SRM VALLIAMMAI ENGINEERING COLLEGE
SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

QUESTION BANK

VII SEMESTER
CS8602 – COMPILER DESIGN
Regulation – 2017
Academic Year 2019 – 20 EVEN

Prepared by
Mrs. K. Devi, Assistant Professor/CSE
Mrs. S. Shanthi, Assistant Professor/CSE
UNIT I - INTRODUCTION TO COMPILERS

PART-A (2 - MARKS)

<table>
<thead>
<tr>
<th>Q. No</th>
<th>QUESTIONS</th>
<th>Competence</th>
<th>BT Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define tokens, patterns and lexemes.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>2</td>
<td>Classify approach would you use to recover the errors in lexical analysis phase.</td>
<td>Apply</td>
<td>BTL-3</td>
</tr>
<tr>
<td>3</td>
<td>Apply the regular expression for identifier and white space.</td>
<td>Apply</td>
<td>BTL-3</td>
</tr>
<tr>
<td>4</td>
<td>Point out why is buffering used in lexical analysis? What are the commonly used buffering methods?</td>
<td>Analyze</td>
<td>BTL-4</td>
</tr>
<tr>
<td>5</td>
<td>Define transition diagram for an identifier.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>6</td>
<td>Compare syntax tree and parse tree.</td>
<td>Analyze</td>
<td>BTL-4</td>
</tr>
<tr>
<td>7</td>
<td>Summarize the issues in a lexical analyzer.</td>
<td>Evaluate</td>
<td>BTL-5</td>
</tr>
<tr>
<td>8</td>
<td>Define buffer pair.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>9</td>
<td>Differentiate the features of DFA and NFA.</td>
<td>Understand</td>
<td>BTL-2</td>
</tr>
<tr>
<td>10</td>
<td>State the interactions between the lexical analyzer and the parser.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>11</td>
<td>Explain parse tree and construct a parse tree for –(id + id)</td>
<td>Evaluate</td>
<td>BTL-5</td>
</tr>
<tr>
<td>12</td>
<td>Describe the operations on languages.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>13</td>
<td>List out the phases of a compiler.</td>
<td>Remember</td>
<td>BTL-1</td>
</tr>
<tr>
<td>14</td>
<td>Generalizes the advantage of having sentinels at the end of each buffer halves in buffer pairs.</td>
<td>Create</td>
<td>BTL-6</td>
</tr>
<tr>
<td>15</td>
<td>Classify the four software tools that generate parser.</td>
<td>Analyze</td>
<td>BTL-4</td>
</tr>
<tr>
<td>16</td>
<td>Discuss Regular expression and the Algebraic properties of Regular Expression.</td>
<td>Understand</td>
<td>BTL-2</td>
</tr>
<tr>
<td>17</td>
<td>Formulate the regular expressions are used though the lexical constructs of any programming language can be described using context free grammar.</td>
<td>Create</td>
<td>BTL-6</td>
</tr>
<tr>
<td>18</td>
<td>Apply a grammar for branching statements.</td>
<td>Apply</td>
<td>BTL-3</td>
</tr>
<tr>
<td>19</td>
<td>Express the main idea of NFA? And discuss with examples (a/b)*</td>
<td>Understand</td>
<td>BTL-2</td>
</tr>
<tr>
<td>20</td>
<td>Define lex. Discuss the components of a lex.</td>
<td>Understand</td>
<td>BTL-2</td>
</tr>
<tr>
<td>PART-B (13- MARKS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>1. <strong>Describe</strong> the various phases of compiler with suitable example (13)</td>
