Lesson Plan & Schedule (Sub.Code: CS6660 – Sub Name: COMPILER DESIGN)

<table>
<thead>
<tr>
<th>Unit</th>
<th>Task</th>
<th>No of Hours Required</th>
<th>Actual hours Taken, Reason for the deviation</th>
<th>Teaching Method</th>
<th>Book</th>
<th>Dependencies (page no.)</th>
<th>Remarks</th>
<th>No of Students Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT I (5 hours)</td>
<td>INTRODUCTION TO COMPILERS</td>
<td></td>
<td></td>
<td>BB</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Translators-Compilation and Interpretation Language processors</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Phases of Compiler</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Errors Encountered in Different Phases, Grouping of Phases</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>11</td>
<td>Assignment 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compiler Construction Tools</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming Language basics</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tutorial(Test)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Phases of Compiler
<table>
<thead>
<tr>
<th>Unit</th>
<th>Task</th>
<th>No of Hours Required</th>
<th>Actual hours Taken</th>
<th>Reason for deviation</th>
<th>Teaching Method</th>
<th>Book</th>
<th>Dependencies (page no.)</th>
<th>Remarks</th>
<th>No of Students Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT II (9 hours)</td>
<td>LEXICAL ANALYSIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Need and Role for Lexical analysis, Lexical Errors</td>
<td>1</td>
<td>BB</td>
<td>T</td>
<td>109</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Expressing Tokens by regular expression</td>
<td>1</td>
<td>BB</td>
<td>T</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Converting Regular expression to NFA</td>
<td>2</td>
<td>BB</td>
<td>T</td>
<td>147</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Converting NFA to DFA</td>
<td>2</td>
<td>BB</td>
<td>T</td>
<td>152</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimization of DFA</td>
<td>1</td>
<td>BB</td>
<td>T</td>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Language for specifying lexical analyzers, LEX, Design of lexical analyzer for a sample language</td>
<td>2</td>
<td>BB</td>
<td>T</td>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tutorial (Test)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Converting Regular expression to NFA, Converting NFA to DFA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Name of the Department: INFORMATION TECHNOLOGY  
Name of the Faculty: MRS.A.THAVA VINU  
Year /Semester: II /IV  

Lesson Plan & Schedule (Sub.Code: CS6660 – Sub Name: COMPILER DESIGN)

<table>
<thead>
<tr>
<th>Unit</th>
<th>Task</th>
<th>No of Hours Required</th>
<th>Actual hours Taken, Reason for the deviation</th>
<th>Teaching Method BB/PP/Video</th>
<th>Book</th>
<th>Dependencies (page no.)</th>
<th>Remarks</th>
<th>No of Students Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT III</td>
<td>(10 Hours) SYNTAX ANALYSIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Need and Role of the Parser</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Context free grammars</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>197</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Top down parsing</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>217</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>General strategies, Recursive decent parser, Predictive parser, LL(1) parser</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>219</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shift Reduce parser</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>236</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LR Parser, LR(0) item, Construction of SLR parsing table</td>
<td>2</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>248</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Introduction to LALR Parser</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>266</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error Handling and recovery in syntax analyzer</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>283</td>
<td>Assignment-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YACC, Design of a syntax analyzer for a sample language</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>T</td>
<td>287</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tutorial (Test)</td>
<td>1</td>
<td>Predictive parser, LL(1) parser</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lesson Plan & Schedule (Sub.Code: CS6660 – Sub Name: COMPILER DESIGN)

<table>
<thead>
<tr>
<th>Unit IV (12 Hours) SYNTAX DIRECTED TRANSLATION &amp; RUN TIME ENVIRONMENT</th>
<th>Task</th>
<th>No of Hours Required</th>
<th>Actual hours Taken, Reason for the deviation</th>
<th>Teaching Method</th>
<th>Book</th>
<th>Dependencies (page no.)</th>
<th>Remarks</th>
<th>No of Students Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit IV</td>
<td>Syntax Directed definitions</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>304</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Construction of Syntax Tree</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>318</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bottom up Evaluation of s attribute</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>310</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design of predictive translator</td>
<td>1</td>
<td></td>
<td>BB WS1</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type Systems, Specification of simple type checker</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>386</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Equivalence of Type Expressions, Type Conversions</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>371,388</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Language Issues</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>443</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Storage Organization, Storage allocation, Dynamic Storage allocation</td>
<td>1</td>
<td></td>
<td>BB T</td>
<td></td>
<td>427</td>
<td>Assignment-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parameter Passing, Symbol tables</td>
<td>1</td>
<td></td>
<td>BB R2</td>
<td></td>
<td>43,116</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Storage allocation in FORTAN</td>
<td>1</td>
<td></td>
<td>BB WS2</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tutorial (Test)</td>
<td>1</td>
<td></td>
<td>Storage Allocation strategies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Lesson Plan & Schedule (Sub.Code: CS6660 – Sub Name: COMPILER DESIGN)

