



SUMMER- 18 EXAMINATION

Subject Name: System Programming

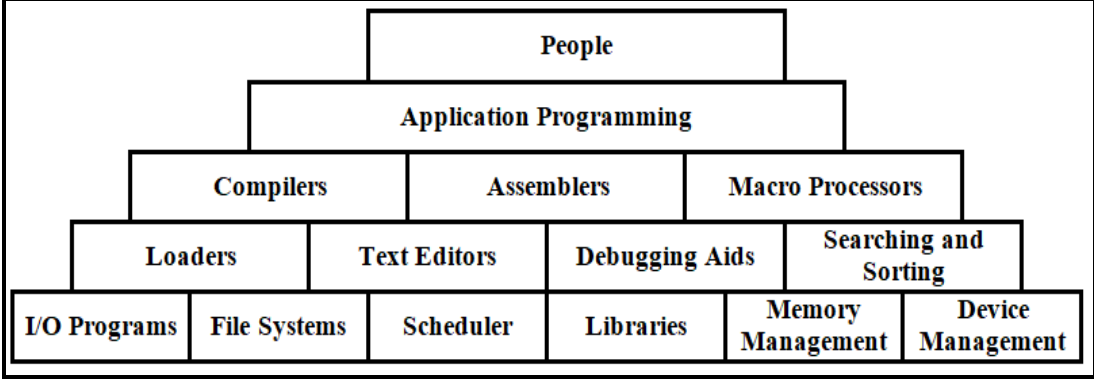
Model Answer

Subject Code:

17517

**Important Instructions to examiners:**

- 1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
- 2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
- 3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
- 4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
- 5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
- 6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
- 7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answers	Marking Scheme
1.	a)	<b>Attempt any three:</b>	<b>12 Marks</b>
	1)	<b>Define:</b> a) Allocation b) Relocation c) Linking d) Loading.	<b>4M</b>
	Ans:	a) <b>Allocation:</b> Allocate the space in the memory where the object programs can be loaded for execution. b) <b>Relocation:</b> Adjust the address sensitive instructions to the allocated space. c) <b>Linking:</b> Resolving external symbol reference d) <b>Loading:</b> Placing the object program in the memory in to the allocated space.	<b>(Each Definition: 1 mark)</b>
	2)	<b>Explain the foundation of system programing.</b>	<b>4M</b>
	Ans:	 <p>• System programs e.g. Compilers, loaders, macro processor, operating systems</p>	<b>(Diagram: 2 marks, Description: 2 marks)</b>



**SUMMER– 18 EXAMINATION**

**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

		<p>were developed to make computer better adapted to the needs of their users.</p> <ul style="list-style-type: none"><li>• Compiler is system program that accept people life languages and translate them into machine language.</li><li>• Loaders are system programs that prepare machine language programs for execution.</li><li>• Macro processors allow programmers to use abbreviations.</li><li>• Operating system and file system allows flexible to bring and retrieval of information.</li><li>• The productivity of each computer is heavily dependent upon the effectiveness, efficiency and sophistication of the system programs.</li></ul>																	
	3)	Compare the binary and linear search.	4M																
	Ans:	<table><tr><th>Binary Search</th><th>Linear Search</th></tr><tr><td>A binary search cut down search to half as soon as its finds middle of a sorted list.</td><td>A linear search scans one item at a time, without jumping to any item</td></tr><tr><td>Input data needs to be sorted in Binary Search</td><td>Input data can be unsorted in linear Search</td></tr><tr><td>Binary search performs ordering comparisons</td><td>Linear search performs equality comparisons</td></tr><tr><td>Time Complexity is <math>O(\log_2 N)</math></td><td>Time Complexity is <math>O(N)</math></td></tr><tr><td>Cannot be directly implemented on linked list.</td><td>Can be implemented on Array and Linked list.</td></tr><tr><td>Algorithm type is Divide and conquer in nature.</td><td>Algorithm type is iterative in nature.</td></tr><tr><td>Algorithm can conclude after only <math>\log_2 N</math> comparisons.</td><td>N comparisons are required in worst case.</td></tr></table>	Binary Search	Linear Search	A binary search cut down search to half as soon as its finds middle of a sorted list.	A linear search scans one item at a time, without jumping to any item	Input data needs to be sorted in Binary Search	Input data can be unsorted in linear Search	Binary search performs ordering comparisons	Linear search performs equality comparisons	Time Complexity is $O(\log_2 N)$	Time Complexity is $O(N)$	Cannot be directly implemented on linked list.	Can be implemented on Array and Linked list.	Algorithm type is Divide and conquer in nature.	Algorithm type is iterative in nature.	Algorithm can conclude after only $\log_2 N$ comparisons.	N comparisons are required in worst case.	(Any 4 points of comparison: 1 mark each)
Binary Search	Linear Search																		
A binary search cut down search to half as soon as its finds middle of a sorted list.	A linear search scans one item at a time, without jumping to any item																		
Input data needs to be sorted in Binary Search	Input data can be unsorted in linear Search																		
Binary search performs ordering comparisons	Linear search performs equality comparisons																		
Time Complexity is $O(\log_2 N)$	Time Complexity is $O(N)$																		
Cannot be directly implemented on linked list.	Can be implemented on Array and Linked list.																		
Algorithm type is Divide and conquer in nature.	Algorithm type is iterative in nature.																		
Algorithm can conclude after only $\log_2 N$ comparisons.	N comparisons are required in worst case.																		
	4)	Explain syntax phase of compiler with database and example.	4M																
	Ans:	<p>The function of the syntax phase is to recognize the major constructs of the language and to call the appropriate action routines that will generate the intermediate form to matrix for the constructs.</p> <p>Databases involved in syntax analysis are as follows:</p> <p><b>Uniform Symbol Table (“UST”):</b> It is created by the lexical analysis phase and containing the source program in the form of uniform symbols. It is used by the syntax and interpretation phases as the source of input to the stack. Each symbol from the UST enters the stack only once.</p> <p><b>Stack:</b> the stack is the collection of uniform symbols that is currently being worked on by the stack analysis and interpretation phase. The stack is organized on a Last In First Out (LIFO) basis. The term “Top of Stack” refers to the most recent entry and “Bottom of Stack” to the oldest entry.</p> <p><b>Reductions:</b> The syntax rules of the source language are contained in the reduction table. The syntax analysis phase is an interpreter driven by the reductions.</p>	(Description of Syntax phase: 2 marks, Database : 2 marks)																



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

	<p><b>Example:</b> // ***/ &lt;idn&gt; PROCEDURE/bgn_proc/S1 ***/4 &lt;any&gt;&lt;any&gt;&lt;any&gt;/ERROR/S2S1*/2</p> <p>These three reductions will be the first three of the set defined for the example. The interpretation is as follows:</p> <ol style="list-style-type: none"> <li>1. Start by putting the first three uniform symbols from the UST onto the stack.</li> <li>2. Test to see if top three elements are &lt;idn&gt;:PROCEDURE.</li> <li>3. If they are, call the begin procedure (bgn_proc) action routine, delete the label and get the next four uniform symbols from the UST onto the stack and go to reduction</li> <li>4. If not, call action routine ERROR, remove the third uniform symbol from the stack get one more from the UST, and go to reduction 2.</li> </ol> <p>The reduction state that all programs must start with a „&lt;label&gt;:PROCEDURE“.</p> <p>The syntax phase deletes the label and the „:“, gets four more tokens and interprets reduction 4, which will start parsing of the body of the procedure.</p> <p>If the first statement is not a &lt;label&gt;: PROCEDURE until a match is found or until all the symbols in the UST have been tried.</p>	
b)	Attempt any one :	6 Marks
1)	List components of system software and explain any two of them.	6M
Ans:	<ol style="list-style-type: none"> <li>1. Assembler</li> <li>2. Macros</li> <li>3. Compiler</li> <li>4. Loader</li> <li>5. Linker</li> </ol> <p><b>Assembler:</b> -Assembler is a language translator that takes as input assembly language program (ALP) and generates its machine language equivalent along with information required by the loader.</p> <p>ALP -&gt; Assembler -&gt; Machine language equivalent + Information required by the loader.</p> <p><b>Macro:</b> A macro is a rule or pattern that specifies how a certain input sequence (often a sequence of characters) should be mapped to a replacement output sequence (also often a sequence of characters) according to a defined procedure. The mappings process that instantiates (transforms) a macro use into a specific sequence is known as macro expansion. A facility for writing macros may be provided as part of a software application or as a part of a programming language. In the former case, macros are used to make tasks using the application less repetitive. In the latter case, they are a tool that allows a programmer to enable code reuse or even to design domain-specific languages.</p> <p>MACRO MACRO_NAME ...</p>	(List of components of System software: 2 marks; Description of any two: 2 marks each)