<td>Remember</td>
<td>BTL1</td>
<td></td>
</tr>
<tr>
<td>2. (i) Give the structure of compiler (4)</td>
<td>Analyze</td>
<td>BTL4</td>
<td></td>
</tr>
<tr>
<td>(ii) <strong>Analyze</strong> structure of compiler with an assignment statement (9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. (i) <strong>Discuss</strong> in detail about the role of Lexical analyzer with the possible error recovery actions. (7)</td>
<td>Understand</td>
<td>BTL2</td>
<td></td>
</tr>
<tr>
<td>(ii) Describe in detail about issues in lexical analysis. (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. (i) <strong>Describe</strong> the Input buffering techniques in detail. (7)</td>
<td>Remember</td>
<td>BTL1</td>
<td></td>
</tr>
<tr>
<td>(ii) Discuss how a finite automaton is used to represent tokens and perform lexical analysis with examples. (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. <strong>Summarize</strong> in detail about how the tokens are specified by the compiler with suitable example. (13)</td>
<td>Understand</td>
<td>BTL2</td>
<td></td>
</tr>
<tr>
<td>7. (i) <strong>Solve</strong> the given regular expression (a/b)* abb (a/b)* into NFA using Thompson construction. (7)</td>
<td>Apply</td>
<td>BTL3</td>
<td></td>
</tr>
<tr>
<td>(ii) Compare NFA and DFA. (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. <strong>Create</strong> DFA the following NFA. M={q0,q1}, {0,1}, δ, q0, {q1} Where δ(q0,0)={q0,q1} δ(q0,1)={q1} δ(q1,0)=ϕ δ(q1,1)={q0,q1} (13)</td>
<td>Create</td>
<td>BTL6</td>
<td></td>
</tr>
<tr>
<td>9. (i) <strong>Show</strong> how the DFA is directly converted from an augmented regular expression ( ( ε / a ) b * ) * . (7)</td>
<td>Apply</td>
<td>BTL3</td>
<td></td>
</tr>
<tr>
<td>(ii) Draw NFA for the regular expression ab*/ab (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. (i) <strong>Define</strong> the language accepted by FA. Convert the following NFA into DFA. (7)</td>
<td>Remember</td>
<td>BTL1</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="NFA Diagram" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii) Draw the DFA for the augmented regular expression (a</td>
<td>b)*# directly using syntax tree. (6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. <strong>Define</strong> Lex and Lex specifications. How lexical analyzer is constructed using lex? Give an example. (13)</td>
<td>Remember</td>
<td>BTL1</td>
<td></td>
</tr>
<tr>
<td>12. (i) <strong>Explain</strong> an algorithm for Lex that recognizes the tokens. (7)</td>
<td>Evaluate</td>
<td>BTL5</td>
<td></td>
</tr>
<tr>
<td>(ii) Describe in detail the tool for generating lexical analyzer. (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. (i) <strong>Analyze</strong> the algorithm for minimizing the number of states of a DFA (7)</td>
<td>Analyze</td>
<td>BTL4</td>
<td></td>
</tr>
</tbody>
</table>
14. **Show** the minimized DFA for the regular expression: 
\((0 + 1)^* (0 + 1) \, 1 \, 0.\)  
\((a|b)^*a(a|b)(a|b)(a|b).\)

