Department of Computer Science

Gujarat University

Certificate

Roll No: **36** Seat No: _____

This is to certify that Mr./Ms. **PREKSHA K. SHETH** student of MCA Semester – III has duly completed his/her term work for the semester ending in December 2020, in the subject of **SYSTEM SOFTWARE** towards partial fulfillment of his/her Degree of Masters in Computer Applications.

Date of Submission **10th-Dec-2020** Internal Faculty

Head of Department

Department Of Computer Science
Rollwala Computer Centre
Gujarat University

MCA – III

Subject: - System Software

Name: - Preksha Sheth

Roll No.: - 36 **Exam Seat No.: -**_____

Sr. No.	Contents	Pg. No	Date	Signature
1.	Assignment – 1		10/12/ 20	
	(Introduction to System Software			
	and Assembler)			
2.	Assignment - 2		10/12/ 20	
	(Linkers and Loaders)			
3.	Assignment – 3		10/12/ 20	
	(Scanner and Parser)			
4.	Assignment – 4		10/12/ 20	
	(Macro Processor)			
5.	Assignment – 5		10/12/ 20	
	(Compilers and Interpreters)			

Date : / /

Assignment - I

Q.1 Define the following instruction.

a) STOP :- It is an imperative statement. It stop the execution of the Program of code is 00.

b) Sub :- It is an imperative statement. Its op code is 02. It substract the value with the register value.
eg :-
MOVER AREG1 = 's'
SUB AREG = '1'

c) Add :- It is an imperative statement. Its op code is 01. It add the value with the register value.

d) MULT :- It is an imperative statement. Its op code is 03. It multiply the value with the register value.
Eg :-
MOVER AREG = 's'
MULT AREG = '1'

e) MOVER :- It is an imperative statement. It's op code is 04. It is used to move the value from memory to the register.

Page No. :
Date : / /

E.g. :- MOVER AREG, X

f) MOVER :- It is the imperative statement. Its op code is 05. It is use to move data value from the register to the memory.

Eg :-

MOVEM AREG, X.

g) COMP :- It is an imperative statement. Its op code is 06. It is use to compare and set condition code.

e.g. :-

comp AREG, X

h) BC :- It is an imperative statement. Its op code is 07. It is used for the branch condition.

e.g. :-

comp AREG, X.

BC LE, AGAIN.

Here, it will check whether, the value of x and value of AREG should be less than or equal to. If condition is successful then the control will shift to the AGAIN label.

Page No.:
Date: / /

(i) READ :- It is an imperative statement. Its op code is 09. It is used to read a value.
Eg:- READ x.
Here value of x will be read.

ii) PRINT :- It is an imperative statement. Its op code is 10. It is used to print the contents of register.
Eg:- PRINT X

k) ORIGIN :- This is assembler directive statement. The this directive instructs the assembler to set the address given by Laddress specification> in the location counter.
Eg:- ORIGIN LOOP+2
Here, through this statement the value of the location counter will be the location value of loop label plus 2. i.e. if loop is at 2m, then the location counter value will be 2014.

l) EQU :- It is Assembler directive statement. In EQU left side is label/symbol and right side is address.

Eg:- <label> EQU <Addr>

Eg:- X EQU Y

Here, X is set to the address of Y.

m) PURGE :- It is an assembler directive statement. It is used undefined the symbol names which are define by the ORIGIN statement.

Eg:- ABC EQU XYZ : ABC represent XYZ

PURGE ABC : ABC NO longer XYZ.

n) ASSUME :- It is an assembler directive statement. It tells the assembler that it can assume the address of the indicated segment to be present in registers.

Syntax :- ASSUME <register> <segment name>

Eg. ASSUME DS : SAMPLEDATA.

Here, the ASSUME directive tells the assembler that the start address of SAMPLEDATA can be assumed to be in the DS register.

Date: / /

c) SEGMENT :- It is the memory used to store three components of a Program i.e. Program code, data & stack. So, the code, stack & Data segment register are used to condition the start address about of these three components. The extra segment register Points to another memory area which can be used to store data.

D) PROC :- It is indicate that the it is a Procedure.
e.g :- CALCULATE PROC.
Here CALCULATE is a Procedeer.

a) NEAR :- It indicate that the whether the call to the procedure it to be assembled as near call. i.e. the Procedure is called from the same segment.
e.g. : CALCULATE PROC FAR.

S) PUBLIC :- When a Symbol is define as a Public in a Program that means it can be referred by the another Program units also.

t) EXTRN :- when an assembly module wishing to use a symbol declear in another assembly module is. The symbol should be declear as Extrn.

syntax :-

EXTRN <symbolic Name : <types>

v) OFFSET :- It is a displacement from the base address of segment.

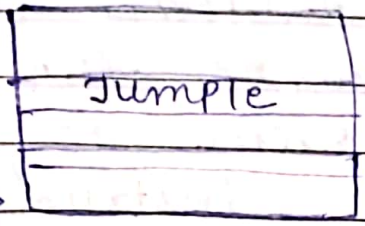
Q.2 Explain the meaning of

b) Base register :-

s The base register is used for Program Relocation.

s It holds the address of data in RAM to be excuted recently.

E.g.:- let say we have memory and in that we have store a Program P1 from 100 to 199



P1 diagram 1

4 So according to diagram 1 their is an instruction Jmp 160. C using absulute address. so now control will be on address 160. Here, all works good.

5 But now this program is send to secondary memory. For some times then another program i.e P2 will occupy address 100 of 199.

4 Then again Program P1 comes back to main memory then new address is allocated.

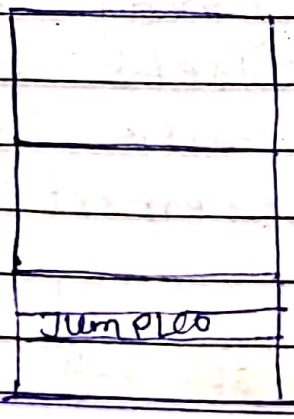


Diagram 2

Page No. :
Date : / /

So, now the Problem will occur as we have change the memory address of program PI but it is having an instruction jump 160. so it create an illegal instruction. so the program PI memory is from 100 to 199 so to solve this Problem we have to add the displacement instead of giving absolute value.

So the effective address
EA = Base register value to displacement.

So their will be no Problem if address is change.

c) Index register :-

The index register have the source index of array and destination Index . i.e. offset . They are provided with auto increment and auto - decrement facility.

E.g :-

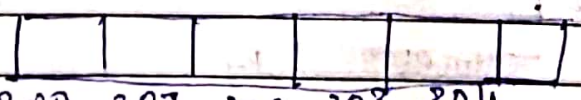


Diagram 1

Here, in diagram 1 the source index is 300. If we went to make at 303

Then,

Effective address = source index offset

= 300 + 3

= 303

This way use can move to 303 address.

d) Symbol table in

Ans An identifier used in the source program is called symbol. Thus names of variables function and procedure are symbols.

Language Processor user the symbol table to maintain the information about the symbol used in source program.

Symbol table performs following things.

i) Add the symbol

ii) Add the location of symbol.

iii) Add the length of symbol

iv) Delete the symbol entry.

(v) Access the symbol entry.
e.g.:-

SYMTAB

Symbol	Address	length
LOOP	202	4
NEXT	214	1

(e) literal table :- The use of literal table is to collect all literals used in a program. At any stage the current literal encountering an LTORG statement literals in the current POOL are allocated address starting with the current value in LC.

E.g. littab

literal	address
=‘5’	211
=‘1’	212

(f) POOL table :-
This table contains the literal number of the starting literal of each literal POOL.

E.g.:- POOLTAB

literal no
#1
#3

g) optab :-

optab contains the fields mnemonic code, opcode and size of the operator. It also contain the class filed which tells us that the statement is imperative Assembler directive or declaration statement.

If an imperative statement the mnemonic info field contains the pair machine opcode, instruction length else it contain the id of a routine to handle the declaration or direction statement.

h) Declaration statment :-

y The syntax of declaration statment is follows :-

[label] DS < constant >

[label] DC < value >

y The DS statement reserves area of memory and assignable names with them.

y The DS statement constructs a memory word containing constants.

e.g. :- A DS 200

ONE- DS '1'

Date : / /

(i) Imperative statement :-

An imperative statement indicates an action to be performed during the execution of the assembled program each imperative statement typically translate into one machine instruction.

E.g. :-

MOVER AREG, X

(ii) Assembler Directives :-

Assembler direction instruct the assembler the Perform certain action during the assembly program.

eg :-

START :n

This directive indicate that first word of machine should be placed in that memory word with address <constant>.

START <constant>

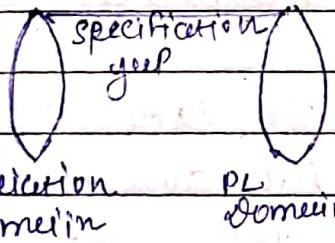
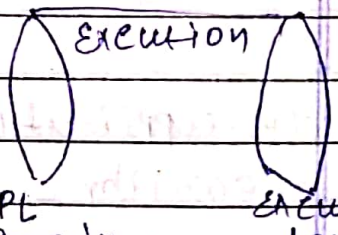
START 500

Here,
first word of the target program is stored from memory location 500 onwards.

Q.3 Diffrent between the following.

	Application Domain	Execution Domain
1)	In this domain designer expression their ideas.	In this the ideal express by designer are implemented
2)	The code of any program is written	on basis of code written output is generated.
3)	In application domain algorithm and syntax is checked	In Execution domain the run time probims are checked.
4)	In application domain algorithm and work of analysis these is done	In execution domain work of the synthesis these is done

Page No. :
Date : / /

↳	e.g:- Printf ("Hello world"); Here we write the command and its syntax is checked	↳ e.g:- Hello world! By Processing the comm output is generated by excutation domain
b)	Specification Gap and Execution gap.	
↳	specification Gap	Excution Gap.
↳	The gap between the specification domain and the Programming domain is called specification gap. 	The gap between the programming domain and Execution domain is called Execution Gap. 
↳	Specification gap is the semantic gap between two specification and same task.	Execution Gap is the gap between the semantic of program coriten in different Programming language

Date: / /

3)	The specifion gap is briged by the sofitwear development team	Excution gap is bridged by designer of Programming language Processor, translator or interpreter.
4	Eg:- suppose a person who dont know programming language can use the application but no able to do Programming.	Eg:- Programmer write Program in high level language and computer does not understand the high level language directly.

c) Source Program and target Program.

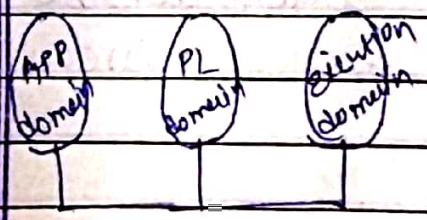
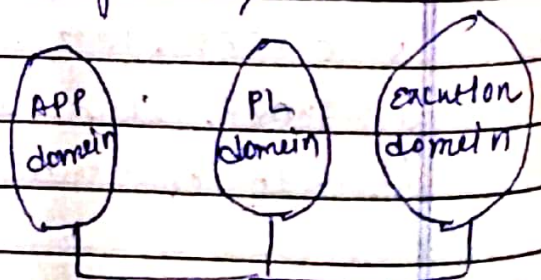
Source Program	Target Program.
A Program written in a Programing language is called source Program	Target program is an output genrated after compling the source Program

	Source Program is written in Programming language by the Programmer	Target Program is compiled and given by the compiler.
2)	Source Program contains english words according Programming language	Target Program only contains machine code.
3)	Programmer can read the source Program.	Machine can read the target program
4)	source program is an input to compiler	Target Program is the out put to compiler.
	Compiler	Interpretor
5.	Compiler converts the entire Program	Interpreter translate code one line at time.
6)	It Produce optimized code.	Code is not too much optimized.
7)	Analyses time is near.	Program analysis time is less.

	machine code is genrated	no machine code is generated.
5	There is an execution gap	there is no execution gap.
6	language translator	Preprocessor
7	Language translate bridge an execution gap to the machine language of the computer system	It is a language Processer which bridges an Execution gap but is not a language translater
8	Assembler is an language translater whose source code is assembler language and convert it to machine lang...	Processor Process highlevel language before language translater takes over.

Date: / /

Q) Problem oriented and Procedure oriented.

Problem oriented	Procedure oriented
Problem oriented language are specific oriented language.	Procedure oriented language are common oriented language
Problem oriented language have large execution gap.	Procedure oriented language have large specification gap.
Execution gap is bridged by translator or interpreter.	specification gap is bridge by an application designer.
Eg:- KOBOL, FOTRAN	Eg:- C, C++
	
Specification gap Execution gap	specification gap Execution gap.

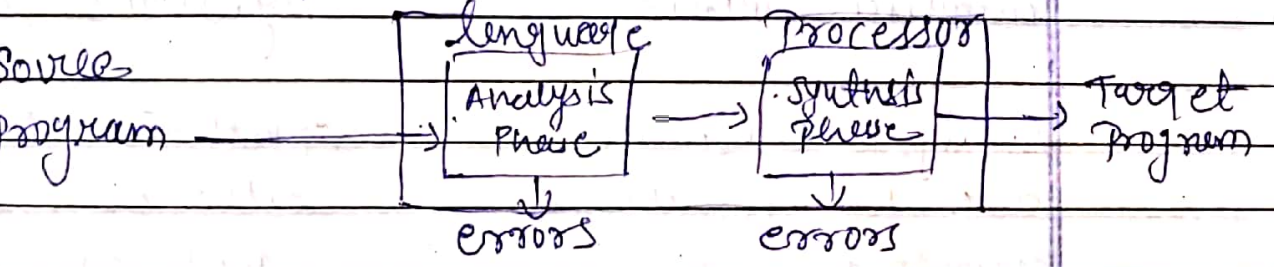
Page No. :
Date : / /

g) Lexical, syntax & semantic rules

Lexical :- It governs lexical rules in the source language eg identifier name is Proper or not.

Syntax :- It checks validity of statement for syntax in the source language.

Symantic Analysis :- It check the semetic rules in the source language.



9 Analysis and syntasis Phase.

Analysis Phase	syntasis Phase
Analysis Phase uses each components of source Program to	syntasis Phase target language statement which have same

determine relevant information out of it	meaning of source language statement
Analysis Phase consists of lexical syntax and semantic analysis	Synthesis phase work 2. Activities (i) creating data structure in target Program (ii) generation of target code.
Intermediate Representation (IR) it the output of analysis Phase	Intermediate representation (IR) is the input for synthesis Phase
Analysis Phase is also known as Pass I	synthesis Phase is also caution as Pass II.

Q.4 Explain the following

b) Difference between single Pass assembler and two Pass assembler.

	Single Pass assembler	Two Pass assembler
1	Scans entire source file only once.	Require two passes to scan source file. First pass :- responsible for label definition and introduce them in syntax table. Second pass :- translate the instruction into assembly language or generate machine code
2	Generally :- i) Deals with syntax. ii) Constructs symbol table. iii) Creates label list iv) Identifies the code segment, data segment stack segment etc.	2 Along with pass i pass ii is also required which - i) Translates actual opcode. ii) Compute actual address of every label iii) Assign code Address for assigning the information iv) Translate operand

Date : / /

		register memory. v) Immediate value is translated to binary string.
4	cannot resolve forward reference of class symbol.	can resolve forward references of data symbol.
5	No object Program is written hence no loader is required	Loader is required as object code is genated.
6	Tends to be faster compared to two pass	Two pass assembler requires recamming. Hence slow compare to one pass assembler
7	only create table with all symbols as address of symbol is calculated.	Address of symbols can be calculated
8	more memory required compare two pass assembler.	less memory required compare to single pass assembler.