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

	<p>...</p> <p><i>MEND</i></p> <p><b>Compiler:</b> A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for converting a source code is to create an executable program. E.g. Javac, TurboC, CC (used in Unix/Linux).</p> <p><b>Loader:</b> Loader is a system program which places program into the memory and prepares for execution. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code e.g. Boot Strap loader.</p> <p><b>Linker:</b> A linker which is also called binder or link editor is a program that combines object modules together to form a program that can be executed. Modules are parts of a program.</p>	
2)	<b>Explain working of macro-processor.</b>	<b>6M</b>
<b>Ans:</b>	<p>The assembly language programmer often finds it necessary to repeat some blocks of code many times in the course of a program. The block may consist of code to save or exchange sets of registers, for example, or code to set up linkages or perform a series of arithmetic operations. In this situation the programmer will find a macro instruction facility useful. Macro instructions (often called macros) are single-line abbreviations for groups of instructions. In employing a macro, the programmer essentially defines a single "instruction" to represent a block of code. For every occurrence of this one-line macro instruction in his program, the macro processing assembler will substitute the entire block.</p> <p>By defining the appropriate macro instructions, an assembly language programmer can tailor own higher level facility in a convenient manner, at no cost in control over the structure of his program. Programmer can achieve the conciseness and ease in coding of high level languages without losing the basic advantage of assembly language programming. Integral macro operations simplify debugging and program modification and they facilitate standardization. Many computer manufacturers use macro instructions to automate the writing of "tailored" operating systems in a process called systems generation.</p> <p>Macro instructions are usually considered an extension of the basic assembler language, and the macro processor is viewed as an extension of the basic assembler algorithm. As a form of programming language, however, macro instruction languages differ significantly from assembly languages and compiled algebraic languages. Important analogs are to be found in some high level languages and text editing systems.</p>	<b>(Working of macro processor: 4 marks, Syntax/Example: 2 marks)</b>



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

			<table><tr><th>Source</th><th>Expanded source</th></tr><tr><td>MACRO</td><td></td></tr><tr><td>INCR</td><td></td></tr><tr><td>A</td><td>1,DATA</td></tr><tr><td>A</td><td>2,DATA</td></tr><tr><td>A</td><td>3,DATA</td></tr><tr><td>MEND</td><td></td></tr><tr><td>.</td><td>.</td></tr><tr><td>INCR</td><td>{ A 1,DATA</td></tr><tr><td>.</td><td>A 2,DATA</td></tr><tr><td>.</td><td>A 3,DATA</td></tr><tr><td>INCR</td><td>{ A 1,DATA</td></tr><tr><td>.</td><td>A 2,DATA</td></tr><tr><td>.</td><td>A 3,DATA</td></tr><tr><td>DATA</td><td>DATA</td></tr><tr><td>DC</td><td>DC</td></tr><tr><td>F'5'</td><td>F'5'</td></tr><tr><td>.</td><td>.</td></tr></table>	Source	Expanded source	MACRO		INCR		A	1,DATA	A	2,DATA	A	3,DATA	MEND		.	.	INCR	{ A 1,DATA	.	A 2,DATA	.	A 3,DATA	INCR	{ A 1,DATA	.	A 2,DATA	.	A 3,DATA	DATA	DATA	DC	DC	F'5'	F'5'	.	.	
Source	Expanded source																																							
MACRO																																								
INCR																																								
A	1,DATA																																							
A	2,DATA																																							
A	3,DATA																																							
MEND																																								
.	.																																							
INCR	{ A 1,DATA																																							
.	A 2,DATA																																							
.	A 3,DATA																																							
INCR	{ A 1,DATA																																							
.	A 2,DATA																																							
.	A 3,DATA																																							
DATA	DATA																																							
DC	DC																																							
F'5'	F'5'																																							
.	.																																							

2.	Attempt any two :	16 Marks
----	-------------------	----------

1)	Draw flow-chart for Pass-I assembler.	8M
----	---------------------------------------	----

Ans:	<pre>graph TD     Start([Pass 1]) --&gt; LC0[LC &lt;- 0]     LC0 --&gt; ReadCard[Read Card]     ReadCard --&gt; SearchPseudo[Search Pseudo-op table]     SearchPseudo --&gt; SearchMachine[Search Machine-op table]     SearchMachine --&gt; LLength[L &lt;- Length]     LLength --&gt; ProcessLiterals[Process my literals, enter into literal table]     ProcessLiterals --&gt; IsSymbol{Is there symbol in label field}     IsSymbol -- YES --&gt; AssignLC[Assignment current value to LC to symbol]     IsSymbol -- NO --&gt; LCLC[L &lt;- LC + L]     LCLC --&gt; AssignLC     SearchPseudo --&gt; WhichOne{Which one}     WhichOne -- DS DC --&gt; AdjustLC[Adjust LC to proper alignment]     AdjustLC --&gt; LLengthData[L &lt;- Length of data field]     LLengthData --&gt; AssignLC     WhichOne -- EQ --&gt; EvalOperand[Evaluate Operand Field]     EvalOperand --&gt; AssignSymbol[Assign value to symbol in label field]     AssignSymbol --&gt; WriteCopy[Write copy of card on file for use by Pass 2]     WhichOne -- EN --&gt; AssignStorage[Assign storage location to literals]     AssignStorage --&gt; Rewind[Rewind and reset CODV]     Rewind --&gt; OnToPass[On to Pass]     OnToPass --&gt; WriteCopy</pre>	(Correct flowchart: 8 marks)
------	---	------------------------------



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

2)	<p><b>Explain following sort:</b></p> <p>a) Inter-change sort. b) Address calculation sort.</p>	8M
Ans:	<p><b>Interchange Sort:</b> There are a number of ways of doing sort, some simple and some complicated. This simple sort takes adjacent pairs of items in the table and puts them in order (interchanges them) as required. Such a sorting algorithm is not very efficient, but it is simple. Take an example to see how it works. Consider the table of 12 numbers shown in example; each column represents one pass over the numbers interchanging any two adjacent numbers that are out of order. This particular table is completely sorted in only seven passes over the data. In the worst case, N-1 (here, 11) passes would be necessary. The inter-change sort is thus able to exploit whatever natural order there may be in the table. Moreover, on each pass through the data at least one item is added to the bottom of the list in perfect order (in this case the 31 first, then 27, then 26, etc.).</p> <p>Hence, the sort could be made more efficient by</p> <ol style="list-style-type: none"> <li>1) Shortening the portion of the sorted list on each pass; and</li> <li>2) Checking for early completion. Such an optimized sort should require roughly <math>N*(N-1)/2</math> comparisons and thus should take a time roughly proportional to <math>N^2</math>.</li> </ol> <p>Example: -</p> <div data-bbox="394 1031 1281 1453"> </div> <p><b>Address Calculation Sort:</b></p> <p>This can be one of the fastest types of sorts if enough storage space is available. The sorting is done by transforming the key into an address in the table that “represents” the key.</p> <p>For example if the key were four characters long, one method of calculating the appropriate table address would be to divide the key by the table length in items, multiply by the length is a power of 2, then the division reduces to a shift. This sort would take only <math>N*</math> (time to calculate address) if it were known that no two keys would be assigned the same address. However, in general this is not the case and several keys will be reduced to the same address.</p> <p>Therefore, before putting an item at the calculated address it is necessary to first check whether that location is already occupied. If so, the item is compared with the one that is already there, and a linear search in the correct direction is performed to</p>	<p>(Interchange sort: 4 marks, Address calculation: sort 4 marks)</p>



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

find the correct place for the new item.