### PART-C (15- MARK )

1. (i) **Create** languages denoted by the following regular expressions
   a) \((a|b)^*a(a|b)(a|b)\)
   b) \(a^*ba^*ba^*\)
   c) \(!!(a|b)\,(a|b)(aa|bb)\,(ab|ba)(aa|bb)\,\)*

(ii) Write regular definitions for the following languages:
   a) All strings of lowercase letters that contain the five vowels in order.
   b) All strings of lowercase letters in which the letters are in ascending lexicographic order.
   c) Comments, consisting of a string surrounded by / and /, without an intervening */ unless it is inside double-quotes "")

2. **Find** transition diagrams for the following regular expression and regular definition.
   - \(a(a|b)^*a\)
   - \(((e|a)b^*)^*\)
   - All strings of digits with at most one repeated digit.
   - All strings of a's and b's that do not contain the substring abb.
   - All strings of a's and b's that do not contain the subsequence abb.

3. (i) **Prove** that the following two regular expressions are equivalent by showing that the minimum state DFA's are same:
   a) \((a/b)^*\)
   b) \((a^*b^*)^*\)

(ii) Minimize DFA using Thompson Construction
   \((a/b)^*a\,b\,b\,(a/b)^*\)

4. **Generalize** and give an example one regular expression if we were to revise the definition of a DFA to allow zero or one transition out of each state on each input symbol. Some regular expressions would then have smaller DFA’s than they do under the standard definition of a DFA. Give and **generalize** an example of one such regular expression.
## PART-A (2 - MARKS)

<table>
<thead>
<tr>
<th>Q.No</th>
<th>QUESTIONS</th>
<th>BT Level</th>
<th>Competence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Write the rule to eliminate left recursion in a grammar. Prepare and Eliminate the left recursion for the grammar. S → Aa</td>
<td>Create</td>
<td>BTL6</td>
</tr>
<tr>
<td></td>
<td>A → Ac</td>
<td>Sd</td>
<td>ε.</td>
</tr>
<tr>
<td>2.</td>
<td>Define handle pruning.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>3.</td>
<td>Solve FIRST and FOLLOW by use the LL(1) grammar.</td>
<td>Apply</td>
<td>BTL3</td>
</tr>
<tr>
<td>4.</td>
<td>List the concepts of Predictive parsing and shift reduce parsing.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>5.</td>
<td>Differentiate Top Down parsing and Bottom Up parsing.</td>
<td>Understand</td>
<td>BTL2</td>
</tr>
<tr>
<td>6.</td>
<td>Define Recursive Descent Parsing.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>7.</td>
<td>List out the properties of parse tree.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>8.</td>
<td>Compare and contrast top down parsing with bottom up parsing techniques.</td>
<td>Analyze</td>
<td>BTL4</td>
</tr>
<tr>
<td>9.</td>
<td>Solve the following grammar is ambiguous: S→aSbS / bSaS / €.</td>
<td>Apply</td>
<td>BTL3</td>
</tr>
<tr>
<td>10.</td>
<td>Define kernel and non-kernel items.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>11.</td>
<td>Difference between ambiguous and unambiguous grammar.</td>
<td>Analyze</td>
<td>BTL4</td>
</tr>
<tr>
<td>12.</td>
<td>Define parser. Give the advantages and disadvantages of LR parsing.</td>
<td>Evaluate</td>
<td>BTL5</td>
</tr>
<tr>
<td>13.</td>
<td>Define Phrase level error recovery.</td>
<td>Remember</td>
<td>BTL1</td>
</tr>
<tr>
<td>14.</td>
<td>Evaluate the conflicts encountered while parsing.</td>
<td>Evaluate</td>
<td>BTL5</td>
</tr>
<tr>
<td>15.</td>
<td>Analyze the categories of shift reduce parsing.</td>
<td>Analyze</td>
<td>BTL4</td>
</tr>
<tr>
<td>16.</td>
<td>How to create an input and output translator with YACC?</td>
<td>Create</td>
<td>BTL6</td>
</tr>
<tr>
<td>17.</td>
<td>Summarize the Error recovery scheme in yacc.</td>
<td>Understand</td>
<td>BTL2</td>
</tr>
<tr>
<td>18.</td>
<td>What is the main idea of Left factoring? Give an example.</td>
<td>Understand</td>
<td>BTL2</td>
</tr>
<tr>
<td>19.</td>
<td>Discuss when Dangling reference occur?</td>
<td>Understand</td>
<td>BTL2</td>
</tr>
<tr>
<td>20.</td>
<td>Examine the approach would you use in Panic mode error recovery.</td>
<td>Apply</td>
<td>BTL3</td>
</tr>
</tbody>
</table>