Intermediate code not generated	Generation of intermediate code.

4) Explain the significance of location center.

→ To implement memory allocation a data structure called location center (LC) is introduced.

→ The location counter is always made to contain the address of the next memory word in the forget Program.

→ It is initialized to the constant specified in the START statement.

→ Whenever the analysis Phase sees a lable in a assembly statement it enters the lebel and contents of LC in new Entry of the symbol table. Then the value to the location Cunter is increment and it goes to the next statement.

→ It ensure that LC Points to the next memory word in the Forget Program when a machine instruction have different length.

Date: / /

d) To update the contents of le analysis phase needs to know length of different instructions.

E.g. in Symbol table in

Symbol	address
AUTAIN	104
N	113

Here address is inserted by the value stored in the location counter.

e) Differentiate between two variants for generation of intermediate code in two Pass assemblers.

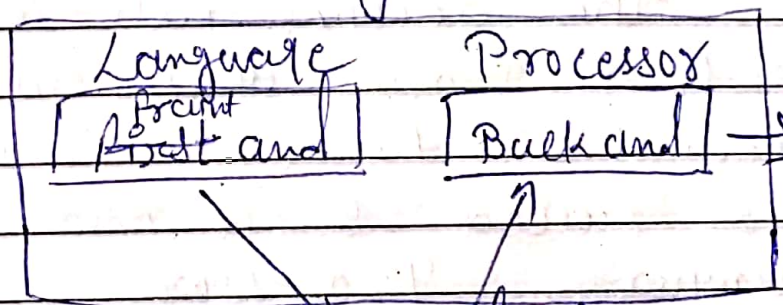
	Variant 1	Variant 2
a)	Is, DL, & AD all statement contain Processed Form.	DL & AD statements contain Processed Form while for IS statement, operand field is Processed only to identify literal references.
b)	Extra work in Pass 1	
c)	simplifies tasks in Pass 2	

Page No.:
Date: / /

Q.4) Explain the following in

(a) What is intermediate Representation in an Assembler?

→ An Intermediate Representation (IR) is a representation of a source program which reflects the effects of some, but not all, analysis and synthesis tasks performed during language processing.



- Two Pass language processor
The first pass performs analysis of the source program and reflects its result in the intermediate representation.

→ The second pass reads and analysis the IR, instead of source

Date :

Program, to perform synthesis of the target program.

- This avoids repeated processing of the source program.

- The first pass is concerned exclusively with source language issues, so it is called front-End of the language Processor.

- The Second pass is concerned with program synthesis for a specific target language so it is called Back-end of the language Processor.

- Properties of an IR :-

Ease of use :- IR should be easy to construct and analysis.

Processing efficiency :- efficient algorithms must exist for constructing and analysing the IR

Page No. :
Date : / /

memory efficiency in IF must be compust.

(vi) give the instruction formut of 8088 microprocessor.

→ The intel 8088 microprocessor supports 8 and 16 bit arithmetic and also provides specical instruction for string manipulation.

→ 8088 microprocessor supports three diffrent instruction formuts.

[a] Register/memory to register.

[opcode d w | mod Reg r/m]

(b) Immediate to Register/memory

→ opcode d w | mod op r/m | data | d

(c) Immediate to Accumulator

| opcode w | data | data :

Date : / /

- The mod and r/m fields specify the first operand which can be in a register or in memory, while the reg field describe the second operand which is always a register.
- The instruction opcode indicates which instruction format is applicable.
- The direction field (d) in the instruction indicates which operand is the destination operand in the instruction.
  if d = 0, The register/memory operand is the destination.
- The width field (w) indicates whether 8 or 16 bit arithmetic is to be used.

Assembly statement	opcod	d	w	mod	reg	r/m	d
ADD AL, BL	000000	0	0	11	011	000	0
ADD AL, 12H[SI]	000000	1	0	01	000	100	0
ADD AX, 3456H	100000	0	1	11	000	000	
ADD AX, 3456H	000001	0	1	01	100	110	

Page No. :
Date : / /

instruction format of microprocessor

Assembly statement	opcode d w	mod reg r/m	Data/ displacem
ADD AL, BL	000000 0 0	11 011 000	00010010
ADD AL, 12H(ST)	000000 1 0	01 000 100	01010110
ADD AX, 3456H	100000 0 1	11 000 000	00110100
ADD AX, 3456H	000001 0 1	01 010 110	00110100

Fig. 4.22 simple instruction of 8086

Adequate, The Third instruction 16 bits of immediate data note that the low byte of immediate data comes first followed by its high byte. The fourth assembly statement is identical to the third. however it has been encoded using the immediate.

(C) Immediate to accumulator

	opcode w	data	data
r/m	mod=00	mod=01	mod=10	mod=11 w=0	w=1
000	(BX)+(SI)	(BX)+(SI)+dx	Note 2	AL	AX
001	(BX)+(DI)	[BX]+(DI]+dx	Note 2	CL	CX
010	(BP)+(SI)	[BP]+(SI)+dx	Note 2	DL	DX
011	(BP)+(DI)	(BP]+(DI]+dx	Note 2	BL	BX
100	(SI)	(SI)+dx	Note 2	AH	SP
101	(DI)	(DI)+dx	Note 2	CH	BP
110	Note 1	(BP)+dx	Note 2	DH	SI
110	(BX]	(BX)+dx	Note 2	BH	DI

Note 1: (BP)+DISP for indirect addressing
d16 for direct.

Note 2: same as Previous column, except d16 instead of dx

Register

reg	8 bit (w=0)	16-Bit (w=1)
000	AL	Ax
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI

Page No. :
Date : / /

III BHL DI

Instruction formats of Intel 8086.

Explain different registers of 8086 architecture.

The intel 8086 microprocessor supports 8 and 16 bit arithmatic and also provides special instruction for string manipulation.

Different register of 8086 architecture
- Data registers AX, BX, CX, and DX.
- Index registers SI and DI.
- stack pointer registers BP and SP
- segment register code, stack, data and Extra.

Data register :-

Each data register is 16 bits in size split into the upper and lower halves either half can be used 8 bit arithmatic, while the two halves together constitute the data register for 16 bit arithmatic

Date :

Index registers :-

The index registers SI also DI are used to index the source and destination addresses in string manipulation instructions. They are provided with the auto-increment and auto-decrement facility.

- Stack Pointer registers :-

Two stack Pointer registers called SP and BP are provided to address the stack.

- SP Points into the stack implicitly also by the architecture to store subroutine and interrupt return addresses.

BP can be used by the Programmer in any desired memory.

Segment registration

- The code, stack and data segment registers are used to contain the start addresses of code, data, and stack segments

Page No. :
Date : / /

The Extra segment register Points to another memory area which can be used to store data.

(i) what is the meaning of segment overiding ? Explain with example.

For arithmatic and mov instruction the arathitecture uses the data segment by defult.

To overide this, an instruction can be Preceded by a 1-byte segment overids PreFix with the following Format :-

001	seg	110

Fix value

where seg, reprented in 2 bits, has the meaning

seg	segment register
00	ES
01	CS
10	SS
11	DS

Page No. :
Date : / /

Example = If the code segment is to be used instead of data segment, it can be rewritten as.

ADD AL, CS : 12H [SI]

The assembler would encode two as. this segment overide instruction.

0011 01 110 0000 10 01 000 000 00000

f) Explain the significance of segment register.

y Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address.

y Need for segmentation :-

(BIU) contains four 16 bits special purpose register called the segment registers.

Page No. :
Date : / /

y Types in

(i) code segment (CS) :- is used for addressing memory location in the code segment of the memory where the executable program is store.

(ii) Extra segment (ES) :- Also refers to a ment in the memory which is another data segment in the memory.

(iii) Data segment (DS) :- Points to the data segment in the memory where the data is stored.

(iv) stack segment (SS) :- It is used for addressing stack segment of the memory the stack segment is that segment of memory which is used to store stack data.

Assignment

Q.2) Define translated link and load time addresses.

a) While Compiling Program P a translater is given an origin specification for P This is called translated origin of P.

b) The Translater uses the value of the transuted origin to Perform memory allocation for the symbols declared in P. This result in the assignment of a translation time address to which each symbol in the Progra

c) Translated Address in The start address pecified by the translater is the translated start address of the Program.

d) Their are A reason for which the linker or loader change the address of Program.

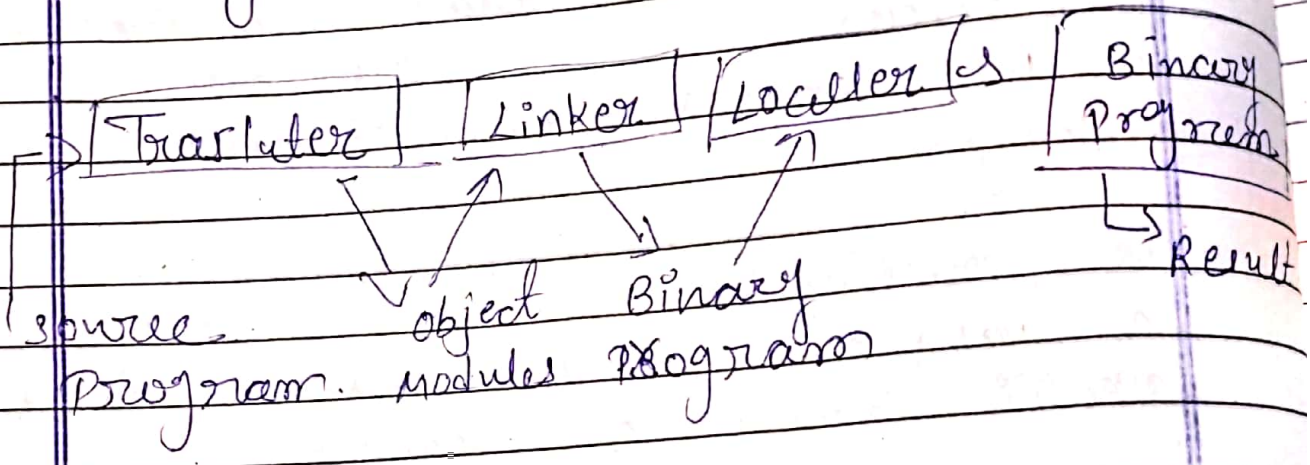
(1) The translated addresses may have been used in different object modules constituting a Program. So memory allocation to such programs would conflict unless their origins are changed.

(2) operating system may required that a program should execute from a specific area of memory.

- Linked address :- Address assigned by the linker
- Local time Address -> Address assigned by the loader.
- Translated origin :- Address of the assigned by the translater. This is the address specified by the programmer in an ORIGIN statement
- Linked origin :- Address of the origin assigned by the linker while producing a binary program.

Page No.
Date : / /

ans Load origin :- Address of the origin assigned by the loader while loading the Program for execution



PROGRAM EXCUTION

Q.3 Define

a) Program Relocation.

ans Address sensitive instruction :- An instruction which uses an Address ie AA.

Address constent :- a cluter word which contains an address ie AA.

ans Program relocation is the process of modifying the address word

Page No.:
Date: / /

The address sensitive instruction of a program such that the program can execute correctly from the region of memory allocation area.

- If linked origin ≠ translated origin relocation must be performed by the linker.
- If load origin ≠ linked origin relocation must be performed by the loader.
- In general a linker always perform relocation.
- If the load origin = linked origin the loader are called absolute loaders.
- Loaders that origin = linked Perform relocation i.e. relocating loaders.

b) Address sensitive instruction or ISR.

- It's a set of instruction or ISR that perform relocation in program P.

Date :

4) steps :-

i) Calculate Relocation Factor (RF)

ii) Add it to translation time address (es) for every instruction which is member of IIR

Eg :- Relocation Factor = 900 - 500 = 400
IIR contains translation address 540 & 558 (instruction : read 17)

Address is changed to 540 + 400 = 940 &
558 + 400 = 838

C) Relocation Factor :-

1) Let the translated & linked origins of program P be t-origin and l origin P respectively.

2) Consider a symbol symb in P (let its translation time address be t symb and link time address symb.
The relation factor is :-

relocation factor P = longing - t - origin

The relocation ceen be Pribive negative or zero.

c) Public definiction & extand Refernees

i) Public definitims :- The ENTRY statement lists the Public definitims of a Program unit i.e it lists thoes symbole defined in the Program unit which may be refernced in other Program units.

ii) Extenal Refernee :- The EXTRN statement Lists the symbol to which Externel refernes are mute in the Program unit i.e the symbol we from the other Program unit.

Q.4) Diffrent b/w non Relocatable relocatable, self relotable module.

1. Non Relocatable inodule :- A Program is one that cannot be executed in any memory area other then the area starting on its translated origin.

E.g:- A hand coded machine language Program.

u Relocatable Module :- A Program is one that can be Processed to relocate it to a desired area of memory.

E.g:- An object Module if can be relocated to the new memory address.

y The difference b/w a relocatable and a non relocatable Program is the availability of information concerning the address sensitive instruction.

y self relocatable Module in A Program is the one can perform the relocation of its own address sensitive instruction.

A self relocatable module can be executed in any area of memory.

E.g:- This is very important in time sharing operating system where local

address of a Program is likely to be different, for different executions.

Q1) Explain data structure used in nested macro calls.

⇒ Two Provision a.rg. required to implement the expension of nested macro calls :-

I.) each macro under expansion must have its own set of data structure i.e

⇒ MEC :- The flow of control during macro experion is implemented using a macro expension counter CMEC.

⇒ APTAB :- This table contain the list actual parameter from the macro

⇒ EVTAB :- This table contein the expans time variable of the macro.

⇒ APTAB ptr :- It points to the index value of APTAB and it is incremental when new parameter is encountered.

Date : / /

4) EVTAB ptr :- It Points to the index value of EVTAB and it is incremented when new parameter is encountered.

5) to extended stack model. A stack is used to accommodate the expansion time data structure.

- The stack consist of expansion records, each expression record accommodating one set of expansion time data structure.
- The expression record of the top of the stack corresponds to the macro call currently being expanded to the main call - when nested macro call is recognized a new expansion record is pushed on the stack to hold the data structure for the call.
- At MEND an expansion record is popped off the stack.

2) An expansion nesting counter (NEST CNTR) is maintained to count the number of nested macro calls.

- Nest cntr is incremented when a macro call is reginzed and decrem

Page No. :
Date : / /

when a MEND statement is encounted Thus nest-enter > 1 indicates that a nested macro call is under expesion. (while nest-entr = 0 implies that macro expansion is not in Progress currently

Q.6

Explain Basic object Module format

Ans The object module of a program contains all information necessary to relocate and link the program with other program

The object module of a program P consists of h components

1.) HEADER :- The Header contains translated origin, size and execution start address of P.

2.) Program :- This component contains the machine language program corresponding to P.

3.) Relocation table :- (RELOCTAB). This table describe IRRP. Each Relotab Entry contains a single Field.