If there will be 'n' empty space in which to put the item in order. Otherwise, it will be necessary to move some previous entries to make room.

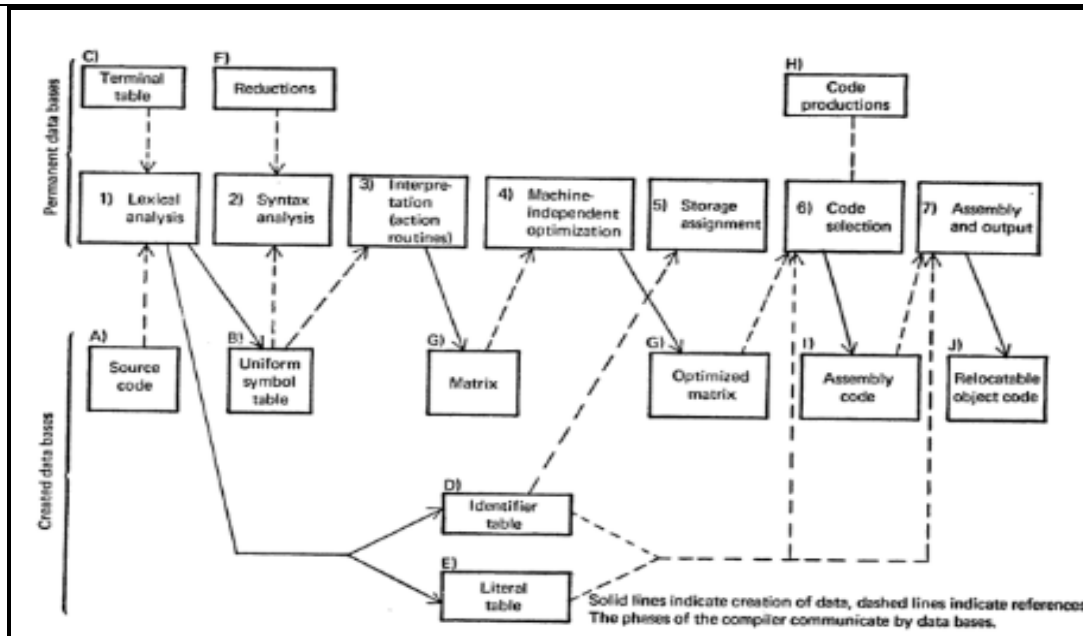
Example:

Data number	=	1	2	3	4	5	6	7	8	9	10	11	12
Data	=	19	13	05	27	01	26	31	16	02	09	11	21
Calculated address	=	6	4	1	9	0	8	10	5	0	3	3	7
Table	=												
0		---	---	---	---	01	01	01	01	*01	01	01	01
1		---	---	05	05	05	05	05	05	02	02	02	02
2		---	---	---	---	---	---	---	---	05	*05	05	05
3		---	---	---	---	---	---	---	---	09	*09	09	09
4		---	13	13	13	13	13	13	13	13	13	11	11
5		---	---	---	---	---	---	---	16	16	16	13	13
6		19	19	19	19	19	19	19	19	19	19	16	16
7		---	---	---	---	---	---	---	---	---	---	19	*19
8		---	---	---	---	26	26	26	26	26	26	26	21
9		---	---	---	27	27	27	27	27	27	27	27	26
10		---	---	---	---	---	31	31	31	31	31	31	27
11		---	---	---	---	---	---	---	---	---	---	---	31

3) Draw block diagram of phases of compiler.

8M

Ans:



(Block diagram of phases of compiler: 8 marks)

3. Attempt any four:

16 Marks

1) What is system software? List three types of system program.

4M

Ans: **System Software:** System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications.

(2 marks for definition, 2 marks for types)



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

	<p><b>Types of System programs (software):</b></p> <ul style="list-style-type: none"><li><b>i. Assemblers:</b> The program known as assembler is written to automate the translation of assembly language to machine language.</li><li><b>ii. Compilers:</b> These are system programs that accept people like languages and translate them into machine language.</li><li><b>iii. Loaders:</b> These are system programs that prepare machine language program for execution.</li><li><b>iv. Macro processors:</b> allow programmers to use abbreviation.</li><li><b>v. Operating system:</b> it is a system program that interfaces hardware and application software</li><li><b>vi. Parser:</b> It is system software that constructs the parsing tree to identify syntactical errors in given statement.</li><li><b>vii. The BIOS (basic input/output system):</b> gets the computer system started after you turn it on and manages the data flow between the operating system and attached devices such as the hard disk, video adapter, keyboard, mouse, and printer.</li></ul>																					
2)	<b>How to improve the assembler design?</b>	<b>4M</b>																				
Ans:	<p><b>Look for Modularity</b></p> <ul style="list-style-type: none"><li>• After designing algorithms for pass 1 and 2, we can review these algorithms to improve our design, by looking for functionalities that can be isolated.</li><li>• Modules/functions can be multi-use or unique.</li><li>• Lets look at our algorithms for passes 1 &amp; 2 and see if we can find a logical separation and put them in the following format.</li></ul> <table border="1"><tr><td><b>Function</b></td></tr><tr><td><b>Name</b></td></tr></table> <ul style="list-style-type: none"><li>• Where name is the name assigned to the function like MOTGET, EVAL, PRINT, POTGET etc.</li><li>• Accordingly we can list some logical modules that may be isolated in passes 1 &amp; 2.</li><li>• These functions are more or less indicated in the flow chart for the algorithms in both passes.</li><li>• The tables next summarize functions we may consider for modularity, isolating from the rest of the algorithm so that the module will be autonomous in its processing.</li><li>• Pass 1 Functions that may be considered for isolation</li></ul> <table border="1"><tr><th>No</th><th>Module</th><th>Description</th></tr><tr><td>1</td><td>READ1</td><td>Read the next instruction from source code</td></tr><tr><td>2</td><td>POTGET1</td><td>Search the pass 1 pseudo-op table (POT)</td></tr><tr><td>3</td><td>MOTGET1</td><td>Search MOT for a match with the current instruction</td></tr><tr><td>4</td><td>STSTO</td><td>Store label and associated value in ST</td></tr><tr><td>5</td><td>LTSTO</td><td>Store literals in LT. Do not store same literal</td></tr></table>	<b>Function</b>	<b>Name</b>	No	Module	Description	1	READ1	Read the next instruction from source code	2	POTGET1	Search the pass 1 pseudo-op table (POT)	3	MOTGET1	Search MOT for a match with the current instruction	4	STSTO	Store label and associated value in ST	5	LTSTO	Store literals in LT. Do not store same literal	
<b>Function</b>																						
<b>Name</b>																						
No	Module	Description																				
1	READ1	Read the next instruction from source code																				
2	POTGET1	Search the pass 1 pseudo-op table (POT)																				
3	MOTGET1	Search MOT for a match with the current instruction																				
4	STSTO	Store label and associated value in ST																				
5	LTSTO	Store literals in LT. Do not store same literal																				





SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

		twice.
6	WRITE1	Write a copy of the assembly source for use by pass 2
7	DLENGTH	Scan operand of DS, DC to determine storage required
8	EVAL	Evaluate arithmetic expression consisting of constants and symbols (eg. 6, ALPHA, 4*BETA ....)
9	STGET	Search ST for entry corresponding to specific symbol (used by STSTO and EVAL)
10	LITASS	Assign storage locations to each literal in the LT (may use DLENGTH)

- Pass 2 functions that may be considered for isolation

No	Module	Description
1	READ2	Read the next instruction from copy of source code
2	POTGET2	Search the pass 2 pseudo-op table (POT)
3	MOTGET2	Search MOT for a match with the current instruction
4	EVAL	Evaluate arithmetic expression consisting of constants and symbols (eg. 6, ALPHA, 4*BETA ....)
5	PUNCH	Convert generated instruction to appropriate format
6	PRINT	Convert generated code and location to character format
7	DLENGTH	Scan operand of DS, DC to determine storage required
8	DCGEN	Process the fields of DC to generate the object code (uses EVAL and PUNCH)
9	BTSTO	Enter data into appropriate entry in BT
10	BTDROP	Enter 'unavailable' indicator into appropriate entry in BT
11	BTGET	Convert effective address into base and displacement by searching BT for available base registers
12	LTGEN	Generate code for literals (use DCGEN)

- Each of these functions should go through the entire design process (problem statement, data base, algorithm and modularity).



