Roll No. | | | | | | | | | | |

SHAMBHUNATH INSTITUTE OF ENGINEERING AND TECHNOLOGY

Subject Code : RCS 602 **Subject :Compiler Design**

B.Tech. SEMESTER VI

FIRST SESSIONAL EXAMINATION, EVEN SEMESTER, (2019-2020)

Branch : Computer Science & Engineering

Time –1hr 30 min **Maximum Marks – 30**

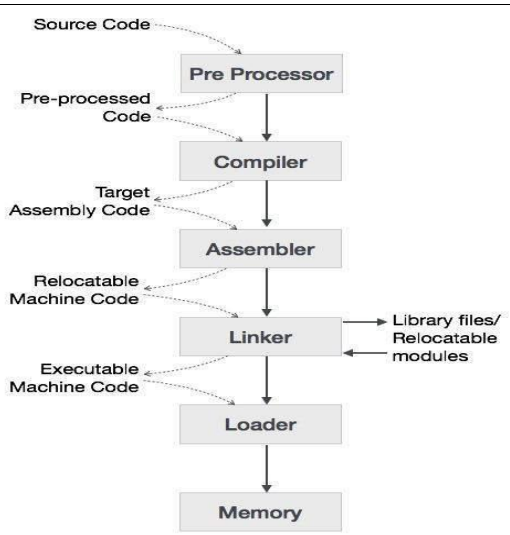
SECTION – A

1. Attempt all questions in brief. **(1*5 = 5)**

<table>
<tr><th>Q N</th><th>QUESTION</th><th>Marks</th><th>CO</th><th>BL</th></tr>
<tr><td>a.</td><td>Differentiate between compiler and interpreter.
<table>
<tr><th>BASIS FOR COMPARISON</th><th>COMPILER</th><th>INTERPRETER</th></tr>
<tr><td>Input</td><td>It takes an entire program at a time.</td><td>It takes a single line of code or instruction at a time.</td></tr>
<tr><td>Output</td><td>It generates intermediate object code.</td><td>It does not produce any intermediate object code.</td></tr>
<tr><td>Working mechanism</td><td>The compilation is done before execution.</td><td>Compilation and execution take place simultaneously.</td></tr>
<tr><td>Speed</td><td>Comparatively faster</td><td>Slower</td></tr>
<tr><td>Memory</td><td>Memory requirement is more due to the creation of object code.</td><td>It requires less memory as it does not create intermediate object code.</td></tr>
<tr><td>Errors</td><td>Display all errors after compilation, all at the same time.</td><td>Displays error of each line one by one.</td></tr>
</table></td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>b.</td><td>Discuss the advantages and disadvantages of single and multipass compiler
Ans:
A one-pass compiler is a compiler that transfers through the reference code of each compilation unit for only once. A multi-pass compiler is a type of compiler that prepares the reference code or general syntax tree of performance numerous times. A one-pass compiler is faster than multi-pass compilers.

Single-pass Compiler :
Advantage: More effective than multi-pass compilers in the compiler point of view.
Disadvantage: It compiles less efficient programs.

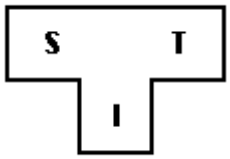
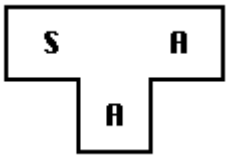
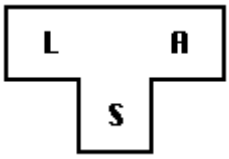
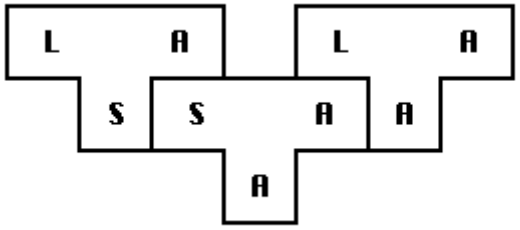
Multi-pass Compiler :
Advantages: It can be played very role useful when optimizing code.
Disadvantages: It is a very Slower process which takes a lot of time to compile the codes.</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>c.</td><td>Define language processing system.</td><td>1</td><td>1</td><td>1</td></tr>
</table>