Translated address :- Translated address of an address sensitive instruction.

4.) Linking table (LINKTAB) :- This table

Date : / /

information concering the Public definitions and external refrence in P

Each LINKTAB Entry contains there fields in

symbol :- symbolic name.
TYPE :- PD / EXT indicating whether Public definition or external reference
Translated address :- For a Public definition this is the address of the first memory word allocated to the symbol For an external reference its is the address of the memory word which is required to contain the address of the symbol.

	Eg: in statement	Address	Code
	Start 5000		
	Entry Total		
	Extrn max, Alpha		
	Read A	5002	to 4 0 5640
Loop		5011	
	:		
	Mark AREA, Alpha	5182	to 4 1 000

```
        BC      Any, Mult       519)  to 6    6  000
        :
        BC      LT, LOOP        538)  to 6    1  501
        STOP                    539)  to 0    0  000
A       DS      1               540)
total   DS      1               541)
        END
```

⇒ object module of above Program :-

1) translated origin = 500, size = 42, execution, start address = 500

2) machine language instruction

3) Relocation table

500
538

4) Linking table

Alpha	Ext	518
Mult	Ext	519
A	PD	540

Date: / /

Q] MSDOS Linker.

Explain MSDOS Linker.

Ans: Object Module Format (explain object module of the Program)

An Intel 8086 object module is a sequence of object records, each object record describing specific aspects of the Program in the object module.

There are 14 types of object records containing the following five basic categories of information.

- Basic Binary image (i.e. code generated)
- External references
- Public definitions
- Debugging information
- Miscellaneous information

We only consider the object records corresponding to first three categories a total of eight object record types.

Each object record contains variable length information and may refer to the contents of previous object records.

Each name in an object record

Date : / /

is represented in the following format

length (1 byte) name

THEADR, LNAMES and SEDEF records

THEDR Record

80H length T-Module name checksum

- The module name in the THEADR record is typically derived by the translator from the source file name.
- This name is used by the linker to report errors.
- An assembly Programmer can specify the module name the NAME directive

→ LNAMES record:

96H length name-list checksum

SEGDEF record.

98H length attributes segment name segment length (2) index (1)

A SEGDEF record designates a segment name using an index into this list.

The attributes filed of SEGDEF record indicates whether the segment is relocatable or absolute, whether (and in what manner) it can be combined with other segment & also the alignment requirment of its base address (e.g. byte, word or paragraph i.e. 16 byte, alignment)

steeld segments with the same name are concatenated with each other, while common segment withe same name are overlapped one other.

The attribute filed also contains the origin specification for an absolute segment.

→ EXTDEF and PUBDEF record:

8CH	length	External reference list	check-sum
90H	length base	name offset (2-4)	check-sum

The EXTDEF record contains a list of External used by the Programs of this module.

Date : / /

- A FIXUPP record designates an External Symbol name by using an index into this list.
- A PUBDEF record containing a list of Public declared in a segment of the this list.
- The base specification identifies the segment.
- Each (name, offset) Pair in the record define one public name, specifying the name of the symbol and it's offset within the segment designated by the base specification.

→ LEDATA Records

ADH	length	segment Index (1-2)	data offset (2)	data	check-sum

- An LEDATA record contains the binary image of the code generated by the language translator.
- segment index identifies the segment to which the code generated by the language translator

→ FIXUPP record

GCH	length	locat (1)	fix data (1)	frame datum (1)	target datum (1)	target offset (2)	... check sum

- A FIXUPP record contains the binary information for the one or more relocation and linking fixup to be performed.
- The locat field contains a numeric code called loc code to indicate the type of a fixup.
- The meaning of these codes are given in table.

y MODEND records in

8AH	length	type (1)	start addr (5)	check sum

y The MODEND record signifies the end of the modules with the types field indicating whether it is the main program.

Date : / /

Q.9

Sr no	statement		offset
001	NAME	DEMO	
002	TEST	SEGMENT	
003		EXTRN X, Y, Z	
004		Public A, B, C	
007	A	---	0010
!		!	
012		MOV AX, SEGT X	0025
!		!	
022		MOV AX, offset Y	00 ED
!		!	
030		MOV AX, SEGT Z	00 F5
!			
035	B	---	0080
!			
040	C	---	008F
!			
045		DDB 25	01A9
046	TEST	ENDS	
		END	

object Modules DEMO :

Type	Length	other Fields		checksum	
80H	---	04	DEMO	---	THEADR
96H	---	04	TEST	---	LNAME
98H	---	60H	120 01	---	SEGDEF

Type	Length	other fields	checksum
90H	---	01 01 A 0010	--- PUBDEF
90H	---	01 01 B 0020	--- PUBDEF
90H	---	01 01 C 0025	--- PUBDEF
90H	---	01 X 01 Y 012	--- EXTDEF
8CH	---	01 0025 01 0000	--- LEDATA
ADH	---	8801 06 01 01	--- FIXUPP
9CH	---	01 0050 A1 00 00	--- LEDATA
ADH	---	8401 06 01 02	--- FIXUPP
9CH	---	01 0074 A1 0000	--- FIXUPP
ADH	---	8801 06 01 03	--- FIXUPP
90H	---	80H	--- LEDATA
80H	---	00H	---

7) FIXUPP codes and fix data field codes

The locat field contains a numbers code could for code for to indicate the type of a fixup.

Loc code	Meaning
0	Low order byte fix
1	offset fix
2	segment fix
3	Pointer fix

FIXUPP codes

5) Local also contains the offset of the FIXUP location in the previous LEDATA record.

6) The Fix data field indicates the manner in which the target data and target offset fields are to be interpreted.

Code	contest of target datum & offset fields
0	segment index and displacement
2	External index and target displacement
4	segment index (offset field not used)
6	External index (offset not used)

(6) Take a assembly language Program and Explain the execution of- MSDOS linker.

5) Linker Performs relocation and linking of all named object modules to produce a binary Program with the specified load origin.

Page No. :
Date : / /

- Linker execution terminates when all external reference have been resolved or when the unresolved external references cannot be found in any of the library files.
- Linker uses a two-pass strategy.
- In the first pass the object modules are processed to collect information concerning segment and Public definition.
- The second pass perform relocation and linking.
- First pass :- In the first pass LINKER only processes the object records for bridge NTAB,

second pass :- In the second pass LINKER builds NAMEDEF, SEGDEF, and EXTDEF

Sr no	statement		offset
001		NAME first	
002	Test 1	SEGMENT	
003		EXTRN A: BYTE, Y: WORD	
004		PUBLIC A	
007	A	---	015
012		MOV AX, SEG X	021
022		MOV AX, OFFSET Y	056
035	TEST 1	ENDS	
036		END.	

001		Name second	
002	TEST 2	END. SEGMENT	
003		EXTRN Z: BYTE	
004		PUBLIC X	
010	X	---	033
015		MOV AX, SEG Z	1FF
069	TEST 2	ENDS	
		END	

001		NAME THIRD.	
002	TEST 3	SEGMENT	
003		EXTRN A: WORD	
004		PUBLIC Y, Z,	
015	Y	---	043
033	Z	---	047

Page No. :
Date : / /

037		MOV AX, SEGT A	274
059	TEST 3	ENDS	
		END.	

a) after Prossing FIRST

NTAB

symbol	load address
TEST I	10000
A	10015

NAMFLIST

TEST I.

SGTTAB

segment	load address
TEST I	1000

EXTTAB

External symbol	load address
X	10153
Y	10219

b) after Prossing SECOND

NTAB

symbol	load address
TEST 2	10120
X	10153

SEGTAB	segment TEST 2	Load address 10120

EXTTAB	External Z	load address 10266

c) after passing THIRD

NTAB	symbol TEST 3 Y Z	load address 10176 10219 10266

NAMELIST	TEST 3

SEGDEF	segment name TEST 3	load address 10176

EXTAB	External symbol A	load address 10015

Page No.:
Date: / /

SS Assignment :

1) Different between scanning and parsing.

* Scanning.

a) Scanning is the process of recognizing the lexical components in a source string.

b) The lexical features of a language can be specified using Type-3 regular grammars. This facilitate automatic construction of efficient recognizers for the lexical features of the language

c) Finite state automatic (FSA): A finite state automation is a triple (S, F, T), where.
S - is a finite set of state one of which is the initial state and one or more of which are to final state.

Σ - is the alphabet of source symbol

T - is a finit set of state reations out of each $s_i \in S$, on encountering the symbol of t.

- Deterministic finit state Automation (DFA) = It is an FSA such that $t_i \in T, T_i (s_i, symb, s_j)$ implies $A, t_i \in T, t_i = (s_i, symb, s_k)$. in other words It is a finit state automatic name of whose states has two on more transitions for the same source symbol The DFA has the property that it reaches a unique state for every source string input to it.

- Regules Expression :- A regular expression is a sequence of chatercters that define a search pattern, mainly for use in pattern matching is ried and replace " = like, operations = example -r, s, r, s, etc.

* Parcing

Parsing is used to check the validity of a source string and to

determine its syntatic structure.

4) For an invalid string the parser issues diagnostic messages reporting the cause and nature of errors in the string for valid string it builds a parse tree to reflect the sequence of derivations or reductions performed during parsing.

5) Two fundamental approaches to parsing is.

- TOP-down parsing :- It is the parser which operates according to a grammar which attempts to derive a string matching a source string through a sequence of derivations & future it is classified into -

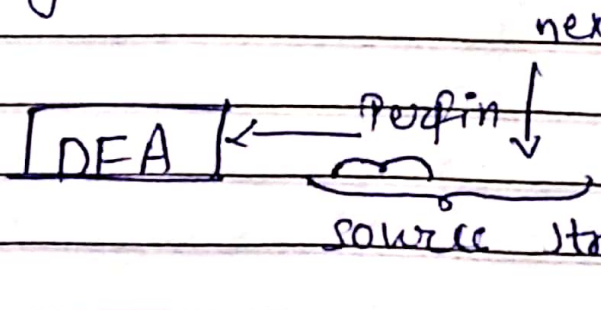
(i) TOP-down Parsing with metracking
(ii) TOP-down Parsing without backtracking
(iii) LL(I_2) parser.
(iv) Recursive descent parser

Bottom-up parsing: A bottom up parser constructs a parse tree for a source string through a sequence of reductions. It proceeds in a left-to-right manner. Further it is classified into

(i) operator precedence parser
(ii) LR parser.

Q.2 Define and give the usage of DFA

A It is a ~~def~~ finite state automaton none of whose states has two or more transitions for the same source symbol. The DFA has the property that reaches a unique state for every source string input to it.



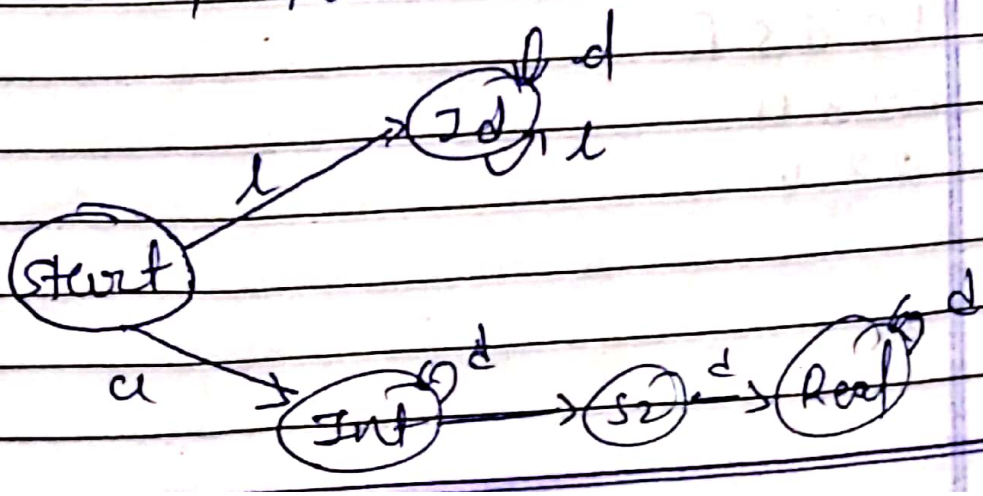
A At any point, the DFA recognized some prefix of the source string possibly the null string and would be poised to recognize the symbol

Pointed by the Pointed not symbol.

- The validity of a string is determined by going it at the input by of a DFA in its intial state.
- The string is valid if and only if the DFA recognized every symbol in the string and find itself in a final state at the end of string.

Example of Building DFAs

State	Next symbol		
	l	d	
State	Id	Int	
Id	Id	Id	
Int		Int	S1
S_2		Real	
Real		Real	



Date : / /

A comined DFA for integres real number & identifies.

→ Here, in this DFA is useing 3 final state - Id, int & Real corresponding to identities, unsigned integer and unsigned real respectivley, string like '25' in invalid because it leaves the DFA in state s_2 which is not a final state.

→ correct input would be like '1234-45' or 'asd12' etc

Q.3) Take Example of any three valied string and Content DFA.

→ 3 valid string.

12d.5F

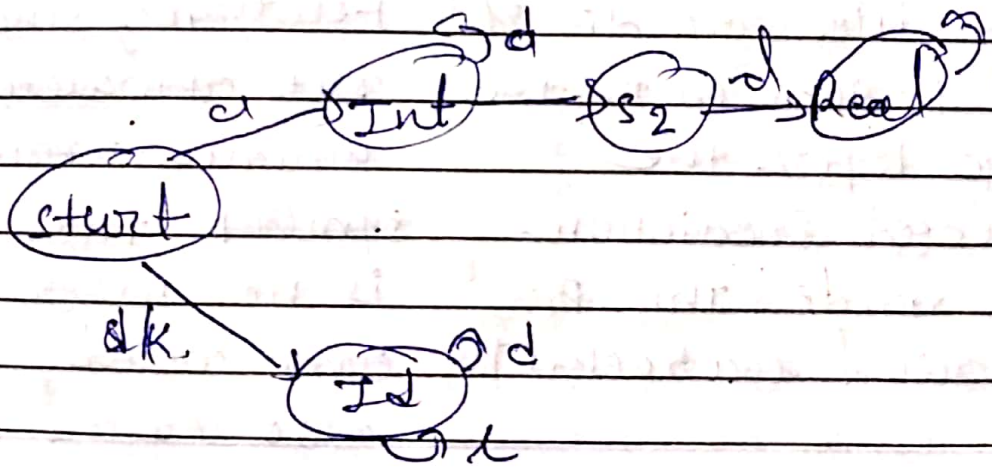
.234

23.4

Date: / /

start	Next symbol		
.	.d	l	-
start	Int	Id	S_2
int	Int		S_2
Id	Id	Int	S_2
S_2	Real		
Real	Real		

* Graphical Representation.



Q.4) For Above same Example constant register Expression also.

Q.5) Different between ~~two~~ t. TOP-down Parsing with backtracking and without backtracking.