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

	<ul style="list-style-type: none"><li>• These functions can be implemented as:<ul style="list-style-type: none"><li>○ External subroutines</li><li>○ Internal subroutines and</li><li>○ Sections in pass 1 or pass 2 programs</li></ul></li><li>• In any case dividing a bigger problem into its parts (modularity) making solving the problem easier.</li><li>• Easier to handle small coordinated routines than a big single program which contains all these routines.</li></ul>	
3)	<b>State four function of compiler.</b>	<b>4M</b>
Ans:	<p>{{ <b>** NOTE: any relevant answer can be considered**</b> }}</p> <ol style="list-style-type: none"><li>1. A compiler accepts a program written in a high level language as input and produces its machine language equivalent as output.</li><li>2. Scan source programs and identify Tokens.</li><li>3. Analyze syntactic and semantic errors.</li><li>4. Generate optimized code for source program.</li><li>5. Assign storage for variables and literals, etc</li><li>6. Generate assembly language program.</li></ol>	<b>(4 marks for 4 functions)</b>
4)	<b>Write function of ESD, RLD, TXT and END.</b>	<b>4M</b>
Ans:	<ol style="list-style-type: none"><li>1. <b>ESD card:</b><ul style="list-style-type: none"><li>• The ESD card contains the information necessary to build the external symbol. The external symbols are symbols that can be referred beyond the subroutine level. The normal labels in the source program are used only by the assembler.</li></ul></li><li>2. <b>RLD card:</b> contain the following information<ul style="list-style-type: none"><li>• The location and length of each address constant that needs to be changed for relocation or linking.</li><li>• The external symbol by which the address constant should be modified.</li><li>• The operation to be performed.</li></ul></li><li>3. <b>TXT card:</b><ul style="list-style-type: none"><li>• The TXT card contains the blocks of data and the relative address at which data is to be placed. Once the loader has decided where to load the program, it adds the Program Load Address (PLA) to relative address. The data on the TXT card may be instruction, non-related data or initial values of address constants.</li></ul></li><li>4. <b>END card :</b><ul style="list-style-type: none"><li>• The END card specifies the end of the object deck.</li></ul></li></ol>	<b>(Function of each card : 1 mark)</b>
5)	<b>Explain concept of top-down parser.</b>	<b>4M</b>
Ans:	<p><b>Top-down Parser</b></p> <ul style="list-style-type: none"><li>• When the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input, it is called top-down parsing.</li></ul>	<b>(Description: 4 marks)</b>



SUMMER- 18 EXAMINATION

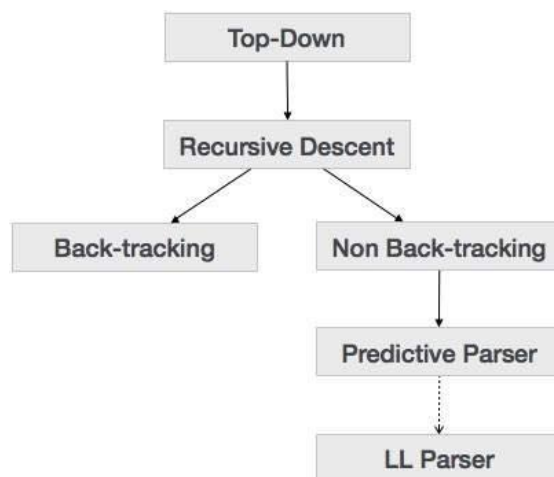
Subject Name: System Programming

Model Answer

Subject Code:

17517

The types of top-down parsing are depicted below:



- **Recursive Descent Parsing**

- Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right.
- It uses procedures for every terminal and non-terminal entity.
- This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking.
- A form of recursive-descent parsing that does not require any back-tracking is known as **predictive parsing**. This parsing technique is regarded recursive as it uses context-free grammar which is recursive in nature.

- **Back-tracking**

- Top- down parsers start from the root node (start symbol) and match the input string against the production rules to replace them (if matched).
- The following example of CFG:  
$$S \rightarrow rXd|rZd$$
$$X \rightarrow oa|ea$$
$$Z \rightarrow ai$$
- **For an input string: read, a top-down parser, will behave like this:** It will start with S from the production rules and will match its yield to the left-most letter of the input, i.e. 'r'. The very production of S ( $S \rightarrow rXd$ ) matches with it. So the top-down parser advances to the next input letter (i.e. 'e'). The parser tries to expand non-terminal 'X' and checks its production from the left ( $X \rightarrow oa$ ). It does not match with the next input symbol. So the top-down parser backtracks to obtain the next production rule of X, ( $X \rightarrow ea$ ). Now the parser matches all the input letters in an ordered manner. The string is accepted.

- **Predictive Parser**

- Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string.
- The predictive parser does not suffer from backtracking.
- To accomplish its tasks; the predictive parser uses a look-ahead pointer, which points to the next input symbols.



**SUMMER– 18 EXAMINATION**

**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

		<ul style="list-style-type: none"><li>○ To make the parser back-tracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar. Predictive parsing uses a stack and a parsing table to parse the input and generate a parse tree. Both the stack and the input contains an end symbol \$ to denote that the stack is empty and the input is consumed. The parser refers to the parsing table to take any decision on the input and stack element combination.</li><li>● <b>LL Parser</b><ul style="list-style-type: none"><li>○ An LL Parser accepts LL grammar.</li><li>○ LL grammar is a subset of context-free grammar but with some restrictions to get the simplified version, in order to achieve easy implementation.</li><li>○ LL grammar can be implemented by means of both algorithms namely, recursive-descent or table-driven.</li><li>○ LL parser is denoted as LL(k).<ul style="list-style-type: none"><li>▪ The first L in LL(k) is parsing the input from left to right, the second L in LL(k) stands for left-most derivation and k itself represents the number of look ahead. Generally k = 1, so LL(k) may also be written as LL(1).</li></ul></li></ul></li></ul>					
4.	a)	<b>Attempt any three :</b>	<b>12 Marks</b>				
	1)	<b>State and explain need of GEST and LESA.</b>	<b>4M</b>				
	<b>Ans:</b>	<p><b>Global External Symbol Table (GEST)</b></p> <ul style="list-style-type: none"><li>● It is used to store external symbols defined by means of Segment Definition (SD), or Local Definition (LD) entry on an External Symbol Dictionary(ESD) card.</li><li>● When these symbols are encountered in pass 1, they are assigned an absolute core address.</li><li>● This address is stored, along with the symbol, in the GEST.</li><li>● The GEST has same general use and characteristics as assembler’s symbol table.</li></ul> <p style="text-align: center;">←———— 12 bytes per entry —————→</p> <table border="1" style="margin-left: auto; margin-right: auto;"><tr><th><b>External Symbol (8 bytes) (characters)</b></th><th><b>Assigned core address (4 bytes) (decimal)</b></th></tr><tr><td> </td><td> </td></tr></table> <p style="text-align: center;"><b>Global External Symbol Table format</b></p> <p><b>Local External Symbol Array (LESA):</b></p> <ul style="list-style-type: none"><li>● It is necessary to establish a correspondence between ID number on an RLD card and the absolute core address value.</li><li>● The ESD card contains ID numbers and symbols they corresponds to, while the information relating these symbols to absolute core address values may be found in the GEST.</li></ul>	<b>External Symbol (8 bytes) (characters)</b>	<b>Assigned core address (4 bytes) (decimal)</b>			<b>(GEST: 2 marks, LESA: 2 marks)</b>
<b>External Symbol (8 bytes) (characters)</b>	<b>Assigned core address (4 bytes) (decimal)</b>						



