1. C++11 has lambdas, a construction with which one can write anonymous functions which depend on parameters (n in the example below) and capture non-local variables (x in the example below). In this example, a lambda is used to remove elements less than 5 from a C++ vector.

```cpp
#include <vector>
#include <iostream>
#include <algorithm>
#include <functional>

int main () {
    std::vector<int> c { 1,2,3,4,5,6,7 };  
    int x = 5;
    c.erase(std::remove_if(c.begin(), c.end(),  
                            [x](int n) { return n < x; } ), c.end());
    std::cout << "c: ";
    for (auto i: c) {
        std::cout << i << " , ";
    }
    std::cout << '\n';
}
```

What implementation challenges does this new construct pose for C++? What approach would you take in its implementation?
2. Give a regular expression for all sequences of 0’s and 1’s that

(a) contain exactly three 1’s.

(b) contain no consecutive 0s.

(c) contain an even number of 0s.
3. Give a (simple) grammar for which the following two LR(1) items appear in the same state of the LR(1) parsing automaton. Or, explain why it cannot happen. $A$, $P$, $Q$, and $R$ are non-terminals; and $x$ is a terminal.

$$A \rightarrow P \cdot QR, x \quad A \rightarrow PQ \cdot R, x$$
4. Consider the following grammar with non-terminals \{S, E, B, L\}.

\begin{align*}
1. & S \rightarrow \text{print} \ (E) ; \\
2. & S \rightarrow \text{while} \ (B) \ S \\
3. & S \rightarrow \{L\} \\
4. & E \rightarrow \text{id} \\
5. & E \rightarrow \text{num} \\
6. & B \rightarrow E > E \\
7. & L \rightarrow S \ L \\
8. & L \rightarrow \epsilon \\
\end{align*}

(a) Compute nullable, FIRST, and FOLLOW for all nonterminals of the grammar.

<table>
<thead>
<tr>
<th>nullable</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Fill-in the table below. Fill-in the FIRST of the right-hand side, or the FOLLOW of the left-hand side, as appropriate for computing the LL(1) parsing table.

<table>
<thead>
<tr>
<th>N \rightarrow \alpha</th>
<th>null(N)?</th>
<th>FIRST(\alpha)</th>
<th>FOLLOW(N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \ S \rightarrow \text{print} \ (E) ;</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 \ S \rightarrow \text{while} \ (B) \ S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 \ S \rightarrow {L}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 \ E \rightarrow \text{id}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 \ E \rightarrow \text{num}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 \ B \rightarrow E &gt; E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 \ L \rightarrow S \ L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 \ L \rightarrow \epsilon</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(c) Fill-in the partial LL(1) parse table below for the indicated terminals.

<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th>num</th>
<th>while</th>
<th>print</th>
<th>{   }</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. For the following augmented grammar:

\[
\begin{align*}
0 & \quad S' \rightarrow S \$ \\
1 & \quad S \rightarrow a A \\
2 & \quad S \rightarrow b B \\
3 & \quad A \rightarrow C a \\
4 & \quad A \rightarrow D b \\
5 & \quad B \rightarrow C b \\
6 & \quad B \rightarrow D a \\
7 & \quad C \rightarrow E \\
8 & \quad D \rightarrow E \\
9 & \quad E \rightarrow
\end{align*}
\]

(a) Give a diagram of the LR(1) states and transitions.
(b) Give the LR(1) parsing tables.
(c) Is the grammar LR(1)? Explain.
(d) Is the grammar LALR(1)? Explain.