<table>
<thead>
<tr>
<th>Unit</th>
<th>No of Hours Required</th>
<th>Actual hours Taken, Reason for the deviation</th>
<th>Teaching Method</th>
<th>Book Dependencies (page no.)</th>
<th>Remarks</th>
<th>No of Students Present</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit V</strong> <em>(9 Hours)</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE OPTIMIZATION AND CODE GENERATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Sources of Optimization</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>583</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop optimization</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>590</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAG</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>533</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAG representation of Basic Blocks</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>533</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimization of basic blocks</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>533</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Data flow Analysis</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>597</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficient Data Flow Algorithms</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>598</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issues in the design of code generator</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>506</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A simple code generator algorithm</td>
<td>1</td>
<td></td>
<td>BB</td>
<td>542</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tutorial (Test)</td>
<td>1</td>
<td></td>
<td>DAG, flow graphs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

St. Joseph College of Engineering
Sriperumbudur, Chennai – 600 602117
EVEN SEMESTER (2017 - 2018)

Name of the Department: INFORMATION TECHNOLOGY
Name of the Faculty: MRS.A.THAVA VINU
Year /Semester: II /IV
Name of the Department: INFORMATION TECHNOLOGY
Name of the Faculty: MRS.A.THAVA VINU
Year /Semester : II /IV

Lesson Plan & Schedule (Sub.Code: CS6660 – Sub Name: COMPILER DESIGN)

(Note: BB= Black Board, PP=Power Point Presentation, VIDEO = Video Lecture)
Text Book:

Reference Book:

Web Sources:
1: https://www.tutorialspoint.com/compiler_design/compiler_design_tutorial.pdf
2: https://www.tutorialspoint.com › Compiler Design › Compiler - Run-time Environment

Signature of the Staff
Signature of the HOD
Signature of Principal
UNIT 1
INTRODUCTION TO COMPILERS

1. Define Compiler.
Complier is a program that reads a program written in one language – the source language- and translates it into an equivalent program in another language- the target language. In this translation process, the compiler reports to its user the presence of the errors in the source program.

2. What are the main two parts of compilation? What are they performing?
The two main parts are
Analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.
Synthesis part constructs the desired target program from the intermediate representation

3. Define interpreter.
Interpreter is a language processor program that translates and executes source code directly without compiling it to machine code.

4. Define Assembler.
Assembler is a translator from human readable (ASCII text) files of machine instructions into the actual binary code (object files) of a machine.

5. Define Loader.
Loader is a program that performs the functions of loading and linkage editing. The process of loading consists of taking re-locatable machine code, altering the re-locatable address and placing the altered instruction and data in memory at the proper location.

6. Define Preprocessor. What are its functions?
A preprocessor is one, which produces input to compilers. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a distinct program called a preprocessor.
The preprocessor may also expand macros into source language statements.
The functions of a preprocessor are:
- Macro processing.
- File inclusion.
- Rational preprocessors.
- Language extensions

7. **Define Debugger.**

Debugger is a program to help you see what is going on when your program runs. It can print the values of variables, show what procedure called what procedure to get where you are, run up to a particular line, run until a particular variables gets a special value etc.

8. **What is front-end and back-end of the compiler?**

Often the phases of a compiler are collected into a front-end and back-end. Front-end consists of those phases that depend primarily on the source program and largely independent of the target machine. Back-end consists of those phases that depend on the target machine language and generally those portions do not depend on the source language, just the intermediate language. In back end we use aspects of code optimization, code generation, along with error handling and symbol table operations.

