Q1. (Give answer in short)

A. Define annotated parse tree. Give example.
   A parse tree decorated with attributes, is called an annotated parse tree

B. What do you mean by CFG?

In formal language theory, a context-free grammar (CFG) is a formal grammar in which every production rule is of the form

\[ V \rightarrow w \]

where \( V \) is a single nonterminal symbol, and \( w \) is a string of terminals and/or nonterminals (\( w \) can be empty). A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal. No matter which symbols surround it, the single nonterminal on the left hand side can always be replaced by the right hand side.

C. What do you mean by panic mode error recovery?

A simple panic-mode error handling system requires that we return to a high-level parsing function when a parsing or lexical error is detected. The high-level function re-synchronizes the input stream by skipping tokens until a suitable spot to resume parsing is found. For a grammar that ends statements with semicolons, the semicolon becomes the synchronizing token.

We add error-handling code to all the parsing functions so that when they detect parsing errors, instead of exiting, they return FALSE. We check the return codes of all the parsing functions, and return FALSE when any of them return FALSE. The high-level parsing function detects the FALSE return and does the appropriate re-synchronization of the input stream by skipping tokens.

D. Mention the conflicts that occur in shift reduce parser?

Ans: LR(1) parsers are table-driven, shift-reduce parsers that use a limited right context (1 token) for handle recognition

- LR(1) parsers recognize languages that have an LR(1) grammar

Informal definition:
A grammar is LR(1) if, given a rightmost derivation
\[ S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow \text{sentence} \]

We can
1. isolate the handle of each right-sentential form \( \gamma_i \)
2. determine the production by which to reduce,
   by scanning \( \gamma_i \) from left-to-right, going at most 1 symbol beyond the right end of the handle of \( \gamma_i \)
E. Write any four types of errors generated in compilation process.

Ans: Common C++ compilation errors

- Undeclared identifier, e.g.:
  doy.cpp: In function `int main()': doy.cpp:25: `DayOfYear' undeclared (first use this function)[1]
  This means that the variable "DayOfYear" is trying to be used before being declared.

- Common function undeclared, e.g.:
  xyz.cpp: In function `int main()': xyz.cpp:6: `cout' undeclared (first use this function)[1]
  This means that the programmer most likely forgot to include `<iostream`.

- Parse error, e.g.:
  somefile.cpp:24: parse error before `something'[2]
  This could mean that a semi-colon is missing at the end of the previous statement.

F. Name the techniques used in loop optimization?

Loop optimization can be viewed as the application of a sequence of specific loop transformations (listed below or in [3]) to the source code or intermediate representation, with each transformation having an associated test for legality. A transformation (or sequence of transformations) generally must preserve the temporal sequence of all dependencies if it is to preserve the result of the program (i.e., be a legal transformation). Evaluating the benefit of a transformation or sequence of transformations can be quite difficult within this approach, as the application of one beneficial transformation may require the prior use of one or more other transformations that, by themselves, would result in reduced performance.

G. What are the functions performed by analysis phase?

Ans: A lexer forms the first phase of a compiler frontend in modern processing,[a] and is generally done in a single pass.

Lexers and parsers are most often used for compilers, but can be used for other computer language tools, such as prettyprinters or linters. A lexer itself can be divided into two stages: the scanner, which segments the input sequence into groups and categorizes these into token classes; and the evaluator, which converts the raw input characters into a processed value.

Lexers are generally quite simple, with most of the complexity deferred to the parser or semantic analysis phases, and can often be generated by a lexer generator, notably lex or derivatives. However, lexers can sometimes include some complexity, such as phrase structure processing to make input easier and simplify the parser, and may be written partially or completely by hand, either to support additional features or for performance.

H. What is the difference between assembler and compiler?
Ans: A Compiler and Interpreter both carry out the same purpose – convert a high level language (like C, Java) instructions into the binary form which is understandable by computer hardware. They are the software used to execute the high level programs and codes to perform various tasks. Specific compilers/interpreters are designed for different high level languages. However both compiler and interpreter have the same objective but they differ in the way they accomplish their task i.e. convert high level language into machine language.

SECTION-B

Q2. Construct LALR(1) parsing table for the following grammar

\[ S \rightarrow Aa \mid a \ A \ c \mid Bc \mid b \ Ba \]

\[ A \rightarrow d \]

\[ B \rightarrow d \]

Ans: Solved example in class

Q3. Generate three address code for the following

a) \(\text{while}(A<C \text{ and } B>D) \text{ do} \)
   \(\text{If } A=1 \text{ then and } B>D) \text{ do} \)
   \(\text{Else} \)
   \(\text{While } A<=D \text{ do} \)
   \(A=A+3\)

b. \(\text{Begin} \)
   \(\text{add}=0;\)
   \(I=1;\)
   \(j=1;\)
   \(\text{do} \)
   \(\text{begin} \)
   \(\text{add} = \text{add} + a[i][j] \ast b[j][i];\)
   \(i=i+1;\)
   \(j=j+1;\)
   \(\text{end} \)
   \(\text{while } i<=20 \text{ and } j<=20;\)
   \(\text{end} \)

Where \(a\) and \(b\) are arrays of size 20X20, and there are four bytes per word.