Date: / /

	Top-down parsing with Backtracking	Top-down parsing without Backtracking
1.	The leftmost nonterminal in CSF is identified i.e. E, so, then, it matches the input string its assigned grammar. If the check fails the predicities are discarded and it goes on to the step before the rejected prediction was made. This proud is called backtracking.	Here, excluded the use of Backtracking which makes the Prediction very exercise. The grammer, have to be modified so that it does not head left factoring since its ruts alternatives produce unique Terminal symbol thus, Parsing is no longer by trial and error. It is also called predictive Parsers.
2)	The backtracking results in problems like. (1) semantic actions cannot be performed while making a perdiction. (2) Percise error reporting is not possible	when removed backtracking it resulted into several advantages like. (1) Parsing become more efficient and also be possible to perform semantic actions.

Page No.:
Date: / /

③ also it consumes time as it goes to pervious stage for serching;	② Precise error reporting during Parsing. ③ It does not consume time compared to top-down parsing with Backtracking
③ Grammer used Here, is, $E \to T + E / T$ $T : V * T / V$ $V := \langle id \rangle$	Grammer used Here is, $E := TE''$ $E'' := +E / \varepsilon$ $T := VT'$ $T' := *T / \varepsilon$ $V := \langle id \rangle$

Date: / /

ii) Diffrent between top-down passing and bottom up passing.

	Top-down passing	Bottom up passing
1)	It is a passing strategy that first looks at the lighter liked of Posse that tree 'who' which down this pases tree by using the rules of grammer.	It is a passing strategy that first lots of the lowest lived of the pase tree and wirts up the Prese tree by using the rules of gramma
2)	Top-down passing: attempts to find the left most clerition for an input storing.	Bottom up passing can be define as an attempt to reduce this input string to starts symbol of grammer.
3.	In this passing technique, we stents passing from top (start symbol of pases tree) to down (the lodd node apases tree) in to a drawn manner.	In the passing technique we stert pasing from bottom (and node of passes trees ho up (this start symbol of passes tree) in bottom up manner.

Date : / /

4.	This passing technique using left most derivation.	This passing technique uses Right most derivation.
5.	It's main decision is to select which production rule to use in order to construct the string	It's main decision is to select when to use a production rule to reduce the string to get the starting symbol.

Q. Define

(a) Grammar :-

→ The lexical and syntactic feature of a programming language are specified by its grammar. A grammar is a set of rules which precisely specify the sentences of a language I.

Example :-

< Noun phrase > → < Article > < Noun >

< Article > a | an | the

< Noun > boy | apple

(b) Terminal symbol

→ A symbol in the alphbet is known as afernineal symbol.

(c) Non - Terminal symbol.

→ A non - Terminal symbol is the name of a symbon category of a language, p.g. now, new. etc. An non - terminal symbol is written as a single capital letter, or as, name enclosed between <---->. eg. A or (NOUN)

→ During grammetic analysis, a non terminal symbol no present on instance of the category. Thus <NOUN> represites a now.

(d) Production rule :-

→ A moduction also called a rewriting rule, is a rule of grammar, It has the from A non-terminal symbol → string of terminal and non-terminal symbol.

where the notation '→' stands for 'is defined as' example < Noun phrase > → < Article > < Noun >

(c) Reduction Rule :

→ Let Production : P_j of grammar of be of the form

$P_j : A ::= \alpha$

and let σ be a string such that $\sigma = \gamma \alpha \theta$, then replacement of α by in string σ constitutes a reduction according to Production P_j.

For Example :-

Grammar : < Noun phase > → < Article > < Nouns >

< Article > → a | an | the

< Nouns > → boy | apple

string = the boy

Step	string
0	the boy
1	< Article > boy
2	< Article > < Nouns >
3	< Noun phrase >

Date: / /

(5) Derivation Rule :-

→ Let Production P_1 of Grammer G, be of the form

$P_1 : A ::= \alpha$

and let B be a string such that $B \equiv \delta A \theta$,

then replacement of A by α in string B constitutes a derivation according to Production P_1.

Example :-

Grammar :- <Noun phrase> → <Article> <Noun>

<Article> → a | an | the

<Noun> → boy | apple.

String : the boy.

derivation

<Noun phrase> = <Article> <Noun>

⇒ <the> <Noun>

⇒ the boy.

(d) Abstract syntax Tree (AST)

- A Parse tree depicts the steps in parsing, hence it is useful for understanding the process of parsing. However, it contains too much information. So, an abstract syntax tree (AST) represents the structure of a source study in a more economical manner.

- An AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code.

- The word 'abstract' implies that it is a representation designed by a compiler designer for his own purpose. Thus, the designer has total control over the information represented in an AST.

(h) Left-to-left Parsing :

- TOP down Parsing traversing a

attempts to derive a string meeting a source string through a sequence of derivations starting with the existing wished symbol of CT. For a valid source string α, a top down parse thus determineds a derivation sequence:

$$S \Rightarrow \cdots \Rightarrow \cdots \Rightarrow \alpha$$

Since, we make the derivation for the left-to-left Passing (LL Passing

(i) Left-to-right Passing :-

A bottom up Passer constructs a Pases tree for a source string throught a sequnce of reductions the sources string is valid if it can be reduced so, h, the existing wished symbol of CT it, not, an error is to be detected, Bottom up Passing Proceds in a left-to-right, manner i.e in attempts, at reduction. start with the right thumee, bottom up - Passing is also known as left-to-right Passing.

(j) SSM :-

Ans SSM stands for source string marker. SSM points the the first unmatched symbol is the source string.

(k) Prodition :-

Ans when we select on alternative on the RHS of the production for the leftmost terminal in csf, since, we do not know wheter the derived string with satisfy the crittication check, we call this choice of prediction.

Q.8] Explain the following parsing with grammer algorithm and example

(a) Topdown Parser with Backtracking:

Grammar :- E := T + E / T

T := V * T / V

V := <id>

Source string → a + b + c to <id> + <id> * <id>

The Prediction making mechanism selects the RHS alternative of a Production in a left-to-right manner.

1.	E	(starts with E)
2.	T + E	(APPlied rule 1 and replaced E → T + E)
3.	V * T + E	(APPlied rule 2 and replace → V * T)
4.	<id> * T + E	(APPlied rule 3 and replace V → <id> but it does not match with our and vasult use next <id>+ and not <id> so match go to step 3 and replace T → V.
3.	V + E	(APPlied rule 2 and replaced T → V)
4.	<id> + E	(APPlied rule 3 and replace V → <id>)
5.	<id> + T + E	(APPlied rule 1 & replaced E → T + E)
6.	<id> + V * T + E	(APPlied rule 2 & replaced T → V * T)
7)	<id> + <id> * T + E	(APPlied rule 3 & replace V → <id>)

Page No.:
Date: / /

8. <id> + <id> * V * T * E (Applied rule 3 & replaced T → V * T)

9. <id> + <id> * <id> * T + E (Applied rule 3 & 2 replaced V → <ID> but it does not match so we have to backtrack and go back to step 5 and replace E by F).

5) <id> + T (Applied rule 1 & replaced E → T)

6) <id> + V * T (Applied rule 2 & replaced T → V * T)

7) <id> + <id> * T (Applied rule 3 & replaced V → <id>)

8) <id> + <id> * V * T (Applied rule 2 & replaced T → V * T)

9) <id> + <id> * <id> * T (Applied rule 3 & replaced V → <id> but it does not match so we have to go back to step 8 and replaced T → V).

8. <id> + <id> * V (Applied rule 2 & replaced E → V)

9) <id> + <id> * <id> (Applied rule 3 & replaced V → <ID>)

Page No. :
Date : / /

So at the end we got the string matching the source string.

(b.) Top-down parser without Backtracking.

Grammar :- $E :- TE''$

$E'' :- +E / \varepsilon$

$T :- VT$

$T^* :- *T / \varepsilon$

$V :- <id>$

Source string $\Rightarrow a * b + c$ (id)

$<id> + <id> + <id>$

1.	E	(Starts with E)
2.	TE	(Applied rule 1 & replaced $E \rightarrow TE''$)
3.	$VT''E''$	(Applied rule 3 & replaced $T \rightarrow VT''$)
4.	$<id>T''E''$	(Applied rule 5 & replaced $V \rightarrow <id>$)
5.	$<id> * TE''$	(Applied rule 4 & replaced $T'' \rightarrow *T$)
6.	$<id> * VT'E''$	(Applied rule 3 & replaced $T \rightarrow VT''$)

Page No. :
Date : / /

7) <id> * <id> T' E' (Applied rule 5 & replaced V → <id>)

8. <id> + <id> ∈ E (Applied rule 4 & replaced T with ε (epsilon) because we wanted + and not so it was the best prediction available)

9) <id> * <id> ∈ E'' (Applied rule 4 & replaced E'' → + E)

10) <id> * <id> + ∈ E' (Applied rule 1 & replaced E → T E')

11) <id> * <id> + VT'' E'' (Applied rule 3 & replaced T → VT'')

12) <id> * <id> + <id> T'' E'' (Applied rule 5 & replaced V → <id>)

13. <id> * <id> + <id> ε ε (Applied rule 4 & 5 and replaced T'' → ε, E' → ε because we have found the matching string.

14) <id> * <id> + <id>

Date: / /

(Q) LL1 Parser.

(A) An LL(1) parser is a table driven parser for left-to-right parsing the '1' in LL(1) indicates that the grammar uses a look ahead of one source symbol - that is, the prediction to be made is determined by the next source symbol.

(A) Example :-

$E ::= TE'$

$E' ::= +TE' / \varepsilon$

$T ::= VT'$

$T' ::= *VT' / \varepsilon$

$V ::= \langle id \rangle$

Source string :-

$\langle id \rangle + \langle id \rangle * \langle id \rangle - 1$

Parser Table

NON Terminal	source symbol			
	$\langle id \rangle$	+	*	-1
E	$E \Rightarrow TE'$			
E'		$E' \Rightarrow +TE'$		$E' \Rightarrow \varepsilon$
T	$T \Rightarrow VT'$			
T'		$T \Rightarrow \varepsilon$	$T \Rightarrow *VT'$	$T \Rightarrow \varepsilon$
V	$V \Rightarrow \langle id \rangle$			

Date: / /

LL Parsing

current sentential form	Symbol	Tradition
⊢ E ⊣	<id>	E => TE'
⊢ TE' ⊣	<id>	T => VT'
⊢ VT' E' ⊣	<id>	V => <id>
⊢ <id> T' E' ⊣	*	T' => * VT'
⊢ <id> T' E' ⊣	*	V => <id>
⊢ <id> * VT' E' ⊣	<id>	V => <id>
⊢ <id> * <id> T' E' ⊣	+	T' => E
⊢ <id> * <id> E' ⊣	+	E' => +TE'
⊢ <id> * <id> + TE' ⊣	<id>	T => VT'
⊢ <id> * <id> + VT' E' ⊣	<id>	V => <id>
⊢ <id> * <id> + <id> T' E' ⊣	+	T' => E
⊢ <id> * <id> + <id> E' ⊣	+	E' => E
⊢ <id> * <id> + <id> ⊣	⊣	⊣

Q. Recursive Descent parser in

A recursive descent parser is a variant of top down parsing without backtracking.

If we a set of recursive procedures to perform parsing.

To implement recursive descent parsing a left factored grammar is modified to remove left recursion

Date : / /

Occurrences of strings more explicit.

- Salient advantage of recursive descent parsing are its simplicity and generality.
- It can be implemented in any language supporting recursive procedures.

Example :-

Recursive descent parser

```
Procedure proc-G : (tree-root)

var
  a,b : pointer to a tree node.
begin
(1)  proc-T (a)
(2)  while (nextsymb) = '+') do
(3)  { match ('+');
       proc-T (b);
       a := tree build ('+', a, b);

(4)  tree-root := a;
     return,
     End proc-E;
```

Date : / /

```
Procedure proc-T (tree-root)
  var
    a,b : pointer to a tree node;
  begin
(5)  proc-V ('a')
(6)  while (next-symb = 'k') do;
(7)  { match ('*');
       proc-V (b);
       a := treebuild ('*', a, b);

(8)  tree-root := a;
     return;
     end proc-T;

(9)  { Procedure proc-V (tree-root)
       var
         a : pointed to a tree-node;
       begin
         (next-sym = <id>) then
       tree-root := treebuild <id>, --)
       else Print " Error ";
       return;
       End proc-V;
```

String to be Persed → p+q*r.

Page No. :
Date : / /

Call Proc E

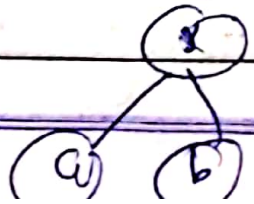
step - 3 Call Proc_T (a).
in that call Proc V(a)
in that next symbol is id
so build a tree node (8)

step 4 :- return to step 5
next symbol is not '*'

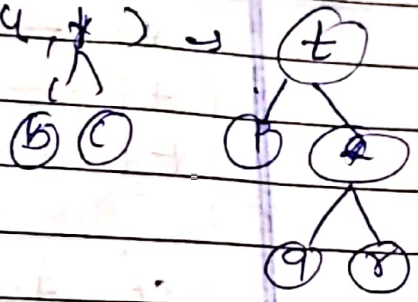
step 5 :- so ~~here~~ return to step 5
next symbol is '+'
so return ('+')
call proc_T (b):
Call Proc_V (b)
next symbol is id
so build a tree node (9)

step 5 :- return to step 5
then Next symbol is '*'
so match '*'
and call Proc V (c)
& create tree node & (7)

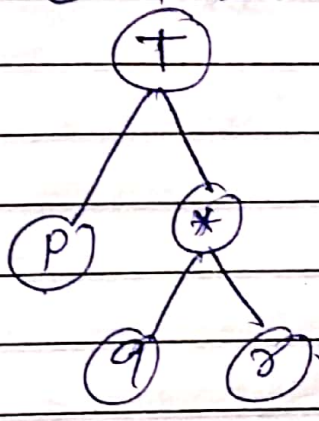
step (1) return to step (7)
and build the tree (*, a, b)



step-③ Now, return, to step 3
and treebuild ('+', q, r) →



so, the final tree is



⑥ operator precedence parser.

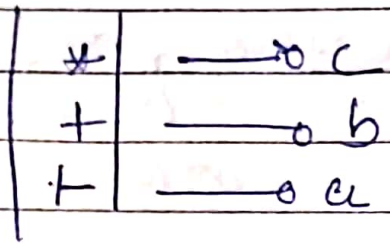
Precedence between operators a & b appearing in a sentential form $Pabq$, where P is a null string is termed a operator precedence.

Example: The parsing string is
a + b * c

Date: / /

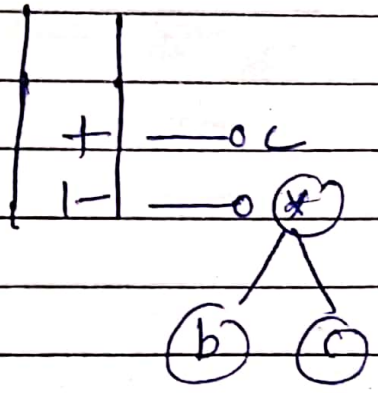
stack

1)



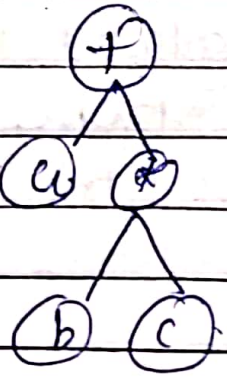
so the operator persistence of is higher so it we pop out

2.