9. **What are the phases of a compiler?**

- Lexical analysis phase or scanning phase
- Syntax analysis phase
- Intermediate code generation
- Code optimization
- Code generation

10. **Name minimum 4 compiler construction tools?**

- LEX – Scanner generator
- YACC-Parser generator
- Syntax directed translation scheme.
- Automatic code generator
- Data flow engines

11. **Mention few cousins of compiler.**

- Preprocessors
- Assemblers
- Loaders
- Link editors

12. **What are the Error-recovery actions in a lexical analyzer?**

- Deleting an extraneous character
- Inserting a missing character
- Replacing an incorrect character by a correct character
- Transposing two adjacent characters

13. **Define linker.**

Linker is a program that combines (multiple) objects files to make an executable. It converts names of variables and functions to numbers (machine addresses).
Editors may operate on plain text, or they may be wired into the rest of the compiler, highlighting syntax errors as you go, or allowing you to insert or delete entire syntax constructed at a time.

15. What is the need for separating the analysis phase into lexical analysis and parsing? (Or) What are the issues of lexical analyzer?
- Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases.
- Compiler efficiency is improved.
- Compiler portability is enhanced.

16. What happens in Hierarchical analysis?
This is the phase in which characters or tokens are grouped hierarchically into nested collections with collective meaning.

17. What happens in linear analysis?
This is the phase in which the stream of characters making up the source program is read from left to right and grouped into tokens that are sequences of characters having collective meaning.

18. What is a Structure editor?
A structure editor takes as input a sequence of commands to build a source program. The structure editor not only performs the text creation and modification functions of an ordinary text editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.

PART – B

1. Describe phases of compilers with an example?

2. Explain in detail about cousins of compiler?

3. Write short notes on compiler construction tools?

4. Explain in details about Grouping of phases.

5. Describe the various phases of compiler and trace it with the program segment (position:=initial+rate*60).
LEXICAL ANALYSIS

PART A

1. What is the role of lexical analysis phase?
   Lexical analyzer reads the source program one character at a time, and grouped into a sequence of atomic units called tokens. Identifiers, keywords, constants, operators and punctuation symbols such as commas, parenthesis, are typical tokens.

2. Define lexeme?
   The character sequence forming a token is called lexeme for the token.

3. What are the two functions of parser?
   It checks the tokens appearing in its input, which is output of the lexical analyzer. It involves grouping the tokens of source program into grammatical phrases that are used by the compiler to synthesize the output. Usually grammatical phrases of the source program are represented by tree like structure called parse tree.

4. Mention the role of semantic analysis?
   Semantic analysis checks the source program for semantic errors and gathers information for the subsequent code-generation phase. It uses hierarchical structure to identify the operators and operands of expressions and statements. An important component of semantic analysis is type checking. In type checking the compiler checks that each operator has operands that are permitted by the source language specification. In such cases, certain programming language supports operand coercion or type coercion also.

5. What is a regular expression? State the rules, which define regular expression?
   Regular expression is a method to describe regular language

   Rules:
   1) $\varepsilon$ is a regular expression that denotes $\{\varepsilon\}$ that is the set containing the empty string
   2) If $a$ is a symbol in $\sum$, then $a$ is a regular expression that denotes $\{a\}$
   3) Suppose $r$ and $s$ are regular expressions denoting the languages $L(r)$ and $L(s)$ Then, a) $(r)/(s)$ is a regular expression denoting $L(r) \cup L(s)$. b) $(r)(s)$ is a regular expression denoting $L(r)L(s)$ c) $(r)^*$ is a regular expression denoting $L(r)^*$. d) $(r)$ is a regular expression denoting $L(r)$.

6. List the rules for constructing regular expressions?
   The rules are divided into two
   - Basic rules
   - Induction rules

   Basic rules:
   - $\varepsilon$ is a regular expression denoting $\{\varepsilon\}$ (i.e.) the language contains only an empty string.
   - For each $a$ in $\sum$, $a$ is a regular expression denoting $\{a\}$, the language with only one string, which consists of a single symbol $a$.

   Induction rules:
   - If $R$ and $S$ are regular expressions denoting languages $LR$ and $LS$ respectively then, ii)$(R)/(S)$ is a regular expression denoting $L(R) \cup L(S)$
   - $(R)(S)$ is a regular expression denoting $L(R) \cdot L(S)$ iv)$(R)^*$ is a regular expression denoting $L(R)^*$.  