## PART-B (13 - MARKS)

<table>
<thead>
<tr>
<th>Q.No</th>
<th>QUESTIONS</th>
<th>BT Level</th>
<th>Competence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>(i) Explain left recursion and Left Factoring. (ii) Eliminate left recursion and left factoring for the following grammar. E → E + T</td>
<td>(7)</td>
<td>Analyze</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E - T</td>
<td>T T → a</td>
</tr>
<tr>
<td>2.</td>
<td>(i) What is an ambiguous and unambiguous grammar? Identify the following grammar is ambiguous or not.</td>
<td>(5)</td>
<td>Create</td>
</tr>
<tr>
<td></td>
<td>E→E+E</td>
<td>E*E</td>
<td>(E)-E id for the sentence id+id*id</td>
</tr>
<tr>
<td></td>
<td>(ii) Prepare the following grammar is LL(1) but not SLR(1). S→AaAb</td>
<td>(8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BbBa</td>
<td>A→€</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B→€</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>(i) Illustrate the predictive parser for the following grammar.</td>
<td>(8)</td>
<td>Apply</td>
</tr>
<tr>
<td></td>
<td>S→ (L)</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 4. | (i) **Evaluate** predictive parsing table and parse the string id+id*id. And find FIRST and FOLLOW.  
    E→E+T | T  
    T→T*F | F  
    F→(E) | id  
    (ii) Construct Stack implementation of shift reduce parsing for the grammar  
    E→E+E  
    E→E*E  
    E→(E)  
    E→id and the input string id1+id2*id3 | (7) | Evaluate | BTL5 |
| 5. | (i). **Describe** on detail about the role of parser.  
    (ii) Discuss about the context-free grammar. | (7) | Remember | BTL1 |
| 6. | (i). **What** are the different kinds of syntax error phased by a program? Explain in detail.  
    (ii). **What** are the Error recovery techniques used in Predictive parsing? Explain in detail. | (7) | Remember | BTL1 |
| 7. | **Give** the predictive parser for the following grammar.  
    S→ (L) | a  
    L→ L, S | S  
    (i). Give a rightmost derivation for (a, (a, a)) and show the handle of each right-sentential form.  
    (ii). Show the steps of a shift reduce parser. | (5) | Understand | BTL2 |
| 8. | **Analyze** the following grammar is a LR(1) grammar and construct (13) LALR parsing table.  
    S → Aa | bAc | dC | bda  
    A → d  
    Parse the input string bdc using the table generated. | Analyze | BTL4 |
| 9. | (i) **Define** YACC parser generator. List out the Error recovery actions in YACC.  
    (ii) List out different error recovery strategies. Explain them. | (8) | Remember | BTL1 |
| 10. | (i) **Show** SLR parsing table for the following grammar  
    S → Aa | bAc | Bc | bBa  
    A → d  
    B → d  
    And parse the sentence "bdc" and "dd".  
    (ii) Construct a parse tree for the input string w-cad using top down | (8) | Apply | BTL-3 |
11. (i) **Define** SLR (1) parser. Describe the Steps for the SLR parser.
(ii) **Predict** the following grammar for generate the SLR parsing table.

```
E → E + T | T
T → T * F | F
F → F* | a | b
```

12. (i) **Consider** the following grammar

```
S → AS | b
A → SA | a.
```

Construct the SLR parse table for the grammar.
(ii) **Show** the actions of the parser for the input string “abab”.

13. **Give** the LALR for the given grammar.

```
E → E + T | T , T → T * F | F , F → ( E ) / id
```

and parse the following. ( a + b ) * c

14. **Examine** the following grammar using canonical parsing table.

```
E → E + T
F → ( E )
E → T
F → id.
T → T * F
T → F
```

**PART-C (15 -MARKS)**

1. **What is an operator grammar?** **Draw** the precedence function graph for the following table.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>(</th>
<th>)</th>
<th>.</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>&lt;</td>
<td>&lt;</td>
<td>=</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. **Explain** in detail about the various types of Top –down parsing.
### PART-A (2 - MARKS)