Last remining operator will pop at

3)



Page No. :
Date : / /

* Macros and Macro Prototype *

Q.1 Macro Prototype :-

- A Macro Prototype statement declares the name of a macro and the names and kinds of its parameters.
- The Macro prototype statment has the full syntax :-

< macro name > [< Format Parameter spec > [, ...]]

- Where < macro name > appears in the mnemonic field of an assembly statement and < Formal parameter spec > is of the form.

& < Parameter name > [< Parameter Kind >

Q.2 Macro Definition :-

- A Macro Definition is enclosed between a macro header statement and a macro end statement.
- Macro definitions are typically located

at the start of a Program. A Macro definitation consists of

1. Macro Prototypes statement
2. one or model statement.
3. Macro Preprocessor statement

→ The Macro Prototype statement declare the name of a assembly language statement may be generated during macro expansion.

→ A Preprocessor statement is used to Perform quility function during Macro expansion.

Exa:in

```
MACRO
INCR  &MEM_VAL, &INC_VAL, &REG
MOVER 2REG    : &MEM_VAL
ADD   &REG    ,2 INC_VAL
MOVEM &REG    ,&MEM_VAL
MEND
```

Q.3 Model statements :-

→ A model statement is a statement

from which assembly language statement may be generated during Macro Expansion.

3) The statements between MACRO and MEND directives define the model statements of the macro and can appear in the Expanded code.

Q.4 Preprocessor Statment :

A Preprocessor statement is used to perform auxiliary functions during Macro expansion ~~AIF~~ and AGO are preprocessor statement.

Q.5 Positional Parameters

3) For positional formal parameters, the specification & Parameter kinds of synex rule is simply committed.

3) Thus, a positional formal parameter is written as & <parameter name> ie, & SAMPLE, where SAMPLE, is the name of a parameter.

Date : / /

3) The call on a macro using positional Parameters in an ordinary string.

3) The value of a positional formula parameter xyz is determined by the rule of positional association as follows:

* Find the ordinal position of xyz in the list of formal parameters in the macro prototype statement.

* Find the actual parameter specification that occupies the same ordinal position in the list of actual parameters in the macro call statement. If it is an ordinary string ABC, the value of formal parameter XYZ would be ABC.

For Ex.

```
MACRO
N1  &P1, &P2, &P3
...
...
MEND
```

Page No. :
Date : / /

Here, P1, P2, and P3, are positional parameters.

Q.6 Keyword Parameters

1) Keyword parameters are symbolic parameters that can be specified in any order when the macro is called.

2) The parameter will be replaced within the macro body by the value specified when the macro is called these parameters can be given a default value. If no default value is specified and if the parameters is not given a value when the macro is called, then the parameters will be replaced by a null string.

3) Each keyword parameter will have an equal sing (=) as the last character of the parameter name.

For example:

```
MACRO
M1    &P1 = , &P2 = , &P3 =
---
---
MEND
```

Here, &P1 =, &P2, & &P3 = are keyword parameter.

Q.7 Defult parameters:

- A defult is a standard assumption in the absence of an explicit specification by the programmer.
- when the desired value is different from the defult value, the desired value can be specified explicity in a macro call, This specification overside the defult value of the parameter for the duration of the call.
- If a parameter has the same value in most calls on a macro, this value can be specified as its defults value in the macro definition.
- Defult values of keyword parameter can be specified by extending the syntax of formal parameter specification, as follows:-

& < parameter name > [< parameter kind >
[< default value >]

for exe.
MACRO
M1 &P1 = A, &P2 = B.

MEND
Here, &P1 = A and &P2 = B are default parameters.

Q.8 Give two Examples of nested macro calls.

i) A macro statement in a macro may constitute a call on another macro such calls are known as nested macro calls.

ii) The macro containing the nested is called as the outer macro.

iii) The macro which is called in the outer macro is called as a inner macro.

Expansion of nested macro calls follows the last-in-first-out (LIFO) rule.

Ex:

1) MACRO
MAC X &PAR1, &PAR2
MOVER REG1, &PAR1
MACRO // Nested macro
MAC = X, &PAR2, REG1 - REG3
ADD REG1, &PAR2
MOVEM REG1, &PAR2
MEND
PRINT &PAR1
MEND

2) MACRO
MAC Y &PAR2, REG1, REG3
ADD REG1, &PAR2
MOVEM REG1, &PAR2
MEND
MACRO
MAC-X &PAR1, &PAR2
MOVER
MAC Y REG1, PAR2
PRINT &PAR2, REG1 - REG3, // Call MAC
MEND &PAR1

Page No.:
Date: / /

Q. Expression Time variable

i) Expression time variables (EV's) are variable which can only be used during the expansion of macro calls.

ii) A local E.V. is created for use only during a paramicular macro calls. A global E.V. exists across all macro calls situated in a program and can be used in any macro which has a declaration for it.

iii) Local and global EV's are created through declaration statement with the following syntax:

LCL < E.V. specification > [, < EV specification > ...]

GBL < G.V. specification > [... < EV. specification > ...]

and < UP Pu. specification > has the syntax P < EV name where < EV name > is an ordinary string

Page No. :
Date : / /

→ values of EV's can be manipulated through the preprocessor statement SET. A SET statement is written as

<PV specification> SET <SET. expression>

→ where <PV specification> appears in the label field and SET in the mnemonic field A SET statement assign the value of <SET. expression> to the EV specified

Example :-

```
        MACRO
        CONSTANTS
        LCL     &A
&A      SET     I
        DB      &A
&A      SET     &A+I
        DB      &A
        MEND
```

Q.10 Parameter Attributes :-

→ An attribute is written with the syntax <attribute name> <

parameter spec>

c) and represents information about the value of the format parameter i.e. about the corresponding actual parameter. The type length, and size attributes have the names T, L, and S.

Example:

```
        NAME
        CONSTANTS
&A      LCL     &A
        SET     I
        DB      &A
&A      SET     &A+I
        DB      &A
        MEND
```

Q.19] AIF-

An AIF statement has the syntax

AIF (&Expression) &Sequencing-symbo

where &Expression is a relational Expression involving ordinary strings

Page No. :
Date : / /

Parameter and their attributes, and expression time variable.

If the relational expression evaluates to be true, expression time control is transferred to the statement containing <sequencing symbol> in its label field.

Q.12 AUTO

An - AUTO statement has the syntax :-

AUTO <sequencing symbol>

It unconditionally transfer expression time control to the statement containing <sequencing symbol> in its labels fileds.

Example : MACRO

PVAL &X, &X, &2

AIF (CPY, EQ, &X), ONLY

MOVER ARG1, &X

SUB ARG2, &Y

ADD ARG3, &Z

MOVEM ARG1, &X

Page No.:
Date: / /

AUTO , OVER
ONLY MOVER ARG1, &2
MOVER ARG1, &X
ONLY MEND

Q.13 ANOP

→ An ANOP statement has the syntax
< sequencing symbol > ANOP.

→ An significance of ANOP statement is to define sequencing symbol.

Q.14 REPT

→ REPT statement has the syntax:

REPT < expression >

→ < expression > should evaluate in to a numerical value during macro expresion. the statement between REPT, and an ENDM statement would be processed for Expression < expression > number of times.

Following example illustrates the use of this facility to declared 10 constants with the values 1, 2 ... 10

```
Exa.        MACRO
            CONSTIO
            LCL     &M
&M          SET     1
            REPT    10
            DC      '&N'
&M          SETA    &M+1
            ENDM
            MEND
```

Q.15) IRP

The IRP statement

IRP <formal parameter>, <argument-list>

the formal parameter mentioned in the statements takes successive value from the argument list. for each value the statement between the IRP and ENDM statement are expanded once.

example:- MACRO

CONSTA1 &M, &N, &P

Page No. :
Date : / /

IRP
DC
ENDH
NEND

P2, 2H, F, 8N
'82'

y A MACRO call CONSTS 4, 10, leads to declarations of 3 constants with the value 4, F, and F.

Q.16. Show combined working of MACRO and assemble.

y A macro processor is functionally independent of the assembler and the output of the macro processor will be part of the input into the assembler

y A macro expresser similar any other assembler scans, and process statements

y Often the use of a seperate macro processor for sending macro instructions leads to less effient Program translation because many

Date : / /

function are duplicated by the assembler and macro processor.

To overcome efficiency issues and avoid duplicate works by the assembler the macro processor is generally implemented within passes of the assember.

The intergration of macro processor and assembler is often referred as macro assembler. such implementions will help in elimination the overheads of creating intermediate files.

thus improving the performance of integrations by combining similar functions

To design the pass structure of a macro-assembler we identify the function of a macro preprocessor and be merged to advantage. After merging the function assemble this process leads to the following pass structures:-

Page No. :
Date : / /

Pass I.
1. Macro definition processing
2. SYNTAB construction

Pass 2:
1. MACRO expresion
2. allocation cam LC processing
3. Processing of literes
4. Intereditate code generation.

Pass III
1. Target code generation

Q.17 Macro Name Table :-

MACRO Name table (MNT) has Entries for all macro defined in a Program.

1. Each MNT Entry contains three Pointers MDTP, KPDTP and SSTP, which are pointed to MDT, KPDTAB and SSNATB, for the macro respectively.

2. MNT also contains number of Positional Parameters (#PP), number of keyword Parameter (#KP) and

Date : / /

of Expansion time variable (EV)

Q.18 parameter name Table.

→ parameter name table (PNTAB) contains all the parameter name used in macro.

→ It contains all keyword, default and positional parameters names.

Q.19 EV name Table :

→ EV Name Table [EVNTAB] contains all the Expansion time variable names.

→ Expansion time variable are local variable and global variable which are declared in macro.

Q.20 SS NAME Table

→ SS name table (SSNTAB) contains the sequencing symbols.

Page No. :
Date : / /

Q.2 Keyword Parameters Default Table.

→ Keyword Parameters Default table (KPDTAB) contains the name of keyword parameters and it's value.

Q.22) Macro Definition Table in

Macro definition table (MDT) entries are constructed while expressing the model statement and preprocessor statements in the macro body.

Page No.:
Date: / /

Assignment - 5 Compiler.

Q) Different between static and dynamic memory allocations

Static memory Allocation	Dynamic memory Allocation
In this memory is allocated to a variable before the execution of a program begin.	The memory binding are established and destroyed during the execution of program
No, memory allocation and deallocation action are performed during the execution of a program.	Memory is allocated and deallocated during the execution of program
Variable remain permanently allocated allocation of variable exists over if the program unit in which is it define is not active.	The variable get allocated only if the all program unit get active. - e.g PL/I, pascal, Ada etc.

Page No.:
Date: / /

(b) explain scope rule with example.

→ For data declaration it is possible to use same name in memory blocks Program.

→ This would be established memory bindings of the form (name, variable) for different value of a variable.

→ scope rule determines which of these binding so, it effective at a specific place in the program

→ Eg:- consider the block structured program. The variable accessible within the variable blocks are as follow :-

```
A [ x, Y, Z : integer ;
    B [ q : ... ;
        C [ h, 2 : real ;
    A [ i, j, : integer ;
```

Page No.:
Date: / /

Block	Accessible variables	
	local	non local
A	xA, yA, zA	-
B	qB	xA, yA, zA
C	hC, zC	xA, yA, qB
D	i0, j0	xA, yA, zA

3) variable zA is not accessible inside Block C since C, contains a declaration using the name z, Thus, zA, and zC, are two distinent variable this would be true even if they had identical attributes. ie. even if z of way declared to be an integer.

Q. Explain memory allocation in Recursion.

4) Recursion Procedure (or, function) are characterized by the fact that many invocations of a Procedure coexist during the execution of a Program.

Page No.:
Date: / /

3) A copy of the local variable of the procedure must be allocated for each invocation.

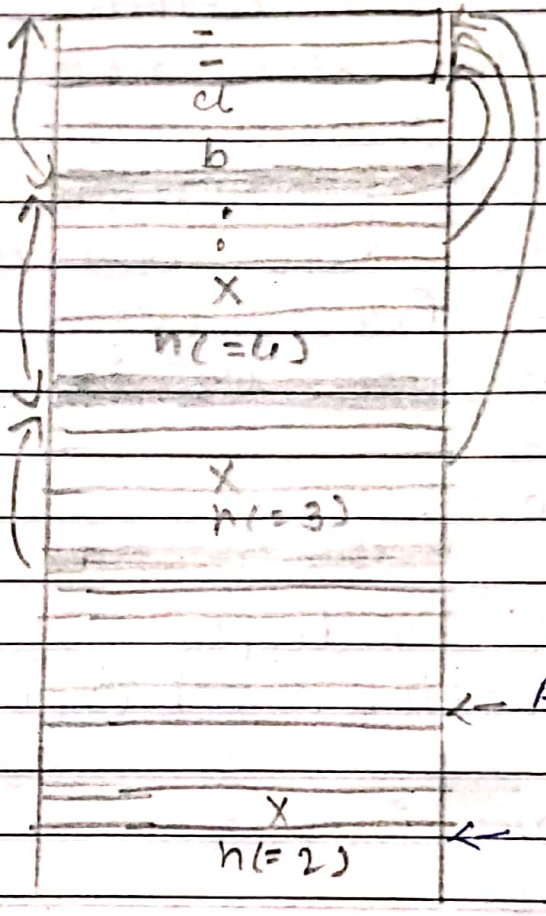
c) This does not pose any problem when a stack model of memory allocations is used because an activation record is created for every introduction of a procedure or function.

4) Program sample (input, output);
var
a, b: integer;
function fib(n): integer;
var
x: integer
begin
if n > 2 then
x := fib(n-1) + fib(n-2);
else
x := 1;
return(x);
end fib;
begin
fib(4);
end.

5) For the call fib(4) the function makes recursive calls.

Page No. :
Date : / /

c) pictorial representation of recursive call in fibonacci series program.



d) define dope vector with Example.

e) If array dimension bounds are not known during compilation the dop vector has to exist during program execution.

f) The number of dimention of an array determines the format and

size of its DV.

The DOP vector is allocated in the A.R of a block and dov, it's displacement in AR, is noted in the symbol table entry of the Array.

```
MOVER    AREG I
COMP     AREG = "5"
BC       GT, I, ERROR, RTN, Error if i>5
COMP     AREG = '1'
BC       LT, ERROR, RTN, Error if i<1
MOVER    AREG, 'I'
COMP     AREG = "10"
BC       LT, ERROR, RTN Error if j>10
OLT      AREG, '-5', Mutiply by range
ADD      AREG, I
MULT     AREG, ='2' Mutiply by
                    element size
ADD      AREG, ADDR, BASE, Add,
                    Address of a[0,0]
```

The array is allocated dynamic by repeting step for the array.

Date : / /

- It's start address value of dj, l, j, and range if are entered in the DV.

- The generated code used a clor to acess the complets of DV to check the validity of subtraction and compute the address of an array element.

Define with Example.

1) operand descripter.

- It has the following fields.
- 1) Attributes : It conteins the subfield types length and miscellaneous information.

2) Addressubility : It spesifies where the opprends is located and between it can be assessed. It has 2 subfields.
   1) Addresbility code
   2) Address

Page No. :
Date : / /

- An operand descriptor is built for every operand participating in an expression.
- An operand descriptor is built for each id when the id reduced during parsing.
- A partial result Pri is the result of evaluating some operator opj... A descriptor is built for Pri immediately after code is generated for operator opj.
- For simplicity assume that all operand descriptor are stored in an array called operand descriptor.