7. **What do you mean by a syntax tree?**

Syntax tree is a variant of a parse tree in which each leaf represents an operand and each interior node represents an operator.

8. **Write the algebraic laws obeyed by the regular expression.**

For any regular expressions $R$, $S$, $T$ the following laws hold:

i. $R|S = S|R$ (commutative)

ii. $R|(S|T) = (R|S)|T$ (associative)

iii. $R.(S.T) = (R.S).T$ (associative)

iv. $R(S|T) = RS|RT$ (distributive)

9. **Define DFA.**

A DFA is an acceptor for which any state and input character has utmost one transition state that the acceptor changes to. If no transition state is specified the input string is rejected.

10. **Write a short note on LEX.**

A LEX source program is a specification of lexical analyzer consisting of set of regular expressions together with an action for each regular expression. The action is a piece of code, which is to be executed whenever a token specified by the corresponding regular expression is recognized. The output of a LEX is a lexical analyzer program constructed from the LEX source specification.

10. **What is symbol table? What are its contents?**

Symbol table is a data structure that contains all variables in the program and temporary storage and any information needed to reference or allocate storage for them. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

11. **What is the demerit in uniform structure of symbol table?**

- Length of the name should not exceed upper bound or limit of name field.
- If length of name is small, then remaining space was wasted.

12. **What are semantic errors? Give example.**

The error s detectable by the compiler is termed as semantic error. These errors are detected both at compile time and runtime. Semantic errors that can be detected at compile time errors of declaration and scope. Semantic errors that can be detected at runtime are range-checking for certain values, particularly array subscripts and case statement selectors.

13. **What are the drawbacks of using buffer pairs?**

- This buffering scheme works quite well most of the time but with it amount of lookahead is limited.
- Limited look ahead makes it impossible to recognize tokens in situations where the distance, forward pointer must travel is more than the length of buffer.

14. **Define NFA.**

A NFA is an acceptor for which any state and input character has more than one transition state that the acceptor changes to. If no transition state is specified the input string is rejected.

15. **What is meant by epsilon move?**

It is a finite automata with empty move. i.e. NFA with empty move. If we want to give transition without taking any input symbol then epsilon move is used. It is denoted by the symbol $\epsilon$.

16. **What is the need for separating the analysis phase into lexical analysis and parsing? (Or)**

What are the issues of lexical analyzer?
Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases.

- Compiler efficiency is improved.
- Compiler portability is enhanced.

17. Write a grammar for branching statements.
- \( S \rightarrow \text{if } E \text{ then } S_1 \)
- \( S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2 \)
- \( S \rightarrow \text{while } E \text{ do } S_1 \)

18. List the Operations on languages

- **Union** - \( L \cup M = \{ s \mid s \text{ is in } L \text{ or } s \text{ is in } M \} \)
- **Concatenation** - \( LM = \{ st \mid s \text{ is in } L \text{ and } t \text{ is in } M \} \)
- **Kleene Closure** - \( L^* \) (zero or more concatenations of \( L \))
- **Positive Closure** - \( L^+ \) (one or more concatenations of \( L \))

19. Differentiate Between lexeme, token and pattern.

**Tokens** - Sequence of characters that have a collective meaning. **Patterns** - There is 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 sequence of characters in the source program that is matched by the pattern for a token.

20. What is a sentinel? What is its usage?

A Sentinel is a special character that cannot be part of the source program. Normally we use ‘eof’ as the sentinel. This is used for speeding-up the lexical analyzer.

**PART-B**

1. Explain the role of the lexical analyzer?
2. Design a lexical analyzer generator?
3. Construct NFA, DFA from NFA and optimized DFA for the following
   - \( aa^*|bb^* \)
   - \((ab)*abb\)
   - \((a^*|b^*)^*\)
   - \(0(0|1)^*0\)
   - \(((ab)^*)^*\)
   - \((011)^*001\)
   - \(((\epsilon/a)b^*)^*\)
   - \(ab*/ab\)
4. Design a Lexical analyzer for a sample language.
5. Discuss about LEX tool?
7. Write notes on regular expressions
8. Construct DFA to recognize the language \((a/b)^*ab\).
9. Discuss how finite automata is used to represent tokens and perform lexical analysis with example
10. Write an algorithm for minimizing the number of states of a DFA

UNIT III
SYNTAX ANALYSIS

PART A

1. Define parser.

Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning. Also termed as Parsing.