				
d.	**Write regular expression to describe a language consist of strings made of even numbers a & b.** **Ans:** r1=((ab+ba)(aa+bb)*(ab+ba)+(aa+bb))*	**1**	**1**	**2**
e.	**What do you mean by left recursion and how it is eliminated?** **Ans:** **Left Recursion-** • A production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS. • A grammar containing a production having left recursion is called as Left Recursive Grammar. if we have the left-recursive pair of productions- **$A \rightarrow A\alpha / \beta$** Then, we can eliminate left recursion by replacing the pair of productions with- **$A \rightarrow \beta A'$** **$A' \rightarrow \alpha A' / \wedge$**	**1**	**2**	**1**

SECTION - B

2. Attempt any <u>TWO</u> of the following. (2*5 = 10)

Q N	QUESTION	Marks	CO	BL
a.	**Discuss the action taken by every phase of the compiler on the following string: x=a+b*c**	**5**	**1**	**2**
b.	**What is bootstrapping? Explain with suitable example. How bootstrapping is done on more than one machine.** **Bootstrapping** • Bootstrapping is widely used in the compilation development. • Bootstrapping is used to produce a self-hosting compiler. Self-hosting compiler is a type of compiler that can compile its own source code. • Bootstrap compiler is used to compile the compiler and then you can use this compiled compiler to compile everything else as well as future versions of itself. A compiler can be characterized by three languages: Source Language Target Language Implementation Language The T- diagram shows a compiler SCIT for Source S, Target T, implemented in I.	**5**	**2**	**2**

	 Follow some steps to produce a new language L for machine A: 1. Create a compiler SCAA for subset, S of the desired language, L using language "A" and that compiler runs on machine A.  2. Create a compiler LCSA for language L written in a subset of L.  3. Compile LCSA using the compiler SCAA to obtain LCAA. LCAA is a compiler for language L, which runs on machine A and produces code for machine A. $${}^{L}C_{S}^{A} \rightarrow {}^{S}C_{A}^{A} \rightarrow {}^{L}C_{A}^{A}$$  The process described by the T-diagrams is called bootstrapping.			
c.	**Let G be the grammar S→0B | 1A, A→0|0S|1AA, B→1|1S|0B. For the string 00110101, find (a) Left most derivation (b) rightmost (c) derivation tree.** **Ans: Derivations are not Possible**	**5**	**1**	**2**
d.	**Consider the grammar as** **S→E** **E→E+E | E*E** **E→num** **E→id** **Input to parse as id+num*id. Perform shift reduce parsing.** **Ans:**	**5**	**2**	**2**

Stack	Input	Action
$	Id_1 + num * $id_2$$	Shift
$$Id_1$	+ num * $id_2$$	Reduce by E → id
$E	+num * $id_2$$	Shift
$E +	num * $id_2$$	Shift
$E + num	* $id_2$$	Reduce by E → num
$E + E	* $id_2$$	Shift
$E + E *	$id_2$$	Shift
$E + E * id_2	$	Reduce by E → id
$E + E * E	$	Reduce by E → E
$E + E	$	Reduce by E → E
$E	$	Reduce by S → E
$S	$	Accept

