtaining a substring from a larger string

   e) passing strings as arguments to functions/procedures that will accept any length string as an argument.
5. [10 marks] Describe the optimizations that a good optimizing compiler would make on the fragment of C code on lines 20 .. 25.

```c
double cubeIt( double X ) {
    return X * X * X
}

double squareIt( double X ) {
    return X * X
}

double A[ 50 ][ 50 ], B[ 50 ][ 50 ], X[ 50 ];
int J, K;
/* Assume A and B are given values here */
...
for( J = 0 ; J < 50 ; J ++ )
    X[ J ] = 0.0;
for( J = 1 ; J < 50 ; J ++ )
    for( K = 0 ; K < 50 ; K ++ )
        X[ J ] = X[ J ] + 7.0 * cubeIt( 2.0 * A[ J ][ K ] )
            + 5.0 * squareIt( B[ J ][ K ] * X[ J - 1 ] ) + 3.0;
```

6. [10 marks] The following fragment of legal C code is known as Duffs device. It is a copy loop that has been unrolled eight times that also efficiently deals with copy lengths that are not multiples of eight.

```c
void duff( char *to, char *from, int count ) {
    /* copy count characters */
    int n = ( count + 7 ) / 8 ;
    switch( count % 8 ) {
        case 0: do { *to++ = *from++ ;
        case 7: *to++ = *from++ ;
        case 6: *to++ = *from++ ;
        case 5: *to++ = *from++ ;
        case 4: *to++ = *from++ ;
        case 3: *to++ = *from++ ;
        case 2: *to++ = *from++ ;
        case 1: *to++ = *from++ ;
            } while( --n > 0 ) ;
    }
}
```

Show the basic blocks for this function
Show the control flow graph for this function
7. [20 marks] The TYPECASE statement in the statically typed Modula-3 programming language has the form:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPECASE Expr OF</td>
<td>PROCEDURE ToText( r : REFANY ) : TEXT =</td>
</tr>
<tr>
<td>[ T_1 ( v_1 ) \Rightarrow S_1 ]</td>
<td>(* Assume r = NIL or r ^ is a BOOLEAN or INTEGER *)</td>
</tr>
<tr>
<td>[ T_2 ( v_2 ) \Rightarrow S_2 ]</td>
<td>BEGIN</td>
</tr>
<tr>
<td>...</td>
<td>TYPECASE r OF</td>
</tr>
<tr>
<td>[ T_n ( v_n ) \Rightarrow S_n ]</td>
<td>NULL \Rightarrow RETURN &quot;NIL&quot;</td>
</tr>
<tr>
<td>ELSE S_0</td>
<td>REF BOOLEAN (rb) \Rightarrow RETURN Fmt.Bool( rb ^ )</td>
</tr>
<tr>
<td>END</td>
<td>REF INTEGER (ri) \Rightarrow RETURN Fmt.Int( ri ^ )</td>
</tr>
<tr>
<td>END</td>
<td>END ToText</td>
</tr>
</tbody>
</table>

Where \( \text{Expr} \) is an expression whose type is a \( \text{REF ANY} \) type (e.g. like void *), the \( S_i \) are statements, the \( T_i \)s are specific reference types, and the \( v_i \) are identifiers. The \( \^ \) operator indicates pointer dereferencing, \( \text{REF type} \) is a pointer to \( \text{type} \). The ELSE part and the \( ( v_i ) \) parts are optional.

The TYPECASE statement evaluates \( \text{Expr} \). If the resulting reference value is a member of any listed type \( T_i \) then \( S_i \) is executed for the minimum such \( i \). If the value of \( \text{Expr} \) is a member of no listed type and ELSE \( S_0 \) is present then \( S_0 \) is executed otherwise a runtime error occurs. Each \( v_i \) declares a variable whose type is \( T_i \) and whose scope is \( S_i \). If \( v_i \) is present it is initialized to the value of \( \text{Expr} \) before \( S_i \) is executed.

a) Describe a set of static semantic analysis checks that should be applied to this statement.
b) Describe how you would implement this statement. You can use pseudo-code or diagrams in your answer. Include all runtime mechanisms that you require.

8. [20 marks] Design a recursive descent parser function for the TYPECASE statement in the previous question. You may assume the existence of functions that recognize statement, types and expressions. The function you are writing will be called when the statement parsing function recognizes the terminal symbol for TYPECASE as its next input token. Your function should parse one entire TYPECASE statement and return success or failure. Describe any assumptions you make in your solution.