1. List out the two rules for type checking.  
   - Remember BTL1
2. Compare synthesized attributes and inherited attributes.  
   - Analyze BTL4
3. What is Annotated parse tree?  
   - Remember BTL1
4. Define Type checker.  
   - Remember BTL1
5. What is a syntax tree? Draw the syntax tree for the assignment statement a := b * -c + b * -c  
   - Create BTL6
6. Define type systems.  
   - Remember BTL1
7. Express the rule for checking the type of a function.  
   - Understand BTL2
8. Define Syntax directed definition of a simple desk calculator.  
   - Remember BTL1
9. Summarize about the S-attributed definition?  
   - Evaluate BTL5
10. Give the difference between syntax-directed definitions and translation schemes.  
    - Understand BTL2
11. State the type expressions.  
    - Remember BTL1
12. Illustrate the methods of implementing three-address statements.  
    - Apply BTL3
    - Analyze BTL4
14. Create the target machine instructions to implement the call statement in static allocation.  
    - Create BTL6
15. Translate the conditional statement if a<b then 1 else 0 into three address code.  
    - Understand BTL2
16. Test whether the following rules are L-attribute or not? Semantic rules  
    - Evaluate BTL5
    - A.s = B.b;  
    - B.i = f(C.c,A.s)
17. What are the methods of representing a syntax tree?  
    - Understand BTL2
18. Explain the syntax directed definition for if-else statement  
    - Analyze BTL4
19. Examine the usage of syntax directed definition  
    - Apply BTL3
20. Discuss the three address code sequence for the assignment statement. d=(a-b)+(a-c)+(a-c)  
    - Understand BTL2

### PART-B (13- MARKS )

1. Discuss the following in detail about the Syntax Directed Definitions.  
   - (i)Inherited Attributes and Synthesized attributes.  
     - (7)
   - (ii) Evaluate SDD of a parse tree.  
     - (6)
   - Understand BTL2
2. Evaluate the expressions for the SDD annotated parse tree for the following expressions.  
   - (i)3 * 5 + 4n  
     - (7)  
   - Evaluate BTL5
3. Suppose that we have a production A→BCD. Each of the four non terminal A, B, C and D have two attributes: S is a synthesized attribute and i is an inherited attribute. **Analyze** For each of the sets of rules below tell whether (i) the rules are consistent with an S-attributed definition (ii) the rules are consistent with an L-attributed definition and (iii) whether the rules are consistent with any evaluation order at all?
   a) A.s = B.i + C.s
   b) A.s = B.i + C.s and D.i = A.i + B.s.

4. **Apply** the S-attributed definition and constructs syntax trees for a simple expression grammar involving only the binary operators + and -. As usual, these operators are at the same precedence level and are jointly left associative. All nonterminal have one synthesized attribute node, which represents a node of the syntax tree. Production: L→E$ E→E1+E,T,E→T, T→T1*F,T→(E),T→digit.

5. **Discuss** in detail about
   i) Dependency graph
   ii) Ordering Evaluation of Attributes.

6. **Create** variants of Syntax tree. Explain in detail about it with suitable examples.

7. (i) **Analyze** the common three address instruction forms.
   (ii) Explain the two ways of assigning labels to the following three address statements
   Do i=i+1;
   While (a[i]<v);

8. **Describe** in detail about
   i) Quadruples
   ii) Triples.

9. (i) **Describe** in detail about addressing array Elements.
   (ii) Discuss in detail about Translation of array reference.

10. **Describe** in detail about types and declaration with suitable examples.

11. **Compare** three address code for expression with the Incremental translation.

12. **Show** the intermediate code for the following code segment along with the required syntax directed translation scheme
    while ( i < 10 )
    if ( i % 2 == 0 )
    evensum = evensum + i
    else
oddsum = oddsum + i

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 13. | (i) **State** the rules for type checking with example.  
     | (ii) **Give** an algorithm for type inference and polymorphic function. | (7) Remember | BTL-1 |
| 14. | **Illustrate** an algorithm for unification with its operation. | (13) Apply | BTL-3 |

**PART-C (15-MARKS)**

1. **Create** the following unwind the arithmetic expression a+-(b+c)*
   into
   (i) Syntax tree 
   (ii) Quadruples 
   (iii) Triples 
   (iv) Indirect Triples
   (15) Create | BTL-6

2. **Explain** the steps for constructing a DAG. Construct the DAG for
   the following expression
   
   $((x+y)-(x+y)*(x-y))+(x+y)*(x-y))$
   (15) Evaluate | BTL5

3. **Generate** an intermediate code for the following code segment
   with the required syntax-directed translation scheme.
   if (a > b)
   
   x = a + b
   else
   
   x = a - b
   (15) Create | BTL-6

4. What is **Type conversion**? What are the two types of type conversion? **Evaluate** the rules for the type conversion.
   (15) Evaluate | BTL5