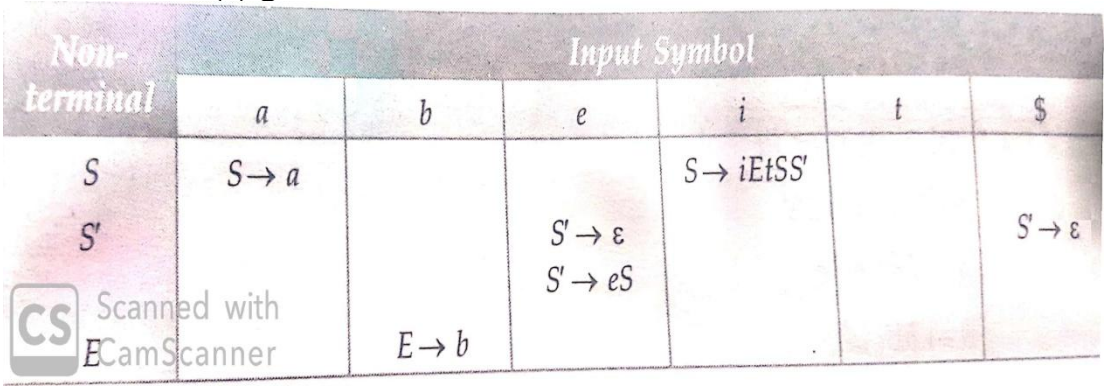
SECTION - C

3. Attempt any ONE part of the following: (1*5 = 5)

QN	QUESTION	Marks	CO	BL
a.	**Write down the regular expression for the set of all string over {a,b} such that fifth from right is a.**	5	1	3
b.	**Write and explain rules for finding FIRST and Follow sets of any given grammar.** So, following are the rules used to compute the FIRST functions : (i) If 'a' is the terminal symbol then FIRST (a) = {a} (ii) If there is a rule $A \rightarrow \varepsilon$ then FIRST (A) = {ε} (iii) For the rule $A \rightarrow A_1 A_2 \ldots A_k$ $FIRST(A) = FIRST(A_1) \cup FIRST(A_2) \cup \ldots \cup FIRST(A_k)$ The rules for computing FOLLOW function are as follows : (i) Place $ in FOLLOW(S), where S is the start symbol and $ is the input output right end marker. (ii) If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) without ε is to be placed in FOLLOW(B). (iii) If there is a production $A \rightarrow \alpha B \beta$ or $A \rightarrow \alpha B$ and FIRST(β) = {ε} then FOLLOW(B) = FOLLOW(A). That means everything in FOLLOW(A) is in FOLLOW(B).	5	2	3

4. Attempt any ONE part of the following : (1*5 = 5)

Q N	QUESTION	Marks	CO	BL

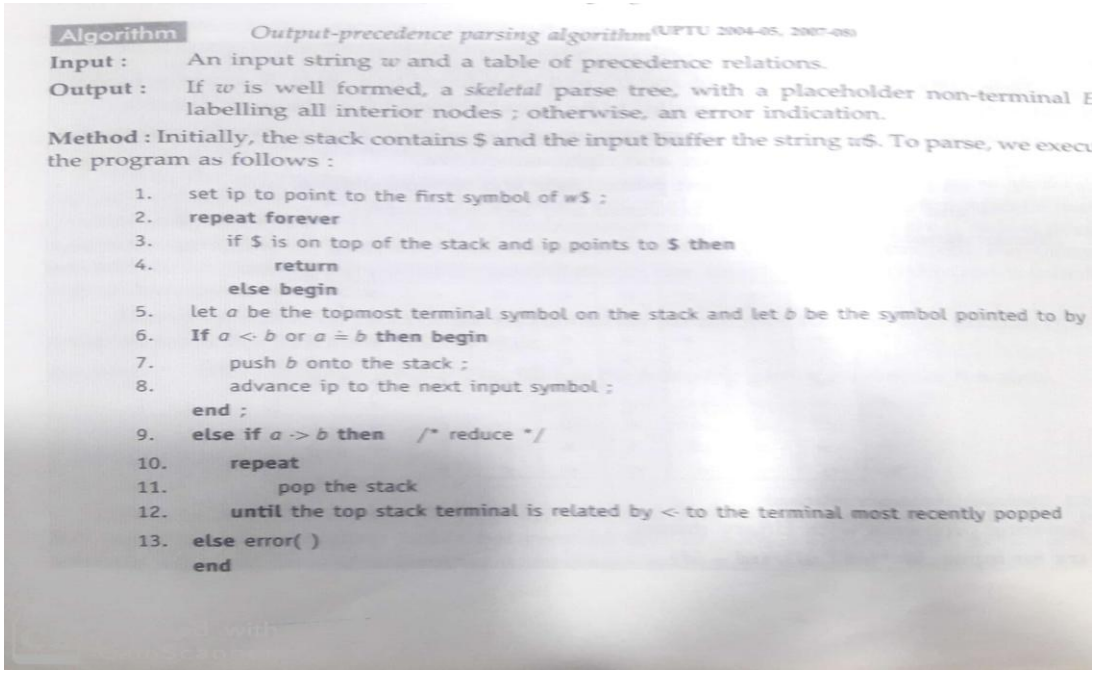
a.	**The following grammar abstract the dangling else problem** **S→iEts | iEtSeS | a** **E→b** **Check it is LL(1) grammar or not.**  **Since this table contain more than one entry in a cell so this is not a LL(1)Grammar.**	**5**	**2**	**3**