**SUMMER– 18 EXAMINATION**

**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

	<ul style="list-style-type: none"><li>• In pass 2 of the loader the GEST and ESD information for each individual object deck is merged to produce the Local External Symbol Array (LESA) that directly relates ID numbers and values.</li><li>• Unlike the case of GEST, it is not necessary to search the LESA; given an ID number the corresponding value is written as LESA(ID) And can be immediately obtained.</li></ul> <div style="text-align: center;"><p>← 12 bytes per entry →</p><table border="1" style="margin: auto;"><tr><td>Assigned core address (4 bytes) (decimal)</td></tr><tr><td> </td></tr></table></div> <p style="text-align: center;"><b>Local External Symbol Array table format</b></p>	Assigned core address (4 bytes) (decimal)						
Assigned core address (4 bytes) (decimal)								
2)	<b>Explain storage allocation concept in compiler.</b>	<b>4M</b>						
Ans:	<ul style="list-style-type: none"><li>• The storage allocation phase first scans through the identifier table, assigning locations to the storage allocation phase first scans through the identifier table, assigning locations to each entry with a storage class of static.</li><li>• It uses a location counter, initialized at zero, to keep track of how much storage it has assigned.</li><li>• Whenever it finds a static variable in the scan, the storage allocation phase does the following four steps:<ol style="list-style-type: none"><li>1. Updates the location counter with any necessary boundary alignment.</li><li>2. Assigns the current value of the location counter to the address field of the variable.</li><li>3. Calculate the length of the storage needed by the variable (by examining its attributes).</li><li>4. Updates the location counter by adding this length to it. Once it has assigned relative address to all identifiers requiring STATIC storage locations, this phase creates a matrix entry:</li></ol></li></ul> <div style="text-align: center;"><table border="1" style="display: inline-table;"><tr><td>STATIC</td><td>Size</td><td> </td></tr></table></div> <ol style="list-style-type: none"><li>5. This allows code generation to generate the proper amount of storage. For each variable that requires initialization, the storage allocation phase generates a matrix entry:</li></ol> <div style="text-align: center;"><table border="1" style="display: inline-table;"><tr><td>Initialize</td><td>Variable</td><td> </td></tr></table></div> <ol style="list-style-type: none"><li>6. This tells code generation to put into the proper storage location the initial values that the action routines saved in the identifier table. A similar scan of the identifier table is made for automatic storage and controlled storage. The scan enters relative location for each entry. An “automatic” entry and a “controlled”</li></ol>	STATIC	Size		Initialize	Variable		<b>(Description: 4 marks)</b>
STATIC	Size							
Initialize	Variable							



**SUMMER- 18 EXAMINATION**

**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

	<p>entry are also made in the matrix. Code generation use the relative location entry to generate the address part of instructions. No storage is generated at compile time for automatic or controlled. However, the matrix entry automatic does cause code to be generated that allocates this storage at execution time, i.e., when the generated code is executed, it allocates automatic storage.</p> <table border="1"> <tr> <td>LIT</td><td>Size</td><td></td></tr> </table> <p>7. The literal table is similarly scanned and location are assigned to each literal, and a matrix entry is made. Code generation generates storage for all literals in the static area and initializes the storage with the values of the literals. Temporary storage is handled differently since each source statement may reuse the temporary storage (intermediate matrix result area) of the previous source statement. A computation is made of the temporary storage that is required for each source statement. The statement required the greatest amount of temporary storage determines the amount that will be required for the entire program. A matrix entry is made of the form this enables the code generation phase to generate code to create the proper amount of storage.</p> <table border="1"> <tr> <td>AUTOMATIC</td><td>Size</td><td></td></tr> </table> <p>8. Temporary storage is automatic since it is only referenced by the source program and only needed while the source program is active.</p> <p><b>The purpose of this phase is to:</b></p> <ol style="list-style-type: none"> <li>1. Assign storage to all variables referenced in the source program.</li> <li>2. Assign storage to all temporary locations that are necessary for intermediate result, e.g the results of matrix lines. These storage references were reserved by the interpretation phase and did not appear in the source code.</li> <li>3. Assign storage to literals.</li> <li>4. Ensure that the storage is allocated and appropriate locations are initialized (Literals and any variables with the initial attribute)</li> </ol>	LIT	Size		AUTOMATIC	Size		
LIT	Size							
AUTOMATIC	Size							
3)	<b>Describe lexical phase of compiler.</b>	<b>4M</b>						
Ans:	<ul style="list-style-type: none"> <li>• The first phase of compiler is lexical analysis. It works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens as: <b>&lt;token-name, attribute-value&gt;</b></li> </ul> <p><b>Algorithm of Lexical Analysis phase of compiler is as follows</b></p> <ul style="list-style-type: none"> <li>○ The first tasks of the lexical analysis algorithm are to the input character string into token.</li> <li>○ The second is to make the appropriate entries in the tables.</li> <li>○ A token is a substring of the input string that represents a basic element of the language. It may contain only simple characters and may not include another token. To the rest of the compiler, the token is the smallest unit of currency. Only lexical analysis and the output processor of the assembly phase concern themselves with such elements as characters. Uniform symbols are the terminal</li> </ul>	<b>(Description: 4 marks)</b>						



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

symbols for syntax analysis.

- Lexical analysis recognizes three types of token:
  - Terminal symbols,
  - possible identifiers, and
  - Literals.
- It checks all tokens by first comparing them with the entries in the terminal table. Once a match is found, the token is classified as a terminal symbol and lexical analysis creates a uniform symbol of type “TRM” and inserts it in the uniform symbol table. If a token is not a terminal symbol, lexical analysis proceeds to classify it as a possible identifier or literal. Those tokens that satisfy the lexical rules for forming identifiers are classified as “possible identifiers”.

**Databases used in lexical analysis are:**

- Source program:** original form of program; appears to the compiler as a string of character
- Terminal table:** a permanent data base that has an entry for each terminal symbol. Each entry consists of the terminal symbol, an indication of its classification, and its precedence.

Symbol	Indicator	Precedence
--------	-----------	------------

- Literal table:** created by lexical analysis to describe all literals used in the source program. There is one entry for each literal, consisting of a value, a number of attributes, an address denoting the location of the literal at execution time, and other information.

Literal	Base	Scale	Precision	Other information	address
---------	------	-------	-----------	-------------------	---------

- Identifier table:** created by lexical analysis to describe all identifiers used in the source program. There is one entry for each identifier. Lexical analysis creates the entry and places the name of identifier into that entry. The pointer points to the name in the table of names. Later phases will fill in the data attributes and address of each identifier.

Name	Data attribute	address
------	----------------	---------

- Uniform Symbol table:** created by lexical analysis to represent the program as a string of tokens rather than of individual characters. Each uniform symbol contains the identification of the table of which a token is a member.

Table	Index
-------	-------



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

4)	<b>Explain concept of bottom-up parser.</b>	<b>4M</b>
<b>Ans:</b>	<ul style="list-style-type: none"> <li>Bottom-up parsing starts from the leaf nodes of a tree and works in upward direction till it reaches the root node.</li> <li>Here, we start from a sentence and then apply production rules in reverse manner in order to reach the start symbol.</li> <li>The image given below depicts the bottom-up parsers available.</li> </ul> <div style="text-align: center;"> <pre> graph TD     A[Bottom-Up] --&gt; B[Shift-Reduce]     B --&gt; C[LR Parsing]     C --&gt; D[SLR Parsing]     C --&gt; E[LR Parser]     C --&gt; F[LALR Parser] </pre> </div> <ul style="list-style-type: none"> <li><b>Shift-Reduce Parsing</b> <ul style="list-style-type: none"> <li>Shift-reduce parsing use two unique steps for bottom-up parsing. These steps are known as shift-step and reduce-step.</li> </ul> </li> <li><b>LR Parser</b> <ul style="list-style-type: none"> <li>The LR parser is a non-recursive, shift-reduce, bottom-up parser. It uses a wide class of context-free grammar which makes it the most efficient syntax analysis technique. LR parsers are also known as LR<sub>k</sub> parsers, where <b>L</b> stands for left-to-right scanning of the input stream; <b>R</b> stands for the construction of right-most derivation in reverse, and <b>k</b> denotes the number of look ahead symbols to make decisions.</li> <li><b>There are three widely used algorithms available for constructing an LR parser:</b> <ul style="list-style-type: none"> <li>SLR1 – Simple LR Parser</li> <li>LR1 – LR Parser</li> <li>LALR1 – Look-Ahead LR Parser</li> </ul> </li> </ul> </li> </ul>	<b>(Description: 4 marks)</b>
b)	<b>Attempt any one:</b>	<b>6 Marks</b>
1)	<b>Define macro and explain conditional macro expansion.</b>	<b>6M</b>
<b>Ans:</b>	<p><b>Macro</b></p> <ul style="list-style-type: none"> <li>A macro is a rule or pattern that specifies how a certain input sequence (often a sequence of characters) should be mapped to a replacement output sequence (also often a sequence of characters) according to a defined procedure.</li> <li>The mappings process that instantiates (transforms) a macro use into a specific sequence is known as macro expansion.</li> </ul>	<b>(Macro definition :2 marks, conditional Macro expansion: 4 marks)</b>





SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

- A facility for writing macros may be provided as part of a software application or as a part of a programming language. In the former case, macros are used to make tasks using the application less repetitive.
- In the latter case, they are a tool that allows a programmer to enable code reuse or even to design domain-specific languages.

MACRO

MACRO\_NAME

...

...

...

MEND

**Conditional macro expansion.**

- Two important macro-processor pseudo-ops AIF and AGO permit conditional reordering of the sequence of macro expansion. This allows conditional selection of the machine instructions that appear in expansions of Macro call. Consider the following program.

Loop 1 A1, DATA1

A2, DATA2

A3, DATA3

.

.

Loop 2 A1, DATA3

A2, DATA2

.

.

Loop 3 A1, DATA1

.

.

DATA1 DC F'5'

DATA2 DC F'10'

DATA3 DC F'15'

- In the below example, the operands, labels and the number of instructions generated change in each sequence. The program can be written as follows:-

.

.

.

MACRO

&ARG0 VARY &COUNT, &ARG1, &ARG2, &ARG3

&ARG0 A1, &ARG1

AIF (&COUNT EQ 1).FINI

A2, &ARG2

AIF (&COUNT EQ 2).FINI

A3, &ARG3

.FINI MEND EXPANDED SOURCE



SUMMER– 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

	<pre> .. .. .. LOOP1 VARY 3, DATA1, DATA2, DATA3 LOOP1 A1,DATA1 . A 2,DATA2 . A 3,DATA3 .. LOOP2 VARY 2,DATA3,DATA2 LOOP2 A 1,DATA3 . A 2,DATA2 .. .. LOOP3 VARY 1,DATA1 LOOP3 A 1,DATA1 . . . DATA1 DC F'5' DATA2 DC F'10' DATA3 DC F'15' </pre> <ul style="list-style-type: none"> <li>Labels starting with a period (.) such as .FINI are macro labels and do not appear in the output of the macro processor.</li> <li>The statement AIF (&amp; COUNT EQ1) .FINI direct the macro processor to skip to the statement. Labeled .FINI if the parameter corresponding to &amp; COUNT is a1; otherwise the macro processor is to continue with the statement following the AIF pseudo-ops. AIF is conditional branch pseudo ops it performs an arithmetic test and branches only if the tested condition is true.</li> <li>AGO is an unconditional branch pseudo-ops or 'Go to' statement. It specifies label appearing on some other statement. AIF &amp; AGO controls the sequence in which the macro processor expands the statements in macro instructions.</li> </ul>	
2)	<b>Compare advantages and disadvantages of top-up and bottom-up parser.</b>	<b>6M</b>
Ans:	<p><b>Top-down Parser:</b></p> <p><b>Advantages:-</b></p> <ol style="list-style-type: none"> <li>1. It is easy to implement</li> <li>2. It never wastes time on sub trees that cannot have an S at the root. Bottom up parsing does this.</li> </ol> <p><b>Disadvantages:-</b></p> <ol style="list-style-type: none"> <li>1. It is not efficient parsing method as compare to bottom up parse</li> <li>2. It cannot handle left recursion</li> <li>3. It is not applicable to large scale of grammar.</li> <li>4. Wastes time on trees that don't match the input (compare the first word of the input with the leftmost branch of the tree). Bottom-up parsing doesn't do this.</li> </ol> <p><b>Bottom-up parser</b></p> <p><b>Advantages:-</b></p> <ol style="list-style-type: none"> <li>1. It is efficient parsing method.</li> <li>2. Left recursion framer is handled by bottom up parser.</li> <li>3. It is applicable to large scale of grammar.</li> </ol>	



SUMMER– 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

		<b>Disadvantages:-</b> <ol style="list-style-type: none"> <li>1. It wastes time on sub trees that cannot have an S at the root.</li> <li>2. Bottom-up parse postpones decisions about which production rule to apply until it has more data than was available to top-down.</li> </ol>	
5.		<b>Attempt any two:</b>	<b>16 Marks</b>
	1)	<b>Explain following term in detail :</b> <ol style="list-style-type: none"> <li>a) Compile and go loader</li> <li>b) Absolute loader.</li> </ol>	<b>8M</b>
	Ans:	<p><b>Compile and go loader</b></p> <p>One method of performing the loader functions is to have the assembler run in one part of memory and places the assembled machine instructions and data, as they are assembled, directly into their assigned memory locations.</p> <p>As a usual practice one method of performing the loader functions is to have to assemble run in one part of memory and place the assembled machine instructions and data they are assembled, directly into their assigned memory locations.</p> <div data-bbox="388 898 1205 1476" data-label="Diagram"> <pre> graph LR     A[Source program deck] --&gt; B[Compile-and-go translator (e.g., assembler)]     B --&gt; C     subgraph Memory         direction TB         C[Program loaded in memory]         D[Assembler]         E[ ]     end     style C fill:#ccc     style D fill:#fff     style E fill:#ccc     style Memory fill:none,stroke:none </pre> <p>When the assembly is completed the assembler causes transfer to the instruction of the program. This is a simple solution, involving no extra procedures. It is used by the WATFOR FORTRAN compiler and several other language processors.</p> <p>Such a loading scheme is commonly called “compile-and-go” or “assembler – and –go”. It is relatively easy to implement. The assembler simply places the code into core, and the “loader” consists of one instruction that transfers to the starting instruction of the newly assembled program.</p> <p><b>Absolute loader</b></p> <p>The simplest type of loader scheme, which fits the general model, is called an absolute loader. In this scheme the assembler outputs the machine language</p> </div>	<p><b>(Compile and go loader: 4 marks; Absolute loader 4 marks)</b></p>

**SUMMER- 18 EXAMINATION**

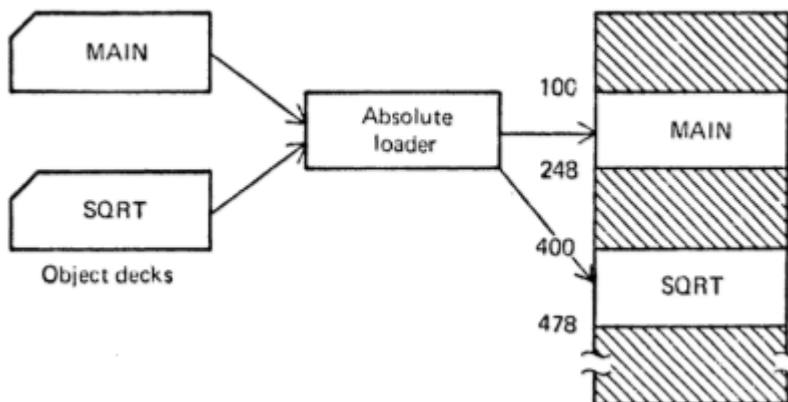
**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

translation of the source program in almost the same form as in the "assemble-arid-go" scheme, except that the data is punched on cards (object deck) instead of being placed directly in memory. The loader in turn simply accepts the machine language text and places it into core at the location pre-scribed by the assembler. This scheme makes more cores available to the user since the assembler is not in memory at load time.