**UNIT IV - RUN-TIME ENVIRONMENT AND CODE GENERATION**


**PART-A (2-MARKS)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>List out</strong> limitations of the static memory allocation.</td>
</tr>
<tr>
<td>2.</td>
<td><strong>How</strong> the storage organization for the run-time memory is organized?</td>
</tr>
<tr>
<td>3.</td>
<td>What is heap allocation?</td>
</tr>
<tr>
<td>4.</td>
<td><strong>How</strong> the activation record is pushed onto the stack?</td>
</tr>
<tr>
<td>5.</td>
<td><strong>Analyze</strong> the storage allocation strategies.</td>
</tr>
<tr>
<td>6.</td>
<td><strong>State</strong> the principles for designing calling sequences.</td>
</tr>
<tr>
<td>7.</td>
<td><strong>List out</strong> the dynamic storage techniques.</td>
</tr>
<tr>
<td>8.</td>
<td><strong>Define</strong> the non-local data on stack.</td>
</tr>
<tr>
<td>9.</td>
<td><strong>Define</strong> variable data length on the stack.</td>
</tr>
<tr>
<td>10.</td>
<td><strong>Differentiate</strong> between stack and Heap allocation.</td>
</tr>
<tr>
<td>11.</td>
<td><strong>Distinguish</strong> between static and dynamic storage allocation.</td>
</tr>
<tr>
<td>12.</td>
<td><strong>Discuss</strong> the main idea of Activation tree.</td>
</tr>
<tr>
<td>13.</td>
<td><strong>Give</strong> the fields in an Activation record.</td>
</tr>
<tr>
<td>14.</td>
<td><strong>Compose</strong> space efficiency and program efficiency.</td>
</tr>
<tr>
<td>15.</td>
<td><strong>Construct</strong> typical memory hierarchy configuration of a computer.</td>
</tr>
<tr>
<td>16.</td>
<td><strong>How</strong> would you solve the issues in the design of code generators?</td>
</tr>
</tbody>
</table>
17. **Evaluate** Best-fit and Next-fit object placement.  
18. **Prepare** optimal code sequence for the given sequence  
   \[ t = a + b \]  
   \[ t = t * c \]  
   \[ t = t / d \]  
19. **Analyze** the different forms of machine instructions.  
20. **Discuss** the four principle uses of registers in code generation.  

**PART-B (13 MARKS)**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Marks</th>
<th>Method</th>
</tr>
</thead>
</table>
| 1.  | (i) **Illustrate** the storage organization memory in the perspective of compiler writer with neat diagram.  
     (ii) Compare static versus dynamic memory allocation.                      | 8     | Apply  |
| 2.  | **Explain** in detail about the various issues in code generation with examples. | 13    | Evaluate |
| 3.  | (i) **Develop** a quicksort algorithm for reads nine integers into an array a and sorts them by using the concepts of activation tree.  
     (ii) Give the structure of the action record.                               | 9     | Create |
| 4.  | **How** to design a call sequences and analyze the principles of activation records with an example. | 13    | Analyze |
| 5.  | **Discuss** in detail about the activation tree and activation record with suitable example. | 13    | Understand |
| 6.  | (i) **Analyze** the data access without nested procedure and the issues with nested procedure.  
     (ii) Give the version of quicksort in ML style using nested procedure.     | 7     | Analyze |
| 7.  | (i) **Discuss** in detail about heap manager.  
     (ii) **Describe** in detail about the memory hierarchy of a computer.       | 7     | Understand |
| 8.  | **Define fragmentation? Describe** in detail about how to reduce the fragment. | 13    | Remember |
| 9.  | **Write** short notes on the following  
     i. Best fit and next object placement.  
     ii. Managing and coalescing free space                                     | 7     | Remember |
| 10. | **Examine** the problems with manual deallocation of memory and explain how the conventional tools are used to cope with the complexity in managing memory. | 13    | Remember |
| 11. | **Explain** in detail about instruction selection and register allocation of code generation. | 13    | Analyze |
| 12. | **Illustrate** in detail about the code generation algorithm with an example. | 13    | Apply |
| 13. | **Describe** the usage of stack in the memory allocation and **discuss** in detail about stack allocation space of memory. | 13    | Understand |
| 14. | Define the heap management of memory and **describe** in detail about it.     | 13    | Remember |