2. What do you mean by context free grammar?
Context free grammar is a notation for specifying the syntax of a language.
A context free grammar has four components:
- A set of tokens, known as terminal symbols.
- A set of non-terminals.
- A set of productions where each production consists of a non-terminal, called the left side of the production, an arrow, and a sequence of tokens and/or non-terminals, called the right side of the production.
- A designation of one of the non-terminals as the start symbol.

3. Define terminal and non-terminal.
- Terminals are the basic symbols from which the strings are formed.
- Nonterminal are syntactic variables that denote set of strings.

4. What are the different strategies that a parser can recover from a syntactic error?
- Panic mode.
- Phrase level.
- Error productions.
- Global correction.

5. Define ambiguous grammar, and specify it demerits.
If a grammar produces more than one parse tree for the given input string then it is called ambiguous grammar. Its demerit is difficult to select or determine which parse tree is suitable for an input string.

6. When does a parser detect an error?
A parser detects an error when it has no legal move from its current configuration, which is determined by it state, its stack contents and the current input symbol. To recover from an error a parser should ideally locate the position of the error, correct the error, revise its current configuration, and resume parsing.
7. What is meant by left most derivation?
   In derivation, only the leftmost nonterminal in any sentential form is replaced at each step. Such derivations are termed leftmost derivation.

8. Define left factoring.
Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.

   It is a program based on a transition diagram attempts to match the terminal symbols against the input.

10. What you mean by nonrecursive predictive parser?
   A nonrecursive predictive parser is a program that matches the terminal symbols against the input by maintaining a stack rather than using recursive calls.

11. What is meant by shift-reduce parsing?
   Shift reduce parsing constructs a parse tree for an input string beginning at the leaves and working up towards the root i.e., “reducing” a string w to the start symbol of a grammar.

12. Define Handle.
   A handle of a string is a substring that matches the right side of a production and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.

13. What are the possible actions of a shift reduce parser? The possible actions are:
   - Shift.
   - Reduce.
   - Accept.
   - Error.

   The set of prefixes of right sentential forms that can appear on the stack of a shift reduce parser are called viable prefixes.

15. List down the conflicts during shift-reduce parsing.
   - Shift/reduce conflict.
     Parser cannot decide whether to shift or reduce.
   - Reduce/reduce conflict.
     Parser cannot decide which of the several reductions to make.

16. What is meant by an operator grammar? Given example. A grammar is operator grammar if,
   - No production rule involves “ε” on the right side.
   - No production has two adjacent nonterminal on the right side. Ex: \( E \rightarrow E+E | E-E | E*E | E/E | E \uparrow E | (E) | -E | id \)

17. What are the disadvantages of operator precedence parsing?
   i. It is hard to handle tokens like the minus sign, which has two different precedences.
   ii. Since the relationship between a grammar for the language being parsed and the operator –
precedence parser itself is tenuous, one cannot always be sure the parser accepts exactly the desired language.

iii. Only a small class of grammars can be parsed using operator precedence techniques.

19. **State error recovery in operator-Precedence Parsing.**

   There are two points in the parsing process at which an operator-precedence parser can discover the syntactic errors:
   i. If no precedence relation holds between the terminal on top of the stack and the current input.
   ii. If a handle has been found, but there is no production with this handle as a right side.

20. **Differentiate Kernel and non-Kernel items.**

   Kernel items, which include the initial item, $S' \rightarrow .S$ and all items whose dots are not at the left end. Whereas the non-kernel items have their dots at the left end.

21. **What are the components of LR parser?**

   The components of LR parser are:
   - An input.
   - An output.
   - A stack.
   - A driver program.
   - A parsing table.

22. **Define handle pruning.**

   A technique to obtain the rightmost derivation in reverse (called canonical reduction sequence) is known as handle pruning (i.e.) starting with a string of terminals $w$ to be parsed.

23. **What is meant by YACC?**

   YACC is a parser tool. It is Yet Another Compiler Compiler. It is used to generate the Parser automatically.

