1. [20 marks] Lexical analysis for the Java™ programming language must process Unicode escape sequences which are defined as

```
\ UnicodeMarker hexDigit hexDigit hexDigit hexDigit
```

Where UnicodeMarker is one or more instances of the letter u and hexDigit is any one of the characters 0123456789abcdefABCDEF. Because the \ character has other uses in Java™ a \ character is the start of a Unicode escape sequence if and only if it is immediately preceeded by an even (possibly zero) number of other \ characters. If it is immediately preceeded by an odd number of \ characters it is not the start of a Unicode escape sequence. There must be exactly four hexDigits in a valid Unicode escape sequence. Examples:

```
Input          Processed   Comments
\u2297          ⊗           Valid Unicode Escape for the character ⊗
\\u2297         \u2297       Odd number of preceeding \s
\\uuu005A \Z    Valid Unicode escape for the character Z
\u2260          ≠            Valid unicode escape for the character ≠
                 (there is a space after \ )
\udefg          Error, not 4 hex digits
```

Unicode escape sequences are processed before the main part of lexical analysis. Describe a lexical analysis algorithm for processing Unicode escape sequences in Java™. Describe any interactions with other parts of lexical analysis. Discuss error handling.

2. [25 marks] Show how the data structure (Z) declared below would be laid out in memory using the fill minimizing structure allocation Algorithm 2 as described in lecture. Give complete details of the layout showing the offsets of all fields.

```
1 union {
2 struct{ char A ; int B ; char C ; double D ; } X ;
3 struct{ short P ; char Q ; double R ; int S ; } Y ;
4 } Z[2] ;
```

You should assume the length and alignment factors for atomic data types shown in the table below.

<table>
<thead>
<tr>
<th>Type</th>
<th>length</th>
<th>align</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>64</td>
</tr>
</tbody>
</table>
3. [20 marks] Describe the static semantic analysis checks that a C compiler would make on the fragments of C code shown below.

```c
int I, J = 7, A[N];
...
S = malloc(sSize);
strncpy(S, T, I - J);
for(J = 0; J < sSize; J++)
    if(S[J] != A[J])
        A[J] = S[J];
printf("The answer is %d (%s)\n", I + J, S);
```

4. [15 marks] In the lectures it was recommended that the symbol table entries for minor scopes (i.e. embedded scopes delimited by { and }) should be merged with the symbol table of the closest enclosing major (i.e function or procedure) scope. Describe a complete symbol table algorithm for implementing minor scope merging. Discuss

- What happens at the start of a minor scope
- What happens at the end of a minor scope
- How the symbol table lookup algorithm is modified to deal with minor scopes.

5. [20 marks] Convert the grammar given below to an LL(1) grammar that defines the same language. \( \lambda \) is the empty string.

```plaintext
1 S = 's', 'b' D 'm' L 'e'
2 L = F L 'm' S \\
3 S \\
4 D = 'd' 'm' D \\
5 F = 'f' \\
6 lambda
```

List the LL(1) director sets for your revised grammar.