Eg: MOVER AREA, A
MOLT AREA, B

Operand descriptor :-

1.	(int, +)	M, addr(a)	Pesmip for a
2	(int, 1)	M, addr(b)	" b
3	(int, 1)	R, (addr(AREA	" a x

(ii) Register Descriptor :-

A Register descriptor has two fields

1) status :-
containd the code free or occupied. For indicate register status.

2) operand descriptor # in

If status = occupied, this field contains the descripter # for the operand contained in the register.

3) Register descriptor are stored in an array called Register discriptor.

4) one register discripter exists for each CPU register.

E.g in
The register descriptor for ARG after generating code for a+, b.

occupied	#3

This indicates that register AREG (contains the operand described by descriptor 1

Q.1 write the steps for Assembly lang code generation from an expression specified in high level language.

Ans Expression : a = x + y * p + q

step no.	Parsing action.	code generation
1	$<id>x \rightarrow E^1$	Build discriptor #1
2	$E^1 \rightarrow T^1$	-
3	$<id>y \rightarrow E^2$	Build descriptor #2
4	$T^1 + E^2 \rightarrow T^3$	Generate MOVER, AREG, x, ADD, AREG, y, Build descriptor #3.
5.	$T^3 \rightarrow E^3$	Build descriptor #
6.	$<id>p \rightarrow E^4$	Build descript #
7	$E^4 \rightarrow T^4$	-
8.	$<id>q \rightarrow E^5$	Build descriptor
9.	$T^4 * E^5 \rightarrow T^6$	Generate MOVER

Date : / /

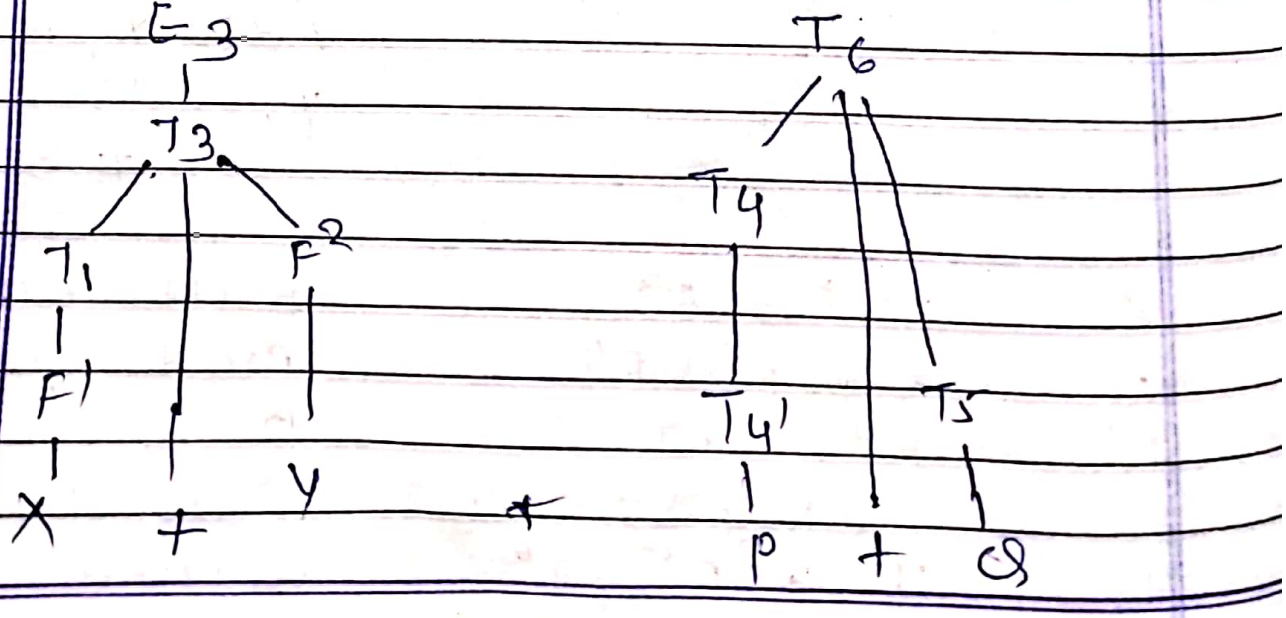
TEM, MOVER, ADD, R
ADD, AREG, Q
Build descriptor for

10 $E^3 + T^6 \rightarrow E^6$ Generate MULT AREG TEM

⇒ operand descriptor in

1	(int, I)	m, addr(x)
2	(int, I)	m, addr(y)
3	(int, I)	m, addr[temp(I)]
4	(int I)	m, addr(P)
5	(int, I)	m, addr(Q)
6	(int, I)	m addr(AREG)

2)

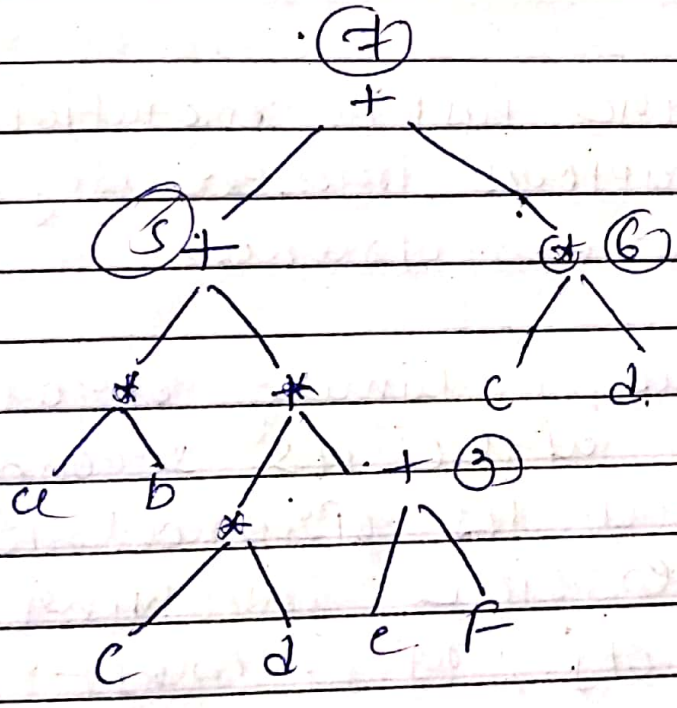


=> Register descriptor

R x c	c

with an example, explain the alternative of assembly language code generation for an expression

X + Y + P * Q + (R + S) + P * Q




```
MOVER    AREG, X
MOLT     AREG, Y
MOVEM    AREG, TEMP 1
MOVER R  AREG, P
MULT     AREG, Q
MOVEM    AREG, TEMP-2
MOVER M  AREG, R
```

```
MOVER    AREG, S
MULT     AREG, TEMP-2
ADD      AREG, TEMP-3
MOVER    AREG, P
MULT     AREG, Q
ADD      AREG, TEMP-3
```

Q.) How Postfix string can be used for generating the intermediate code for expression.

⇒ In the postfix notation each operator appears immediately after its last operands.

⇒ Thus, a binary operator opp. appearers after its second oprend If any of its oprands is itself an expression involving an oprator opj, pop j, must appear, before opi.

E.g :- 1- a + b * c + d * eTF-1

source string.

Postfix string is a popular intermed code in non-optiminimum compiler due to ease of generation at we

- The code generation from the postfix string using a stack of operand descriptor.
- operand descriptior are pused on the stack as oprand appear in the string.
- when an oprater with curity x, appears in the string is descriptor are popped of the stack.
- A conversion of fix to postfix is modified i.e. opercends appearing in the source string are coppied into the postfix string straightway.
- The TOS oprator is popped into the postfix string is to c operators = current oprator.
- How triples and quadraples can be used for generating the intermedicu code for expression.
- Triples :- A triples is a representat of an elementary operation in

the form of a pseudo machine instruction.

Format :-

operator	operand 1	operand 2

⇒ each operand of a triple is either a variable / constant or the result of some evaluation represented by another triple.

⇒ conversion of infix string into triples can be achieved by a slight modification i.e.-

infix string :- a - a + b * c + d * e F - 1

⇒ [I] in the operand field of triples is indicate that the operands i.e. the value of b * c, represented by triple number 1.

	operator	operands	operand 2
1	*	b	c
2	+	[1]	a
3	*	e	F
4	*	d	[3]
5	+	[2]	[4]

Page No. :
Date : / /

quadruples

A quadruples represents an elementry evaluation in the following format.

operator	operand 1	operand 2	result name

- here, result name designates the result of the evaluation. It can be used as the operand of another quadruples.

Eg: n	operator	operand 1	operand 2	Result name
1	*	b	c	t_1
2	+	t_1	a	t_2
3	↑	e	f	t_3
4	*	d	t_3	t_4
5	+	F_2	t_4	t_5

- List the step taken by compiler at function call. Procedure call.

- while implementing a function call the compiler must ensure the following.

(i) Actual parameter are accessible call the complier all function.
(ii) The called function is able to produce side effects according to the rules of the PL
(iii) control is transfered to end is referenced from the called function.
(iv) The function value is returned to the calling program.
(v) All other aspects of execution of a calling programs are unaffected by the function calls.

c) compiler used a set of rector to implement function.
(i) parameter list
(ii) calling convertions
(iii) save area,

d) Different function calls:-
e) different type of function calls are in

(i) call by value:- In this machenism value of actual

Page No. :
Date : / /

Parameters are passed to the called function.

c) These value are assigned to the corresponding formal parameters the passing as value only taken place in are direction i.e. from calling program to the called function.

d) In this value changed in the formal parameter does not effect the value of actual parameter.

(ii) call by value → Result → In this approximated the value of formal parameter is copied back into actual parameter.

e) This preanism indicates the simplicity of the all the by the value mechanism but increase higer overhead.

(ii) call by reference

→ In this the address of actual

parameter is passing to the called function.

- The parameter list is thus a list of address of actual parameter. At every access of a fermineal parameter in the function the address of the corresponding actual parameter is obtain.

- If the value is change in the formal parameter it will after the value of actual Parameter.

(iii) call by name :- In this also the same mechinism is follow of the call by reference.

- But the only diffrence is that if every occurence of a formal parameter in the body of called function is replaced by the name of the corresponding actual parameter.

=> Here, also change in formal parameter effect the value of actual parameter.

D E P A R T M E N T O F C O M P U T E R S C I E N C E
R O L L W A L A C O M P U T E R C E N T R E
G U J A R A T U N I V E R S I T Y
M.C.A. – III