Absolute loaders are simple to implement but they do have several disadvantages. First, the programmer must specify to the assembler the address in core where the program is to be loaded. Furthermore, if there are multiple sub-routines, the programmer must remember the address of each and use that absolute address explicitly in his other subroutines to perform subroutine linkage.

**2) Explain simple machine independent optimization algorithm with an example.**

**8M**

**Ans:** Following are four commonly used algorithms(techniques) for machine independent optimization:-

- 1) Elimination of common sub expression
- 2) Compile time compute.
- 3) Boolean expression optimization.
- 4) Move invariant computations outside of loops.

**1) Elimination of common sub expression:** -The elimination of duplicate matrix entries can result in a more can use and efficient object program. The common sub-expression must be identical and must be in the same statement.

- i. The elimination algorithm is as follows:-
- ii. Place the matrix in a form so that common sub-expression can be recognized.
- iii. Recognize two sub-expressions as being equivalent.
- iv. Eliminate one of them.
- v. After the rest of the matrix to reflect the elimination of this entry.

**(Each method: 2 marks)**



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

Source code

```

B = A
A = C * D * (D * C + B)

```

Matrix before optimization

M1	=	B	A	0	2
M2	*	C	D	1	3
M3	*	D	C	2	4
M4	+	M3	B	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

Matrix after step 1 and 2

M1	=	B	A	0	2
M2	*	C	D	1	3
M3	*	C	D	2	4
M4	+	B	M3	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

Statement

Statement

Matrix after step 4

M1	=	B	A	0	2
M2	*	C	D	1	4
M3	*	C	D	2	4
M4	+	B	M2	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

2) **Compile time compute:** - Doing computation involving constants at compile time save both space and execution time for the object program.

The algorithm for this optimization is as follows:-

- Scan the matrix.
- Look for operators, both of whose operands were literals.
- When it found such an operation it would evaluate it, create new literal, delete old line
- Replace all references to it with the uniform symbol for the new literal.
- Continue scanning the matrix for more possible computation.

For e.g.-  $A = 2 * 276 / 92 * B$

The compile time computation would be

**Matrix Before optimization**

M <sub>1</sub>	*	2	276
M <sub>2</sub>	/	M <sub>1</sub>	92
M <sub>3</sub>	*	M <sub>2</sub>	B
M <sub>4</sub>	=	A	M <sub>3</sub>

**Matrix After optimization**

M <sub>1</sub>			
M <sub>2</sub>			
M <sub>3</sub>	*	6	B
M <sub>4</sub>	=	A	M <sub>3</sub>

3) **Boolean expression optimization:** - We may use the properties of boolean expression to shorten their computation.

e.g. In a statement If a OR b Or c,

Then ..... when a, b & c are expression rather than generate code that will always test each expression a, b, c. We generate code so that if a computed as true, then b OR c is not computed, and similarly for b.

4) **Move invariant computation outside of loops:** - If computation within a loop depends on a variable that does not change within that loop, then computation may be moved outside the loop.

This requires a reordering of a part of the matrix. There are 3 general problems that need to be solved in an algorithm.

- Recognition of invariant computation.
- Discovering where to move the invariant computation.
- Moving the invariant computation.



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

3)	Apply radix sort for following example: 227, 125, 02, 940, 1207,748, 1520.										8M																																																																																																																																																																																																																																																																								
Ans:	<p><b>Pass 1:</b> <b>Step 1:</b> - Equalize numbers to 4 digits. 0227, 0125, 0002, 0940, 1207. 0748, 1520. <b>Step 2:</b> - Put Numbers in associated place. Consider LSB, i.e. unit position.</p> <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>0227</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0227</td><td></td><td></td></tr><tr><td>0125</td><td></td><td></td><td></td><td></td><td></td><td>0125</td><td></td><td></td><td></td><td></td></tr><tr><td>0002</td><td></td><td></td><td>0002</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0940</td><td>0940</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1207</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1207</td><td></td><td></td></tr><tr><td>0748</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0748</td><td></td></tr><tr><td>1520</td><td>1520</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> <p><b>Step 3:</b> - Retrieve the data as per its appearance. 0940, 1520, 0002, 0125, 0227, 1207, 0748 <b>Pass 2:</b> <b>Step 4:</b> - Put Numbers in associated place. Consider 10<sup>th</sup> position.</p> <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>0940</td><td></td><td></td><td></td><td></td><td>0940</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1520</td><td></td><td></td><td>1520</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0002</td><td>0002</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0125</td><td></td><td></td><td>0125</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0227</td><td></td><td></td><td>0227</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1207</td><td>1207</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0748</td><td></td><td></td><td></td><td></td><td>0748</td><td></td><td></td><td></td><td></td><td></td></tr></table> <p><b>Step 5:</b> - Retrieve the data in reverse sequences. 0002,1207 1520, 0125, 0227, 0940, 0748 <b>Pass 3:</b> <b>Step 6:</b> - Put Numbers in associated place. Consider 100<sup>th</sup> position.</p> <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>0002</td><td>0002</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1207</td><td></td><td></td><td>1207</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1520</td><td></td><td></td><td></td><td></td><td></td><td>1520</td><td></td><td></td><td></td><td></td></tr><tr><td>0125</td><td></td><td>0125</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0227</td><td></td><td></td><td>0227</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0940</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0940</td></tr><tr><td>0748</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0748</td><td></td><td></td></tr></table> <p><b>Step 7:</b> - Retrieve the data in reverse sequences. 0002, 0125, 1207, 0227, 1520, 0748, 0940</p>											0	1	2	3	4	5	6	7	8	9	0227								0227			0125						0125					0002			0002								0940	0940										1207								1207			0748									0748		1520	1520											0	1	2	3	4	5	6	7	8	9	0940					0940						1520			1520								0002	0002										0125			0125								0227			0227								1207	1207										0748					0748							0	1	2	3	4	5	6	7	8	9	0002	0002										1207			1207								1520						1520					0125		0125									0227			0227								0940										0940	0748								0748			(Correct pass: 2 marks each)
	0	1	2	3	4	5	6	7	8	9																																																																																																																																																																																																																																																																									
0227								0227																																																																																																																																																																																																																																																																											
0125						0125																																																																																																																																																																																																																																																																													
0002			0002																																																																																																																																																																																																																																																																																
0940	0940																																																																																																																																																																																																																																																																																		
1207								1207																																																																																																																																																																																																																																																																											
0748									0748																																																																																																																																																																																																																																																																										
1520	1520																																																																																																																																																																																																																																																																																		
	0	1	2	3	4	5	6	7	8	9																																																																																																																																																																																																																																																																									
0940					0940																																																																																																																																																																																																																																																																														
1520			1520																																																																																																																																																																																																																																																																																
0002	0002																																																																																																																																																																																																																																																																																		
0125			0125																																																																																																																																																																																																																																																																																
0227			0227																																																																																																																																																																																																																																																																																
1207	1207																																																																																																																																																																																																																																																																																		
0748					0748																																																																																																																																																																																																																																																																														
	0	1	2	3	4	5	6	7	8	9																																																																																																																																																																																																																																																																									
0002	0002																																																																																																																																																																																																																																																																																		
1207			1207																																																																																																																																																																																																																																																																																
1520						1520																																																																																																																																																																																																																																																																													
0125		0125																																																																																																																																																																																																																																																																																	
0227			0227																																																																																																																																																																																																																																																																																
0940										0940																																																																																																																																																																																																																																																																									
0748								0748																																																																																																																																																																																																																																																																											