**PART-C (15 MARKS)**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Marks</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Suppose the heap consists of seven chunks, starting at address 0. The sizes of the chunks, in order, are 80, 30, 60, 50, 70, 20, 40</td>
<td>15</td>
<td>Evaluate</td>
</tr>
</tbody>
</table>

**BTL**: Basic Target Level
When we place an object in a chunk, we put it at the high end if there is enough space remaining to form a smaller chunk (so that the smaller chunk can easily remain on the linked list of free space). However, we cannot tolerate chunks of fewer that 8 bytes, so if an object is almost as large as the selected chunk, we give it the entire chunk and place the object at the low end of the chunk.

If we request space for objects of the following sizes: 32, 64, 48, 16, in that order, what does the free space list look like after satisfying the requests, if the method of selecting chunks is a) First fit. b) Best fit.

2. **Compare** the stack and heap allocation memory in detail with (15) suitable examples.  
   Analyze BTL4

3. **Generate** code for the following sequence assuming that n is in a (15) memory location  
   s=0  
   i=0  
   L1 : if I > n goto L2  
   s=s+i  
   i=i+1  
   goto L1  
   L2 :  
   Create BTL-6

4. **Create** following assignment statement into three address code (15)  
   D:=(a-b)*(a-c)+(a-c)  
   Apply code generation algorithm to generate a code sequence for the three address statement. (13)  
   Create BTL-6

UNIT V- CODE OPTIMIZATION
Principal Sources of Optimization – Peep-hole optimization - DAG- Optimization of Basic Blocks  

<table>
<thead>
<tr>
<th>PART-A (2-MARKS)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>List</strong> out the examples of function preserving transformations.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>2. <strong>Illustrate</strong> the concepts of copy propagation.</td>
<td>Apply BTL3</td>
</tr>
<tr>
<td>3. <strong>State</strong> the use of machine Idioms.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>4. <strong>Show</strong> the flow graph for the quicksort algorithm</td>
<td>Apply BTL3</td>
</tr>
<tr>
<td>5. <strong>Apply</strong> the basic block concepts, how would you representing the dummy blocks with no statements indicated in global dataflow analysis?</td>
<td>Apply BTL3</td>
</tr>
<tr>
<td>6. <strong>Identify</strong> the constructs for optimization in basic block.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>7. <strong>List</strong> out the properties of optimizing compilers.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>8. <strong>Define</strong> the term data flow analysis.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>9. How is liveness of a variable calculated? <strong>Identify</strong> it.</td>
<td>Remember BTL1</td>
</tr>
<tr>
<td>10. What is DAG? <strong>Point out</strong> advantages of DAG.</td>
<td>Analyze BTL4</td>
</tr>
<tr>
<td>11. <strong>Give</strong> the uses of gen and Kill functions</td>
<td>Understand BTL2</td>
</tr>
<tr>
<td>12. <strong>Discuss</strong> the concepts of basic blocks and flow graphs.</td>
<td>Understand BTL2</td>
</tr>
<tr>
<td>13. <strong>Give</strong> the main idea of dead code elimination and constant folding.</td>
<td>Understand BTL2</td>
</tr>
<tr>
<td>14. <strong>Prepare</strong> the three address code sequence for the assignment statement.</td>
<td>Create BTL6</td>
</tr>
<tr>
<td>Question</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 15. | Construct and **explain** the DAG for the follow basic block.  
\[ d := (a - b) + (a - c) + (a - c). \]  
**DAG:**  
\[ d := b * c \]  
\[ e := a + b \]  
\[ b := b * c \]  
\[ a := e - d. \]  
**Evaluate** BTL5
| 16. | What role does the target machine play on the code generation phase of the compiler? **Analyze** it.  
**Analyze** BTL4
| 17. | Draw the DAG for the statement  
\[ a := (a * b + c) - (a * b + c) \]  
**Evaluate** BTL5
| 18. | **Develop** the code for the follow C statement assuming three registers are available.  
\[ x = a / (b + c) - d * (e + f) \]  
**Create** BTL6
| 19. | **Point out** the characteristics of peephole optimization.  
**Analyze** BTL4
| 20. | **Define** algebraic transformations. Give an example  
**Understand** BTL2