R O L L NO : 36
N A M E : Preksha Sheth
S U B J E C T : System Software

NO.	TITLE	PAGE NO.	DATE	SIGN
1	Assembler Pass1		**10/12/2020**	
2	Assembler Pass2		**10/12/2020**	
3	Macro		**10/12/2020**	
4	Scanner		**10/12/2020**	
5	Recursive Decent Parser		**10/12/2020**	
6	Parser without Backtracking		**10/12/2020**	
7	LL(1) Parser		**10/12/2020**	
8	Operator Precedence Parser		**10/12/2020**	

```
**********************************************************************
**********************************************************************
Name :   Preksha Sheth
Roll no: 36
Subject: System Software
Class:   MCA - III
**********************************************************************
**********************************************************************
2 Pass Assembler
**********************************************************************
**********************************************************************
Pass 1:-
**********************************************************************
**********************************************************************
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.*;
import java.io.IOException;
import java.util.*;
import java.util.Map;


public class pass1
{
     public static void main(String args[])throws IOException
     {
          FileReader fr=null;
          FileWriter fw=null;
          BufferedReader br=null;
          BufferedWriter bw=null;
          try
          {
               String
inputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\Input.txt";
               fr=new FileReader(inputfilename);
               br=new BufferedReader(fr);

               String
outputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\IC.txt";
               fw=new FileWriter(outputfilename);
               bw=new BufferedWriter(fw);
               Hashtable<String,String> is=new
Hashtable<String,String>();
```

```
                is.put("STOP", "00");
                is.put("ADD", "01");
                is.put("SUB", "02");
                is.put("MULT", "03");
                is.put("MOVER", "04");
                is.put("MOVEM", "05");
                is.put("COMP", "06");
                is.put("BC", "07");
                is.put("DIV", "08");
                is.put("READ", "09");
                is.put("PRINT", "10");

                Hashtable<String,String> dl=new
Hashtable<String,String>();
                dl.put("DC","01");
                dl.put("DS","02");

                Hashtable<String,String> ad=new
Hashtable<String,String>();
                ad.put("START","01");
                ad.put("END", "02");
                ad.put("ORIGIN", "03");
                ad.put("EQU", "04");
                ad.put("LTORG", "05");

                Hashtable<String,String> cd=new
Hashtable<String,String>();
                cd.put("LT","1");
                cd.put("LE","2");
                cd.put("EQ","3");
                cd.put("GT","4");
                cd.put("GE","5");
                cd.put("ANY","6");

                Hashtable<String,String> symtab=new
Hashtable<String,String>();
                Hashtable<String,Integer> symtabi=new
Hashtable<String,Integer>();
                Hashtable<String,String> littab=new
Hashtable<String,String>();
                Hashtable<String,Integer> littabi=new
Hashtable<String,Integer>();
                ArrayList<Integer> pooltab=new
ArrayList<Integer>();

                String sCurrentLine;
                int locptr=0;
```

```
                int litin=0;
                int litptr=1;
                int symptr=1;
                int pooltabptr=1;

                sCurrentLine=br.readLine();

                String s1=sCurrentLine.split(" |   ")[1];
                if(s1.equals("START"))
                {
                     bw.write("     AD\t01 \t");
                     String s2=sCurrentLine.split(" |   ")[2];
                     bw.write("C \t"+s2+"\n");
                     locptr=Integer.parseInt(s2);
                }

                while((sCurrentLine=br.readLine())!=null)
                {
                     int LC=0,q=0;
                     String type=null;

                     int flag2=0;   //checks whether addr is
assigned to current symbol
                     String cs[]=sCurrentLine.split(" |\\,|  ");
                          String s=sCurrentLine.split(" |\\,|
     ")[0];    //consider the first word in the line
                          if(s.contains("="))
                          {
                               int in,l;
                               String sub;
                               in=s.indexOf(39);
                               in++;
                               l=s.length()-1;
                               sub=s.substring(in,l);

     bw.write("\r\n"+locptr+")\tL\t"+s);
                               locptr++;
                               continue;
                          }
                          for(Map.Entry m:symtab.entrySet())
                          {
                               if(s.equals(m.getKey()) &&
!s.equals(""))
                               {
                                    m.setValue(locptr);
                                    flag2=1;
                               }
```

```
                    }
                if (s.length() != 0 && flag2 == 0 &&
!s.equals(""))
                {         //if current string is not " " or
addr is not assigned,

            //then the current string must be a new symbol.

                    symtab.put(s, String.valueOf(locptr));
                    symtabi.put(s,symptr);
                    symptr++;
                }

                int isOpcode = 0;        //checks whether
current word is an opcode or not

                s = sCurrentLine.split(" |\\,")[1];
          //consider the second word in the line
                for (Map.Entry m : is.entrySet()) {
                    if (s.equals(m.getKey())) {
                        bw.write("\r\n"+locptr+")
     IS\t" + m.getValue() + "\t");      //if match found in
imperative stmt
                        type = "is";
                        isOpcode = 1;
                    }
                }

                int q1=0;
                for (Map.Entry m : ad.entrySet()) {
                    if (s.equals(m.getKey()) &&
!s.equals("END") && !s.equals("EQU") && !s.equals("ORIGIN")&&
!s.equals("LTORG")) {
                        bw.write("\r\n"+locptr+")
     AD\t" + m.getValue() + "\t");      //if match found in
Assembler Directive
                        type = "ad";
                        isOpcode = 1;
                    }
                    else if (s.equals(m.getKey()) &&
s.equals("END"))
                    {
                        bw.write("\r\n\tAD\t" +
m.getValue() + "\t");         //if match found in Assembler
Directive
                        type = "ad";
                        isOpcode = 1;
```

```
                    }
                    else if(s.equals(m.getKey()) &&
s.equals("EQU"))
                    {
                         bw.write("\r\n\tAD\t" +
m.getValue() + "\t");          //if match found in Assembler
Directive
                         type = "ad";
                         isOpcode = 1;
                         LC=1;
                    }
                    else if(s.equals(m.getKey()) &&
s.equals("ORIGIN"))
                    {
                         q1=1;
                         break;
                    }

               }
               for (Map.Entry m : dl.entrySet()) {
                    if (s.equals(m.getKey())) {
                         bw.write("\r\n"+locptr+")
     DL\t" + m.getValue() + "\t");      //if match found in
declarative stmt
                         type = "dl";
                         isOpcode = 1;
                    }
               }


               if (s.equals("LTORG"))
               {
                    pooltab.add(pooltabptr);
                    LC=1;
                    int llocp=locptr;
                    for (Map.Entry m : littab.entrySet()) {
                         if (m.getValue() == "") {
          //if addr is not assigned to the literal
                              m.setValue(locptr);
                              locptr++;
                              pooltabptr++;
                              LC = 1;
                              isOpcode = 1;
                         }
                    }
                    locptr=llocp;
                    bw.write("\r\n\tAD\t05\t");
```

```
                    continue;
                }


                if (s.equals("END")) {
                    pooltab.add(pooltabptr);
                    for (Map.Entry m : littab.entrySet()) {
                        if (m.getValue() == "") {
                            m.setValue(locptr);
                            locptr++;
                            LC = 1;
                        }
                    }
                }


                if(s.equals("EQU"))
                {
                    int sss=0;
                    for(Map.Entry mm:symtab.entrySet())
                    {
                        if(sCurrentLine.split("
|\\,")[3].equals(mm.getKey()) && sss==0)
                        {
                            for(Map.Entry
mmm:symtab.entrySet())
                            {
                                if(sCurrentLine.split("
|\\,")[0].equals(mmm.getKey()))
                                {

     mmm.setValue(mm.getValue());
                                    sss=1;
                                    break;
                                }
                            }
                        }
                        else if(sss==1)
                        {
                            break;
                        }
                    }
                    //symtab.put("equ",
String.valueOf(locptr));
                }
```

```
                    if (sCurrentLine.split(" |\\,").length > 2)
                    {         //if there are 3 words
                         s = sCurrentLine.split(" |\\,")[2];
                    //consider the 3rd word



                         //this is our first operand.

                         //it must be either a
Register/Declaration/Symbol
                         if (s.equals("AREG")) {
                              bw.write("1\t");
                              isOpcode = 1;
                         } else if (s.equals("BREG")) {
                              bw.write("2\t");
                              isOpcode = 1;
                         } else if (s.equals("CREG")) {
                              bw.write("3\t");
                              isOpcode = 1;
                         } else if (s.equals("DREG")) {
                              bw.write("4\t");
                              isOpcode = 1;
                         } else if (type == "dl") {
                              bw.write("C\t" + s + "\t");
                         }
                         else if(q1==1 && sCurrentLine.split("
|\\,")[1].equals("ORIGIN"))
                    {
                              LC=1;
                              String ns=null;
                              if(sCurrentLine.split("
|\\,")[2].contains("+"))
                                   {
                                        int
in=sCurrentLine.split(" |\\,")[2].indexOf('+');
                                        ns=sCurrentLine.split("
|\\,")[2].substring(0,in);
                                   }
                              for(Map.Entry m:symtab.entrySet())
                              {
                                   if(ns.equals(m.getKey()))
                                   {
                                        int nn=0,in=0;

     locptr=Integer.parseInt((String)m.getValue());
```

```
                                        in=sCurrentLine.split("
|\\,")[2].indexOf('+');
                                        in=in+1;
                                        ns=sCurrentLine.split("
|\\,")[2].substring(in);

     nn=Integer.parseInt((String)ns);
                                        locptr=locptr+nn;

     System.out.println("LOOP: "+locptr);
                                             break;
                                        }
                                   }
                              }
                              else
                              {
                                   int qqq=0;
                                   for(Map.Entry m:cd.entrySet())
                                   {
                                        if(s.equals(m.getKey()) &&
qqq==0)
                                        {

     bw.write(m.getValue()+"\t");
                                             qqq=1;
                                             break;
                                        }
                                   }
                                   if(qqq==0)
                                   {
                                        symtab.put(s, "");
                                   }
     //forward referenced symbol
                              }
                         }


                         if (sCurrentLine.split(" |\\,").length > 3)
                         {         //if there are 4 words

                              s = sCurrentLine.split(" |\\,")[3];
                    //consider 4th word.

                              //this is our 2nd operand

                              //it is either a literal, or a symbol
                              if (s.contains("=")) {
```

```
                                litin++;
                                littabi.put(s,litin);
                                littab.put(s, "");
                                bw.write("L\t" + litptr + "\t");
                                isOpcode = 1;
                                litptr++;
                           }
                           else
                           {
                                     int sq=0,qq=0;
                                     for(Map.Entry
m:symtab.entrySet())
                                     {
                                          sq++;
                                          if(s.equals(m.getKey())
&& qq!=1)
                                          {
                                               for (Map.Entry
mm:symtabi.entrySet())
                                               {

     if(s.equals(mm.getKey()))
                                                    {

     bw.write("S\t" +mm.getValue()+ "\t");
                                                         qq=1;
                                                         break;
                                                    }
                                               }

                                          }
                                          else if(qq==1)
                                          {
                                               break;
                                          }
                                     }
                                if(qq==0)
                                {
                                     symtab.put(s, "");
     //Doubt : what if the current symbol is already present in
SYMTAB?

     //Overwrite?
                                     bw.write("S\t" + symptr +
"\t");
                                     symtabi.put(s,symptr);
```

```
                                        symptr++;
                                   }
                              }
                         }

                         bw.write("\n");          //done with a line.

                         if (LC == 0)
                              locptr++;
                    }

                    String f1 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt";
                    FileWriter fw1 = new FileWriter(f1);
                    BufferedWriter bw1 = new BufferedWriter(fw1);
                    for (Map.Entry m : symtab.entrySet())
                    {
                         for(Map.Entry mm:symtabi.entrySet())
                         {
                              String str=(String)mm.getKey();
                              if(str.equals(m.getKey()))
                              {

     bw1.write(mm.getValue()+"\t"+m.getKey() + "\t" +
m.getValue()+"\r\n");
                                   System.out.println(m.getKey() + "
" + m.getValue());
                              }
                         }
                    }

                    String f2 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt";
                    FileWriter fw2 = new FileWriter(f2);
                    BufferedWriter bw2 = new BufferedWriter(fw2);
                    for (Map.Entry m : littab.entrySet())
                    {
                         for(Map.Entry mm:littabi.entrySet())
                         {
                              System.out.println("MM:
"+mm.getValue());
                              String str=(String)mm.getKey();
                              if(str.equals(m.getKey()))
                              {

     bw2.write(mm.getValue()+"\t"+m.getKey() + "\t" +
m.getValue()+"\r\n");
```

```
                                    System.out.println(mm.getValue()+"
"+m.getKey() + " " + m.getValue());

                              }
                         }
                    }

                    String f3 = "C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\POOLTAB.txt";
                    FileWriter fw3 = new FileWriter(f3);
                    BufferedWriter bw3 = new BufferedWriter(fw3);
                    for (Integer item : pooltab) {
                         bw3.write(item+"\r\n");
                        System.out.println(item);
                    }

                    bw.close();
                    bw1.close();
                    bw2.close();
                    bw3.close();

               }
               catch (Exception e)
               {
                    e.printStackTrace();
               }
          }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>javac pass1.java
D:\MCA-III\SS File>java pass1
TWO 213
A 214
ONE 212
INPUT 200
LOOP 201
BACK 201
B 215
MM: 1
1 ='2' 208
1
2
```

```
**********************************************************************
**********************************************************************
Input file:- Input.txt
**********************************************************************
**********************************************************************
 START 200
INPUT READ  A
LOOP MOVER AREG,A
 MOVEM BREG,A
 ADD AREG,ONE
 ADD BREG,TWO
 COMP BREG,='2'
 BC LT,LOOP
 BC GT,BACK
 LTORG
='2'
BACK EQU  LOOP
 MOVEM BREG,B
 PRINT  B
 STOP
ONE DC '1'
TWO DC '2'
A DS 1
B DS 1
 END
**********************************************************************
**********************************************************************
Intermediate Code:- IC.txt
**********************************************************************
**********************************************************************
     AD   01   C    200

200) IS   09   S    2

201) IS   04   1    S    2

202) IS   05   2    S    2

203) IS   01   1    S    4

204) IS   01   2    S    5

205) IS   06   2    L    1

206) IS   07   1    S    3

207) IS   07   4    S    6
```

```
     AD   05
208) L    ='2'
     AD   04   S    3

209) IS   05   2    S    7

210) IS   10   S    7

211) IS   00

212) DL   01   C    '1'

213) DL   01   C    '2'

214) DL   02   C    1

215) DL   02   C    1

     AD   02
```