**SUMMER– 18 EXAMINATION**

**Subject Name: System Programming**

**Model Answer**

**Subject Code:**

**17517**

		<p><b>Pass 4:</b></p> <p><b>Step 8:</b> - Put Numbers in associated place. Consider 1000<sup>th</sup> position.</p> <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>0002</td><td>0002</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0125</td><td>0125</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1207</td><td></td><td>1207</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0227</td><td>0227</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1520</td><td></td><td>1520</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0748</td><td>0748</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0940</td><td>0940</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> <p><b>Step 9:</b> - Retrieve the data in reverse sequences.</p> <p>0002, 0125, 0227, 0748, 0940, 1207, 1520</p>		0	1	2	3	4	5	6	7	8	9	0002	0002										0125	0125										1207		1207									0227	0227										1520		1520									0748	0748										0940	0940										
	0	1	2	3	4	5	6	7	8	9																																																																																	
0002	0002																																																																																										
0125	0125																																																																																										
1207		1207																																																																																									
0227	0227																																																																																										
1520		1520																																																																																									
0748	0748																																																																																										
0940	0940																																																																																										
6.		<b>Attempt any four:</b>	<b>16 Marks</b>																																																																																								
	1)	<b>State and explain four basic task of Macro processor.</b>	<b>4M</b>																																																																																								
	<b>Ans:</b>	<p>1. <b>Recognize Macro Definitions:</b> A Macro instruction processor must recognize macro definitions identified by the MACRO and MEND pseudo ops. This task can be complicate when macro definitions appear within macros. When MACROs and MENDs are nested the macro processor must recognize the nesting and correctly match the last or outer MEND with the first MACRO. All of the intervening text, including nested MACROs and MENDs defines a single macro instruction.</p> <p>2. <b>Save the Macro Definitions:</b> The processor must store the macro instruction definitions, which it will need for expanding macro calls.</p> <p>3. <b>Recognize macro calls:</b> The processor must recognize macro calls that appear as operations mnemonics. This suggests that macro names be handled as a type op-code.</p> <p>4. <b>Expand Macro calls and substitute arguments:</b> The processor must substitute for dummy or macro definitions arguments the corresponding arguments from a macro call; the resulting symbolic text is then substitute for the macro call. This text of course, may contain additional macro definitions or calls.</p>	<b>(Description of each task:1 mark)</b>																																																																																								
	2)	<b>Explain data structure of Pass-I assembler.</b>	<b>4M</b>																																																																																								
	<b>Ans:</b>	<p>Data Structures used in assembler pass 1</p> <p>1. Input source program: Program written in high level language.</p> <p>2. A Location Counter (LC), used to keep track of each instruction’s location.</p> <p>3. A table, the Machine-operation Table (MOT), that indicates the symbolic mnemonic, for each instruction and its length (two, four, or six bytes)</p> <p>4. A table, the Pseudo-Operation Table (POT) that indicates the symbolic mnemonic and action to be taken for each pseudo-op in pass 1.</p> <p>5. A table, the Symbol Table (ST) that is used to store each label and its corresponding value.</p>	<b>(All data structures: 4 marks)</b>																																																																																								



SUMMER- 18 EXAMINATION

Subject Name: System Programming

Model Answer

Subject Code:

17517

		6. A table, the literal table (LT) that is used to store each literal encountered and its corresponding assignment location. A copy of the input to be used by pass: 2.	
	3)	<b>What are the advantages and disadvantages of BSS loader?</b>	<b>4M</b>
	<b>Ans:</b>	<p><b>Advantage of BSS loader:</b></p> <ol style="list-style-type: none"><li>1. Allows the programmer multiple procedure segments and multiple data segments and of giving him complete freedom in referencing data or instructions contained in other segments.</li><li>2. This provides flexible intersegment referencing and accessing ability, while at the same time allowing independent translations of programs.</li></ol> <p><b>Disadvantage of BSS loader</b></p> <ol style="list-style-type: none"><li>1. The transfer vector linkage is only useful for transfer and is not well suited for loading or storing external data (data located in another procedure segment).</li><li>2. The transfer vector increases the size of the object program in memory.</li><li>3. Finally BSS Loader processes procedure segments but does not facilitate access to data segments that can be shared. This last shortcoming is overcome in many BSS loaders by allowing one common data segment often called COMMON.</li></ol>	<b>(2 Advantages : 2 marks; 2 Disadvantages 2 marks )</b>
	4)	<b>Explain intermediate code generation in compiler.</b>	<b>4M</b>
	<b>Ans:</b>	<p>The purpose of the intermediate code generation phase is to produce the appropriate code (assembly or machine language). The intermediate code generation phase has the matrix as input. It uses the code productions (macro definitions) which define the operators that may appear in the matrix to produce code. It also references the identifier tables and literal tables in order to generate proper address and code conversions. More flexible definitions are required that will allow us to generate the more efficient code contained.</p> <p>Once the syntax analysis and semantic analysis, most compilers generate low-level, machine code, known as an intermediate code. This code is generated in intermediate code generator phase.</p> <p>This code has two essential properties:</p> <ul style="list-style-type: none"><li>○ It should be easy to produce.</li><li>○ It should be easy to translate to machine code.</li></ul> <p>The basic purpose of generating this code is ease of translation to machine code and hence it resembles assembly language code greatly.</p>	<b>(Description of inter-mediate code generation: 4 marks)</b>
	5)	<b>Explain overlay structure used in dynamic loading scheme.</b>	<b>4M</b>
	<b>Ans:</b>	<p>The subroutines of a program are needed at different times. For e.g. Pass 1 and pass 2 of an assembler are call other subroutine it is possible to produce an overlay structure that identifies mutually exclusive subroutines. In order for the overlay structure to work it is necessary for the module loader to the various procedures as they are needed. The portion of the loader that actually intercepts the “calls” and</p>	<b>(Description: 3 marks, Diagram: 1 mark)</b>





SUMMER- 18 EXAMINATION

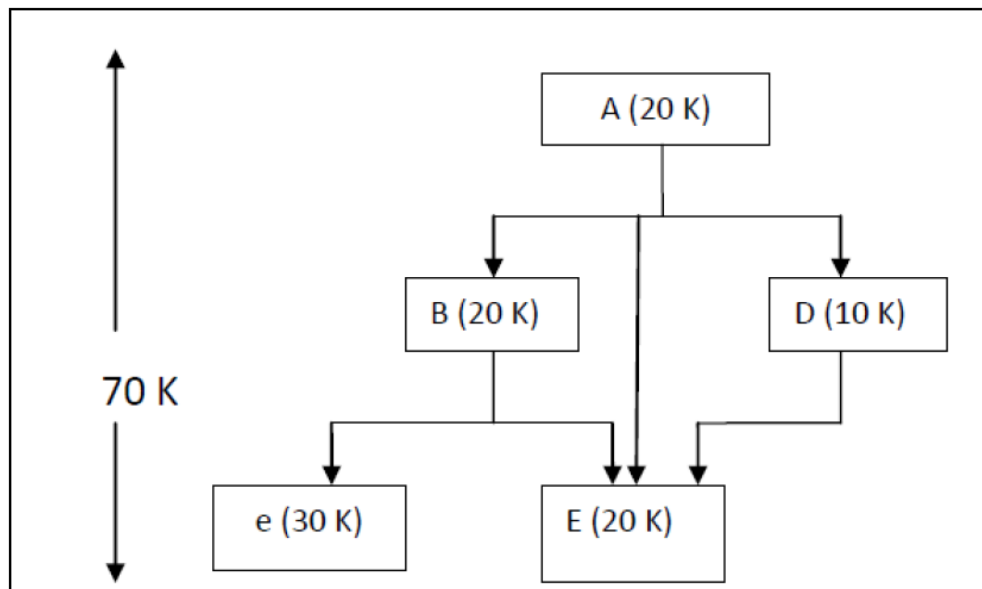
Subject Name: System Programming

Model Answer

Subject Code:

17517

loads the necessary procedure is collect the overlay supervision or simply the upper.



Above program consisting of five subprogram (A, B, C, D & E) that require look bytes of core. The arrow indicate that subprogram A only calls B, D and E; subprogram B only calls C and E; subprogram D only calls E; and subprogram C and E do not call any other routine procedures B and D are never in use at the same time; neither are C and E. If are load only those procedures that are actually to be used at any particular time, the amount of core needed is equal to the longest path of the overlay structure. This happens to be 70k. Overlay reduces the memory requirement of a program. It also makes it possible to execute program where size exceeds the amount of memory which can be allocated to them. For the execution of overlay structured program, the root is loaded in memory and given control for the execution. Other overlays are loaded as and when headed. Loading of an overlay overwrite a previously loaded overlay with the same load origin.