b.	**Explain YACC with example.**
	
	A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.
	Input File:
	YACC input file is divided in three parts.
	`/* definitions */`
	`....`
	`%%`
	`/* rules */`
	`....`
	`%%`
	`/* auxiliary routines */`
	`....`
	Input File: Definition Part:
	• The definition part includes information about the tokens used in the syntax definition:
	• `%token NUMBER`
	`%token ID`
	• Yacc automatically assigns numbers for tokens, but it can be overridden by `%token NUMBER 621`
	• Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should no overlap ASCII codes.
	• The definition part can include C code external to the definition of the parser and variable declarations, within **%{** and **%}** in the first column.
	• It can also include the specification of the starting symbol in the grammar: `%start nonterminal`
	Input File: Rule Part:
		5	**1**	**1**

	• The rules part contains grammar definition in a modified BNF form.
	• Actions is C code in { } and can be embedded inside (Translation schemes).
	Input File: Auxiliary Routines Part:
	• The auxiliary routines part is only C code.
	• It includes function definitions for every function needed in rules part.
	• It can also contain the main() function definition if the parser is going to be run as a program.
	• The main() function must call the function yyparse().
	Input File:
	• If yylex() is not defined in the auxiliary routines sections, then it should be included:
	`#include "lex.yy.c"`
	• YACC input file generally finishes with:
	`.y`
	Output Files:
	• The output of YACC is a file named **y.tab.c**
	• If it contains the **main()** definition, it must be compiled to be executable.
	• Otherwise, the code can be an external function definition for the function **int yyparse()**
	• If called with the **–d** option in the command line, Yacc produces as output a header file **y.tab.h** with all its specific definition (particularly important are token definitions to be included, for example, in a Lex input file).
	• If called with the **–v** option, Yacc produces as output a file **y.output** containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.			

5. Attempt any ONE part of the following : (1*5 = 5)

Q N	QUESTION	Marks	CO	BL

a.	**Give operator precedence table construction algorithm. Consider the following grammar to build up operator precedence table. Also parse the input string id+id*id** **E→E+E| E*E|(E)|id**  **Algorithm** *Output-precedence parsing algorithm* (UPTU 2004-05, 2007-08) **Input :** An input string w and a table of precedence relations. **Output :** If w is well formed, a *skeletal* parse tree, with a placeholder non-terminal E labelling all interior nodes ; otherwise, an error indication. **Method :** Initially, the stack contains $ and the input buffer the string w$. To parse, we execu the program as follows : 1. set ip to point to the first symbol of w$; 2. **repeat forever** 3. if $ is on top of the stack and ip points to $ **then** 4. **return** **else begin** 5. let a be the topmost terminal symbol on the stack and let b be the symbol pointed to by 6. **If** $a \lessdot b$ or $a \doteq b$ **then begin** 7. push b onto the stack ; 8. advance ip to the next input symbol ; **end ;** 9. **else if** $a \gtrdot b$ **then** /* reduce */ 10. **repeat** 11. pop the stack 12. **until** the top stack terminal is related by $\lessdot$ to the terminal most recently popped 13. **else** error( ) **end**	**5**	**2**	**2**
b.	**Explain the role of parser in complier construction. Also define the term Handle.**	**5**	**2**	**2**

5.2 ROLE OF PARSER

Parser plays an important role in the compiler design. It performs the following tasks :

1. It obtains a string of tokens from the lexical analyzer.
2. It groups the tokens appearing in the input in order to identify larger structures in the program.
3. It should report any syntax error in the program.
4. It should also recover from the errors so that it can continue to process the rest of the input.

Figure 5.1 shows the position of parser in the compiler.