**PART – B**

1. Explain the role of parser?

2. Eliminate left recursion from the following grammar: $S \rightarrow Aa|b$
   $A \rightarrow Ac|Sd| \varepsilon$ (Ap)

3. Generate SLR parse table for the grammar

   $S \rightarrow Aa|bAc|Bcb|Ba$
   $A \rightarrow d$
   $B \rightarrow d$

   And parse the sentence “bdc” and “dd”

4. Draw and discuss the model of Non-Recursive predictive parsing?

5. Consider the grammar
Construct predictive parser table and validate the input string $id*id+(id*+id)$

6. Construct operator precedence functions for the operators $+,*$,$id$,$\$.$

7. i) Give an algorithm for finding the FIRST and FOLLOW positions for a given non-terminal.

ii) Consider the grammar,

$$E \rightarrow TE', E' \rightarrow + TE' | \varepsilon, T \rightarrow FT', T' \rightarrow * FT' | \varepsilon, F \rightarrow (E) | id.$$

Construct a predictive parsing table for the grammar given above. Verify whether the input string $id+id*id$ is accepted by the grammar or not.

8. Construct the predictive parser for the following grammar.

$$S \rightarrow (L)/a , L \rightarrow L,S/S$$

9. Describe the conflicts that may occur during shift reduce parsing.

10. (i) Construct Sack implementation of shift reduce parsing for the grammar

$$E \rightarrow E+E$$
$$E \rightarrow E*E$$
$$E \rightarrow (E)$$
$$E \rightarrow id$$

and the input string $id1+id2*id3$

(ii) Explain LL(1) grammar for the sentences $S \rightarrow iEt|EtSeS |a E \rightarrow b$

11. (i) Write an a algorithm for Non recursive predictive parsing.

(ii) Explain Context free grammars with examples

12. (i) Construct parse tree for the input string $w = cad$ using top down parser.

(ii) Construct parsing table for the grammar and find moves made by predictive parser on input $id * id + id$ and find FIRST and FOLLOW

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow (E) / id.$$

13. (i) Explain ambiguous grammer $G:E \rightarrow E + E | E* E | (E) | -E | ID$ for the sentence $id+id*id.$(U)
(ii) Construct SLR parsing table for the following grammer:

\[ G: \quad E \rightarrow E + T \mid TT \rightarrow T^* F \mid FF \rightarrow (E) \mid id. \]

**UNIT IV**

**SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT**

**PART A**

1. **What is mean by syntax directed definition?**
   
   It is a generalization of a CFG in which each grammar symbol has an associated set of attributes like, synthesized attribute and inherited attribute.

2. **Define three address code representations?**

   Three-address code is a sequence of statements of the form \( x = y \text{ op } z \), Where \( x, y, z \) are names, constants or compiler generated temporaries and \( \text{op} \) stands for an operator. Since a statement involves not more than three references, it is called three address statements and hence a sequence of such statement is called three address codes.

3. **List three types of representations of three address statements?**

   • Quadruples  
   • Triples  
   • Indirect triples.

4. **Define quadruple and give one example?**

   A quadruple is a data structure with 4 fields like operator, argument-1, argument-2 and result. Example: \( a = b^* - c \)

5. **Define triple and give one example?**

   A triple is a data structure with 3 fields like operator, argument-1, and argument-2.  
   Example: \( a = b^* - c \)

6. **Define back patching.**

   To generate three address codes, 2 passes are necessary. In first pass, labels are not specified. These statements are placed in a list. In second pass, these labels are properly filled, is called back patching.

7. **What is the need for type checking and type analysis?**

   The semantic analysis is done in order to obtain the precise meaning of programming construct. The need for semantic analysis is to build a symbol table that keeps track of names established in declarations. The data type of identifier is obtained and type checking of the whole expression and statements is done in order to follow the type rules of the language. It also identifies the scope of identifiers.

8. **What do you meant by type expressions?**

   The systematic way of expressing type of language construct is called type expression. It can be defined as the basic type is called type expression. Hence int, char, float, double, enum are type
expressions.

9. **What is meant by type conversion?**
   It is the process of converting one type to another. There are two types of conversion. Implicit conversion and explicit conversion. If the conversion is done automatically by the compiler then it is called implicit conversion. If the conversion is done by the programmer by writing something for converting one type to another then it is called explicit conversion.

10. **Explain with some suitable example type checker.**
    In this language construct the type of the identifier must be declared before the use of that identifier. The type checker is a translator scheme in which the type of each expression from the types of sub expressions is obtained. The type checker can decide the types for arrays, pointers, statements and functions.