Q4. Explain synthesized and inherited attribute with examples. Write syntax directed translation scheme for the following grammar by using postfix notation, syntax tree and three address code.

\[ E \rightarrow E+T \]
\[ E \rightarrow T \]
\[ T \rightarrow T*F \]
\[ T \rightarrow F \]
\[ F \rightarrow id \]
Ans: An attribute grammar is a formal way to define attributes for the productions of a formal grammar, associating these attributes to values. The evaluation occurs in the nodes of the abstract syntax tree, when the language is processed by some parser or compiler.

The attributes are divided into two groups: synthesized attributes and inherited attributes. The synthesized attributes are the result of the attribute evaluation rules, and may also use the values of the inherited attributes. The inherited attributes are passed down from parent nodes.

In some approaches, synthesized attributes are used to pass semantic information up the parse tree, while inherited attributes help pass semantic information down and across it. For instance, when constructing a language translation tool, such as a compiler, it may be used to assign semantic values to syntax constructions. Also, it is possible to validate semantic checks associated with a grammar, representing the rules of a language not explicitly imparted by the syntax definition.

Attribute grammars can also be used to translate the syntax tree directly into code for some specific machine, or into some intermediate language.

| Expr1 → Expr2 + Term [ Expr1.value = Expr2.value + Term.value ] |
| Expr → Term [ Expr.value = Term.value ] |
| Term1 → Term2 * Factor [ Term1.value = Term2.value * Factor.value ] |
| Term → Factor [ Term.value = Factor.value ] |
| Factor → "(" Expr ")" [ Factor.value = Expr.value ] |
| Factor → integer [ Factor.value = strToInt(integer.str) ] |

Q5. Construct the finite automaton equivalent to the regular expression

\[(0+1)^*(00+11)(0+1)^*\]

Ans: Solved example in class

Q6. Generate language for the following grammar

a) \[ S \rightarrow S1 | S2 \]
\[ S1 \rightarrow 0 S1 1 | \text{null} \]
\[ S2 \rightarrow 1 s2 0 | \text{null} \]

Ans: \((01|10)^n\) where \(n\geq0\)

b) \[ S \rightarrow aAd \]
\[ S \rightarrow bAc \]
\[ A \rightarrow bc \]

Ans: \{abcd, bbcc\}

Q7. Draw leftmost derivation, Right most derivation and parse tree for each for the following grammar

a) \[ S \rightarrow aA | bA \]
\[ A \rightarrow aS | b AA | a \]
\[ B \rightarrow bS | a BB | b \]

For string “aabbabbbba”

Ans: Solved example in class

b) \[ S \rightarrow aASa | a \]
\[ A \rightarrow SBa | SS | ba \]

For string “ aabbaa”
Ans: Given string is not acceptable.

Q8. a) Mention any five identities of regular expression and prove with examples.

Ans:
1) $R^*R^* = R^*$
2) $(R^*)^* = R^*$
3) $RR^* = R^*R$
4) $(PQ)^*P = P(QP)^*$
5) $(a+b)^* = (a^*b^*)^* = (a+b)^* = a^*(ba^*)^*$
6) $\emptyset + R = R$
7) $R\epsilon = R$
8) $\emptyset^* = \epsilon$

b) Write a short note on Token Patterns and Lexemes.

Ans: Token: Token is a sequence of characters that can be treated as a single logical entity. Typical tokens are,

1) Identifiers 2) keywords 3) operators 4) special symbols 5) constants

Pattern: A set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token.

Lexeme: A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Example:

<table>
<thead>
<tr>
<th>Description of token</th>
<th>Token</th>
<th>lexeme</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>const</td>
<td>const</td>
<td>const</td>
</tr>
<tr>
<td>if</td>
<td>if</td>
<td>if</td>
<td></td>
</tr>
<tr>
<td>relation</td>
<td>&lt;,&lt;=,=,&lt;&gt;,&gt;=,&gt;</td>
<td>&lt; or &lt;= or = or &lt; or &gt; or &gt;= or letter followed by letters &amp; digit</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>pi</td>
<td>any numeric constant</td>
<td></td>
</tr>
<tr>
<td>nun</td>
<td>3.14</td>
<td>any character b/w “and “except&quot;</td>
<td></td>
</tr>
<tr>
<td>literal</td>
<td>&quot;core&quot;</td>
<td>pattern</td>
<td></td>
</tr>
</tbody>
</table>