**PART-B (13 MARKS)**

<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
<th>Mark</th>
<th>Category</th>
<th>BTL</th>
</tr>
</thead>
</table>
| 1. | **Explain** briefly about the principal sources of optimization.  
(13)  
**Evaluate** BTL5
| 2. | (i) **Explain** in detail about optimization of basic blocks.  
(5)  
**Analyze** BTL4  
(ii) Construct the DAG for the following Basic block & **explain** it.  
(8)  
**Evaluate** BTL5
| 3. | **Discuss** the following in detail  
(i) Semantic preserving transformation  
(7)  
**Understand** BTL2  
(ii) Global Common subexpression  
(6)
| 4. | **Write** about the following in detail  
(i) copy propagation  
(5)  
**Remember** BTL1  
(ii) Dead code Elimination  
(5)  
**Understand** BTL2  
(iii) code motion  
(3)
| 5. | **Explain** in detail about the data-flow schemas on basic block and the transfer equations for reaching definitions with example  
(13)  
**Analyze** BTL4
| 6. | (i) Illustrate the Iterative algorithm for reaching definitions  
(7)  
**Apply** BTL3  
(ii) Discuss the live variable analysis  
(6)
| 7. | **Analyze** Peephole optimization with suitable examples.  
(13)  
**Analyze** BTL4
| 8. | **Demonstrate** optimization of Basic Blocks with an example.  
(13)  
**Apply** BTL3
| 9. | (i) **Discuss** in detail about how to find Local Common Sub expressions.  
(8)  
**Understand** BTL2  
(ii) **Discuss** in detail about the Use of Algebraic Identities.  
(5)
10. **Describe** in detail about the flow of control optimization. Identify the methods to eliminate the unreachable code, load and store data. (7) **Remember** BTL1

11. (i) Give an example to identify the dead code in the DAG. (5) **Remember** BTL1

12. **Summarize** in detail about the dataflow analysis of available expression with suitable example. (13) **Understand** BTL2

13. (i) **Formulate** steps to identify the loops in the basic block. (7) **Create** BTL6

14. **Describe** the efficient data flow algorithms in detail. (13) **Remember** BTL1

**PART-C (15 MARKS)**

1. **Create** DAG and three – address code for the following C program. (15)
   ```c
   i = 1; s = 0;
   while (i <= 10)
   {
       s = s + a[i][i];
       i = i + 1;
   }
   ```
   (15) **Create** BTL6

2. Identify the loops of the flow graph, Identify the global common sub expression for each loop, Identify Induction variables for each loop and Identify loop invariant computation for each loop from the given diagram, (15) **Create** BTL6

3. Compute the grn and Kill sets for each Block, In and Out sets for each block, Compute e_gen and e_kill from the given diagram. (15) **Evaluate** BTL5

**Diagram:**

```
ENTRY

(1) a = 1
(2) b = 2

(3) c = a+b
(4) d = c-a

(5) d = b+d

(6) d = a+b
(7) e = e+1

(8) b = a+b
(9) e = c-a

(10) a = b+d
(11) b = a-d

EXIT
```

**Legend:**

- B1
- B2
- B3
- B4
- B5
- B6
4. **Analyze** the available expressions on the following code by (15) converting into basic blocks and compute global common sub-expression elimination. (15)

```plaintext
i = 0
a:= n-3
if i < a then loop else end
label loop
b:= i -4
c:= p + b
d:= M[c]
e:=d-2
f:=i-4
g:=p+f
m[g]:=e
i:=i+1
a:=n-3
if i < a then loop else end
label end
```