11. **What are the source language issues?**
    The following are the source language issues:
    1. Recursion
    2. How the parameters are passed to the procedure.
    3. Does the procedure refer nonlocal names.

12. **What are the limitations of static allocation?**
    - The size of data objects is known at compile time.
    - It uses static allocation
    - The compiler can determine the amount of storage required by each data object.

13. **Draw the diagram of the activation record and give purpose of any two fields. (Or) Define activation tree.**
    The activation record is a block of memory used for managing information needed by a single execution of a procedure.

14. **Define actual parameters?**
    The arguments passed to a called procedure are known as actual parameters.

15. **Define activation tree?**
    An activation tree depicts the way control enters and leaves activations.

16. **Define activation record?**
    Information needed by a single execution of a procedure is managed using a contiguous block of storage called an activation record.

Various fields of activation records:

- Temporary values
- Local values
- Saved machine registers
- Control link
17. What are the different storage allocation strategies?
1) Static allocation. It lays out storage for all data objects at compile time. 
2) Stack allocation. It manages the runtime storage as a stack. 
3) Heap allocation. It allocates and deallocates storage as needed at runtime from a data area.

18. What do you mean by dangling reference?
A dangling reference occurs when there is a reference to storage that has been deallocated.

19. When does Dangling references occur?
Whenever storage is deallocated, the problem of dangling references arises.

- Occurs when there is a reference to storage that has been deallocated Logical error
- A dangling reference occurs when there is storage that has been deallocated.
- It is logical error to use dangling references, since the value of deallocated storage is undefined according to the semantics of most languages.

20. Write down syntax directed definition of a simple desk calculator.

Syntax directed definition specifies the values of attributes by associating semantic rules with the grammar productions. Syntax directed definition of simple desk calculator:

```
Production | Semantic rules
--- | ---
L --> En | L.val = E.val
E --> E1 + T | E.val = E1.val + T.val
E --> T | E.val = T.val
T --> T1 * F | T.val = T1.val * F.val
T --> F | T.val = F.val
F --> (E) | F.val = E.val
F --> digit | F.val = digit.lexval
```


- Stack: Manage activation of procedures at runtime.
- Heap: holds variables created dynamically

**PART B**

1. Write syntax directed definition for case statements?

2. Explain procedure calls with neat example?

3. Explain in detail about the storage organization.
4. What are different storage allocation strategies? Explain.

5. How names can be looked up in the symbol table? Discuss.

6. Explain about source language issues?

7. Explain in detail about parameters passing methods?

8. Explain the specification for a simple type checker.

9. Explain run time environment with suitable example.

10. Construct a syntax directed definition for constructing a syntax tree for assignment statements.

\[
S \rightarrow id := E \\
E \rightarrow E_1 + E_2 \\
E \rightarrow E_1 \times E_2 \\
E \rightarrow -E_1 \\
E \rightarrow (E_1) \\
E \rightarrow id \ (Ap)
\]

(ii) Discuss specification of a simple type checker.

11. i) Discuss different storage allocation strategies.
   ii) Illustrate type checking with necessary diagram.

12. Explain the following with respect to code generation phase.

   (i) Input to code generator
   (ii) Target program
   (iii) Memory management
   (iv) Instruction selection
   (v) Register allocation
   (vi) Evaluation order.
UNIT – V

CODE OPTIMIZATION & CODE GENERATION

PART – A

1. Define code generation?

The code generation is the final phase of the compiler. It takes an intermediate representation of the source program as the input and produces an equivalent target program as the output.

2. Define Target machine?

The target computer is byte-addressable machine with four bytes to a word and n-general purpose registers. R0, R1… … … .Rn-1. It has two address instructions of the form Op, source, destination in which Op is an op-code, and source and destination are data fields.

3. What are the rules to find “leader” in basic block?

It is the first statement in a basic block is a leader. Any statement which is the target of a conditional or unconditional goto is a leader. Any statement which immediately follows a conditional goto is a leader.

4. Define basic block?

A basic block contains sequence of consecutive statements, which may be entered only at the beginning and when it is entered it is executed in sequence without halt or possibility of branch.

5. Define flow graph?

Relationships between basic blocks are represented by a directed graph called flow graph.

6. What do you mean by DAG?

It is Directed Acyclic Graph. In this common sub expressions are eliminated. So it is a compact way of representation.

7. Specify some advantages of DAGs?

- It automatically detects common sub expression.
- We can determine which identifiers have their values used in the block.
- We can determine which statements compute values, and which could be used outside the block.
- It reconstruct a simplified list of quadruples taking advantage of common sub expressions and not performs assignments of the form a=b unless necessary.

8. What are the characteristics of peephole optimization?

- Redundant instruction elimination.
- Flow of control optimization
- Algebraic simplifications
- Use of machine idioms

9. What is the step takes place in peephole optimization?

It improves the performance of the target program by examining a short sequence of target instructions. It is called peephole. Replace this instructions by a shorter or faster sequence whenever possible, which is useful for intermediate representation.

10. State the problems in code generation?
Three main problems in code generation are,
• Deciding what machine instructions to generate.
• Deciding in what order the computations should be done and
• Deciding which registers to use.

11. Mention some of the major optimization techniques?
• Local optimization.
• Loop optimization. Data flow analysis.
• Function preserving transformations.
• Algorithm optimization.

12. What are the methods available in loop optimization?
• Code movement.
• Strength reduction.

13. What are the properties of optimizing compiler?
(i) Transformation must preserve the meaning of programs.
(ii) Transformation must, on the average, speed up the programs by a measurable amount.
(iii) A Transformation must be worth the effort.

14. Write three address code sequence for the assignment statement
\[ d := (a - b) + (a - c) + (a - c). \] (MAY/JUNE 2016)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Code generation</th>
<th>Register descriptor</th>
<th>Address descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t := a - b )</td>
<td>MOV a, R0</td>
<td>R0 contains ( t )</td>
<td>( t ) in R0</td>
</tr>
<tr>
<td>( u := a - c )</td>
<td>MOV a, R1</td>
<td>R0 contains ( t )</td>
<td>( t ) in R0</td>
</tr>
<tr>
<td>( v := u + u )</td>
<td>ADD R1, R0</td>
<td>R1 contains ( u )</td>
<td>( u ) in R1</td>
</tr>
<tr>
<td>( d := v + u )</td>
<td>ADD R1, R0</td>
<td>R0 contains ( d )</td>
<td>( d ) in R0</td>
</tr>
<tr>
<td></td>
<td>MOV R0, d</td>
<td></td>
<td>( d ) in R0 and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>memory</td>
</tr>
</tbody>
</table>

14. Identify the constructs for optimization in basic block.

There are two types of basic block optimizations.
They are:
• Structure-Preserving Transformations
• Algebraic Transformations

15. Write the characteristics of peephole optimization?
• Redundant-instruction elimination
• Flow-of-control optimizations.
• Algebraic simplifications
• Use of machine idioms
• Loop test replacement
16. What are the techniques used for loop optimization?
   i) Code motion
   ii) Induction variable elimination
   iii) Reduction in strength

17. What are the principle sources of optimization?
   The principle sources of optimization are,
   - Optimization consists of detecting patterns in the program and replacing these patterns by equivalent but more efficient constructs.
   - The richest source of optimization is the efficient utilization of the registers and instruction set of a machine.

18. What are the various types of optimization?
   The various type of optimization is,
   - Local optimization
   - Loop optimization
   - Data flow analysis
   - Global optimization

19. List the criteria for selecting a code optimization technique?
   The criteria for selecting a good code optimization technique are,
   - It should capture most of the potential improvement without an unreasonable amount of effort.
   - It should preserve the meaning of the program.
   - It should reduce the time or space taken by the object program.

20. What is data flow analysis?
   The data flow analysis is the transmission of useful relationships from all parts of the program to the places where the information can be of use.

PART – B

1. Explain the various issues involved in the design of a code generator?
2. Describe in detail about the various allocation strategies in memory management?
3. Explain in detail about register allocation and assignment?
4. Discuss briefly about DAG representation of basic blocks?
5. Construct DAG and optimal target code for the expression
   \[ X = \frac{(a+b)}{(b-c)} - (a+b) \times (b-c) + f \]
6. Explain the various transformations can be applied to basic block?
7. Explain about global flow analysis?
8. Explain Principal sources of optimization with examples.
9. Explain various issues in the design of code generator