****
****

LITTAB:- LITTAB.txt

****
****

```
1    ='2' 208
```

****
****

POOLTAB:- POOLTAB.txt

****
****

```
1
2
```

****
****

SYMTAB:- SYMTAB.txt

****
****

```
5    TWO  213
2    A    214
4    ONE  212
1    INPUT     200
3    LOOP 201
6    BACK 201
7    B    215
```

```
**********************************************************************
**********************************************************************
Pass 2:-
**********************************************************************
**********************************************************************
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.*;
import java.io.IOException;
import java.util.*;
import java.util.Map;


public class pass2
{
     public static void main(String args[])throws IOException
     {
          FileReader fr=null;
          FileWriter fw=null;

          BufferedReader br=null;
          BufferedWriter bw=null;

          try
          {
               String
inputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\IC.txt";

               String
outputfilename="C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\MC.txt";

               fr=new FileReader(inputfilename);
               fw=new FileWriter(outputfilename);
               br=new BufferedReader(fr);
               bw=new BufferedWriter(fw);

               if(br==null)
               {
                    System.out.println("NULL!");
               }

               String sCurrentLine=br.readLine();
               while((sCurrentLine=br.readLine())!=null)
```

```
                {
            if(sCurrentLine.trim().isEmpty())
            {
                 continue;
            }
                        String lc=null;
                        if(sCurrentLine.split(" |
      ")[1].equals("AD"))
                        {
                             continue;
                        }
                        lc=sCurrentLine.split(" |     ")[0];
                        if(sCurrentLine.split(" |
      ")[1].equals("L"))
                        {
                                  String s=sCurrentLine.split(" |
      ")[2];
                                  if(s.contains("="))
                                  {
                                       int in,l;
                                       String sub;
                                       in=s.indexOf(39);
                                       in++;
                                       l=s.length()-1;
                                       sub=s.substring(in,l);

      bw.write("\r\n"+lc+"\t+00\t0\t");

      if(Integer.parseInt((String)sub)<10)
                                       {
                                            bw.write("00"+sub);
                                       }
                                       else
if(Integer.parseInt((String)sub)>=10 &&
Integer.parseInt((String)sub)<100)
                                       {
                                            bw.write("0"+sub);
                                       }
                                       else
                                       {
                                            bw.write(sub);
                                       }

      //bw.write("\r\n"+locptr+")\t"+sub);
                                       continue;
                                  }
                        }
```

```
                    if(sCurrentLine.split(" |
     ")[1].equals("DL"))
                    {
                         if(sCurrentLine.split(" |
     ")[2].equals("01"))
                         {

                              bw.write("\r\n"+lc+"\t+00\t0\t");
                              String s=sCurrentLine.split(" |
     ")[4];
                              int in=s.indexOf(39);
                              in++;
                              int l=s.length();
                              l--;
                              s=s.substring(in,l);
                              if(Integer.parseInt((String)s)<10)
                              {
                                   bw.write("00"+s);
                              }
                              else
if(Integer.parseInt((String)s)>=10 &&
(Integer.parseInt((String)s)<100))
                              {
                                   bw.write("0"+s);

                              }
                              else
                              {
                                   bw.write(s);

                              }
                         }
                         else
                         {

                         bw.write("\r\n"+lc+"\t+00\t0\t000");
                         }
                         continue;
                    }
                    if(sCurrentLine.split(" |
     ")[1].equals("IS") && sCurrentLine.split(" |
     ")[2].equals("00"))
                    {
                         bw.write("\r\n"+lc+"\t+00\t0\t000");
                         continue;
                    }
                    bw.write("\r\n"+lc);
```

```
                    String s=sCurrentLine.split(" |    ")[2];
                    bw.write("\t+"+s);
                    if(sCurrentLine.split(" |     ").length>5)
                    {
                         s=sCurrentLine.split(" | ")[3];
                         bw.write("\t"+s);
                         String s1=sCurrentLine.split(" |
     ")[4];
                         String s2=null;
                         s2=sCurrentLine.split(" |     ")[5];
                         if(s1.equals("S"))
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
                              {
                                   if(sline.trim().isEmpty())
                                   {
                                        continue;
                                   }
                                   if(s2.equals(sline.split(" |
     ")[0]))
                                   {
                                        s2=sline.split(" |
     ")[2];
                                        bw.write("\t"+s2);
                                        qr=1;
                                        break;
                                   }
                              }
                              br1.close();
                         }
                         else if(s1.equals("L"))
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
```

```
                                {
                                     String str=sline.split(" |
     ")[0];
                                     if(s2.equals(str))
                                     {
                                          str=sline.split(" |
     ")[2];
                                          bw.write("\t"+str);
                                          qr=1;
                                          break;
                                     }
                                }
                                br1.close();
                           }
                      }
                      else if(sCurrentLine.split(" |
     ").length==5)
                      {
                           bw.write("\t0");
                           String s1=sCurrentLine.split(" |
     ")[3];
                           String s2=sCurrentLine.split(" |
     ")[4];
                           if(s1.equals("S"))
                           {
                                int cc=0;
                                BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\SYMTAB.txt"));
                                String sline=null;
                                int qr=0;
                                while((sline=br1.readLine())!=null
&& qr==0)
                                {
                                     if(s2.equals(sline.split(" |
     ")[0]))
                                     {
                                          s2=sline.split(" |
     ")[2];
                                          bw.write("\t"+s2);
                                          qr=1;
                                          break;
                                     }
                                }
                                br1.close();
                           }
                           else if(s1.equals("L"))
```

```
                         {
                              int cc=0;
                              BufferedReader br1=new
BufferedReader(new FileReader("C:\\Users\\User\\Desktop\\MCA
3\\SS\\programs\\pass1n\\LITTAB.txt"));
                              String sline=null;
                              int qr=0;
                              while((sline=br1.readLine())!=null
&& qr==0)
                              {
                                   int in=sline.split(" |
     ")[0].indexOf(39);
                                   in=in+1;
                                   int l=sline.split(" |
     ")[0].length();
                                   l=l-1;
                                   String str=sline.split(" |
     ")[0].substring(in,l);
                                   if(s2.equals(str))
                                   {
                                        bw.write("\t00"+str);
                                        qr=1;
                                        break;
                                   }
                              }
                              br1.close();
                         }
                    }
               }
               br.close();
               bw.close();
          }
          catch(Exception e)
          {
               e.printStackTrace();
          }
     }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>javac pass2.java

D:\MCA-III\SS File>java pass2
```

```
**********************************************************************
**********************************************************************
Machine Code:- MC.txt
**********************************************************************
**********************************************************************

200) +09  0    214
201) +04  1    214
202) +05  2    214
203) +01  1    212
204) +01  2    213
205) +06  2    208
206) +07  1    201
207) +07  4    201
208) +00  0    002
209) +05  2    215
210) +10  0    215
211) +00  0    000
212) +00  0    001
213) +00  0    002
214) +00  0    000
215) +00  0    000
**********************************************************************
**********************************************************************
Macro:-
**********************************************************************
**********************************************************************
Pass - 1:-
**********************************************************************
**********************************************************************
import java.util.*;
import java.io.*;

class Macro_Expansion{
     public static String MDT_check(String
word,ArrayList<String> SSNTAB,ArrayList<String>
EVT,ArrayList<String> PNT,String mdt_word,int
count,ArrayList<String> MODEL)
     {
         if(SSNTAB.contains(word))
               mdt_word=mdt_word+"(S ,"
+(SSNTAB.indexOf(word)+1)+")";
          else if(PNT.contains(word.substring(1,word.length())))
               mdt_word=mdt_word+"(P ,"
+(PNT.indexOf(word.substring(1,word.length()))+1)+")";
          else if(EVT.contains(word.substring(1,word.length())))
```

```
                mdt_word=mdt_word+"(E ,"
+(EVT.indexOf(word.substring(1,word.length()))+1)+")";
          else
          {
              if(count==1 && !MODEL.contains(word)){}
               else
               mdt_word=mdt_word+" "+word+" ";

          }
          return mdt_word;
     }
     public static void main(String args[])throws IOException{
          File f1=new File("Macro.txt");
          BufferedReader br_macro=new BufferedReader(new
FileReader(f1));

          String
line=null,word,mdt_word=null,prev_word=null,line1;

          int
count=0,pos_count=0,key_count=0,ev_count=0,n,count_token=0,ssnta
b_count=0,line_no=1,mdt_ptr=1;

          ArrayList<String> MNT=new ArrayList<String>();
          ArrayList<String> PNT=new ArrayList<String>();
          ArrayList<String> KPD=new ArrayList<String>();
          ArrayList<String> EVT=new ArrayList<String>();
          ArrayList<String> MDT=new ArrayList<String>();
          ArrayList<String> Model=new ArrayList<String>();
          ArrayList<String> SSNTAB=new ArrayList<String>();
          ArrayList<Integer> SSTAB=new ArrayList<Integer>();

          File f2=new File("Models.txt");
          BufferedReader br=new BufferedReader(new
FileReader(f2));

          while((line1=br.readLine())!=null)
          {
               Model.add(line1);
          }
          while((line=br_macro.readLine())!=null)
          {
               count_token=1;
               mdt_word=" ";
               StringTokenizer st1= new StringTokenizer(line);
               n=st1.countTokens();
               while(st1.hasMoreTokens())
```

```
                {
                     word=st1.nextToken();
                     if(word.equals("MACRO"))
                     {
                          count++;
                          continue;
                     }
                     if(count==1)
                     {
                          MNT.add(word);
                          while(count<n)
                          {
                               word=st1.nextToken();
                              if(word.contains("="))
                               {
                                  key_count++;
                                  String[] str=word.split("=",2);
                                  String key_par;

key_par=str[0].substring(1,str[0].length());
                                  PNT.add(key_par);
                                  KPD.add(key_par);
                                  KPD.add(str[1]);
                               }
                              else
                               {
                                    String pos_par;

     pos_par=word.substring(1,word.length()-1);
                                    PNT.add(pos_par);
                                    pos_count++;
                               }
                             count++;
                          }
                     }
                     if(word.equals("LCL"))
                     {
                          mdt_word=mdt_word+word+" ";
                          word=st1.nextToken();

     EVT.add(word.substring(1,word.length()));
                          ev_count++;
                          mdt_word=mdt_word+"(E ,"+ev_count+")";
                     }
                     if(count_token==2 && Model.contains(word))
                     {
                          ssntab_count++;
```

```
                         SSNTAB.add(prev_word);
                    }
                    if(count_token==2 &&
SSNTAB.contains(prev_word))
                    {
                         int q=SSNTAB.indexOf(prev_word);
                         SSTAB.add(mdt_ptr);
                    }
                    if(line_no>3 && !word.equals("LCL"))
                    {

     mdt_word=MDT_check(word,SSNTAB,EVT,PNT,mdt_word,count_token
,Model);
                    }
                    count_token++;
                    prev_word=word;
               }
               if(!mdt_word.equals(" "))
               {
               MDT.add(mdt_word);
               mdt_ptr++;
               }
               line_no++;
          }
          MNT.add(Integer.toString(pos_count));
          MNT.add(Integer.toString(key_count));
          MNT.add(Integer.toString(ev_count));
          MNT.add("1");
          MNT.add("1");
          MNT.add("1");

         System.out.println("\nMNT table: "+MNT);
          System.out.println("\nPNT table: "+PNT);
          System.out.println("\nKPD table: "+KPD);
          System.out.println("\nEV table: "+EVT);
          System.out.println("\nSSTAB table: "+SSTAB);
          System.out.println("\nSSNTAB table: "+SSNTAB);
          for(String s:MDT)
          {
               System.out.println(s);
          }
     }
}
```

****

****

OUTPUT

```
**********************************************************************
**********************************************************************
D:\MCA-III\SS File>javac Macro_Expansion.java

D:\MCA-III\SS File>java Macro_Expansion

MNT table: [CLEARMEM, 2, 1, 1, 1, 1, 1]

PNT table: [X, N, REG]

KPD table: [REG, AREG]

EV table: [M]

SSTAB table: [4]

SSNTAB table: [MORE]
 LCL (E ,1)
 (E ,1) SET  0
  MOVER (P ,3) ,  ='0'
  MOVEM (P ,3) , (P ,1) + (E ,1)
 (E ,1) SET (E ,1) +  1
  AIF  ( (E ,1) NE (P ,2) ) (S ,1)

D:\MCA-III\SS File>
**********************************************************************
**********************************************************************
Macro.txt
**********************************************************************
**********************************************************************
     MACRO
     CLEARMEM &X, &N, &REG=AREG
     LCL &M
&M   SET 0
     MOVER &REG , ='0'
MORE MOVEM &REG , &X + &M
&M   SET &M + 1
     AIF ( &M NE &N ) MORE
     MEND
**********************************************************************
**********************************************************************
Models.txt
**********************************************************************
**********************************************************************
ADD
SUB
MUL
```

```
MOVER
MOVEM
COMP
BC
DIV
READ
PRINT
AIF
AGO
**********************************************************************
**********************************************************************
Scanner:-
**********************************************************************
**********************************************************************
import java.util.*;
import java.io.*;
class DFA1{
    static String table[][];
    public static void main(String args[]) throws Exception{
        Scanner sc=new Scanner(System.in);
        System.out.println("Regular Expression :
aa+(c|d)*b(bd)*");
        System.out.println("Enter Expression : ");
        String expr=sc.nextLine();
        getRules();
        eval(expr);
    }
    static void eval(String expr){
        String curr_state=table[1][0],next,curr_char;
        for(int i=0;i<expr.length();i++){
            curr_char=expr.charAt(i)+"";
            next=getnext(curr_state,curr_char);
            if(next == null || next.equals("-")){
                System.out.println("Invalid Expression");
            }
            curr_state=next;
        }
        if(curr_state.equals("D")){
            System.out.println("Valid Expression");
        }
        else{
            System.out.println("Invalid Expression");
        }
    }
    static String getnext(String curr_state,String curr_char){
        int row=0,col=0;
        for(int i=0;i<table.length;i++){
```

```
            if(curr_state.equals(table[i][0])){
                row=i;
                break;
            }
        }
        for(int i=0;i<table[0].length;i++){
            if(curr_char.equals(table[0][i])){
                col=i;
                break;
            }
        }
        return table[row][col];
    }
    static void getRules() throws Exception{
        table=new String[6][5];
        BufferedReader br=new BufferedReader(new
FileReader("states.txt"));
        String line;
        int i=0,j;
        String token[];
        while((line=br.readLine())!=null){
            token=line.split("\t");
            for(j=0;j<token.length;j++){
                table[i][j]=token[j];
            }
            i++;
        }
        System.out.println("State Table : ");
        for(i=0;i<table.length;i++){
            for(j=0;j<table[i].length;j++){
                System.out.print(table[i][j]+"\t");
            }
            System.out.println("");
        }
    }
}
```

states.txt

```
s    a    b    c    d
A    B    -    -    -
B    C    -    -    -
C    C    D    C    C
```

```
D    -    E    -    -
E    -    -    -    D

**********************************************************************
**********************************************************************
 OUTPUT
**********************************************************************
**********************************************************************

D:\MCA-III\SS File>java DFA1
Regular Expression : aa+(c|d)*b(bd)*
Enter Expression :
aacbbd
State Table :
s       a       b       c       d
A       B       -       -       -
B       C       -       -       -
C       C       D       C       C
D       -       E       -       -
E       -       -       -       D
Valid Expression

**********************************************************************
**********************************************************************
RECURSIVE DESCENT PARSER:-
**********************************************************************
**********************************************************************
import java.io.*;
import java.util.*;

class TreeNode{
     String val;
     TreeNode leftChild,rightChild;

     TreeNode(){
          val="";
          leftChild=rightChild=null;
     }
     TreeNode(String val){
          this.val=val;
          leftChild=rightChild=null;
     }
     TreeNode(String val,TreeNode lchild,TreeNode rhild){
          this.val=val;
          leftChild=lchild;
          rightChild=rhild;
     }
```

```
     void dispPostOrder(){
          if(leftChild!=null)
               leftChild.dispPostOrder();
          if(rightChild!=null)
               rightChild.dispPostOrder();
          System.out.print(val);
     }
}

class RecDecParser{
     static int SSM=0;
     static String expression;

     public static void main(String args[]){
          Scanner sc=new Scanner(System.in);

     System.out.print("_____\nEnter
Expression : ");
          expression=sc.next();

     System.out.println("\n_____");
          TreeNode root=new TreeNode();

          root=Proc_E(expression);
          if(SSM != expression.length() || root==null )
               System.out.println("!!! INVALID EXPRESSION !!!");
          else{
               System.out.println("\nPostfix Expression
:\n_____");
               root.dispPostOrder();
          }

     System.out.println("\n_____");
     }

     static TreeNode Proc_E(String expression){
          TreeNode left,right;
          left=Proc_T(expression);
          while(SSM < expression.length() &&
expression.charAt(SSM)=='+' ){
               SSM++;
               right=Proc_T(expression);
               if(right==null)
                    return null;
               else
                    left=new TreeNode("+",left,right);
          }
```

```
          return left;
     }

     static TreeNode Proc_T(String expression){
          TreeNode left,right;
          left=Proc_V(expression);
          while(SSM < expression.length() &&
expression.charAt(SSM)=='*'){
               SSM++;
               right=Proc_V(expression);
               if(right==null)
                    return null;
               else
                    left=new TreeNode("*",left,right);
          }
          return left;
     }

     static TreeNode Proc_V(String expression){
          TreeNode node;
          if(SSM < expression.length() &&
expression.charAt(SSM)!='*' && expression.charAt(SSM)!='+'){
               node=new
TreeNode(expression.charAt(SSM++)+"",null,null);
               return node;
          }
          System.out.println("!!! INVALID EXPRESSION !!!");
          return null;
     }
}
```

****

****

OUTPUT

****

****

```
D:\MCA-III\SS File>JAVAC RecDecParser.java

D:\MCA-III\SS File>JAVA RecDecParser
_____________________________
Enter Expression : a+b*c


_____________________________

Postfix Expression :
_____________________________
abc*+
_____________________________
```

```
D:\MCA-III\SS File>JAVAC RecDecParser.java

D:\MCA-III\SS File>JAVA RecDecParser
_____________________________
Enter Expression : ab+c


_____________________________
!!! INVALID EXPRESSION !!!


_____________________________

**********************************************************************
**********************************************************************
LL(1) PARSER:-
**********************************************************************
**********************************************************************
import java.io.*;
import java.util.*;

class myLL1{
     public static void main(String args[])throws Exception{
          LL1Parser obj=new LL1Parser();
          obj.initTable();
          obj.showTable();
          obj.getExpression();
          obj.parseExpression();
     }
}
class LL1Parser{
     String expression;
     String ruleArray[][]=new String[6][5];

     void getExpression(){
          Scanner sc=new Scanner(System.in);

     System.out.print("\n_____________________________________________
_____\nEnter Expression : ");
          expression=sc.next();

     System.out.println("\n_____________________________________________
_______");
          expression=expression+"|";
     }

     void initTable()throws Exception{
```

```
          BufferedReader br=new BufferedReader(new
FileReader("ruleTable.txt"));
          String arr[],line=br.readLine();
          int i=0;
          while(line!=null){
               arr=line.split("\t");
               for(int j=0;j<arr.length;j++)
                    ruleArray[i][j]=arr[j];
               i++;
               line=br.readLine();
          }
          br.close();
     }

     void showTable(){
          System.out.println("\n_____________RULE
TABLE_____________________ ");
          for(int i=0;i<ruleArray.length;i++){
               for(int j=0;j<ruleArray[i].length;j++)
                    System.out.print( ruleArray[i][j]+"\t" );
               System.out.print("\n");
          }
     }

     void parseExpression(){
          int SSM=0;
          String csf="",newRule;
          if(expression.charAt(SSM)=='a')
          {

     newRule=getNextRule(ruleArray[1][0],expression.charAt(0)+""
);
               csf=newRule+csf;
               System.out.println("\nCSF \t\t Symbol \t
Prediction
\n_____________________________________________\n");

               while(SSM<expression.length()){
                    System.out.println(csf+" \t\t
"+expression.charAt(SSM) +" \t\t "+newRule);

     newRule=getNextRule(csf.charAt(0)+"",expression.charAt(SSM)
+"");

                    if(newRule==null){
                         System.out.println("!!! INVALID
EXPRESSION !!!");
```

```
                    return;
                }

                csf=new
StringBuilder(csf).deleteCharAt(0).toString();
                csf=newRule+csf;


     if((csf.charAt(0)+"").equals(expression.charAt(SSM)+"")){
                    csf=new
StringBuilder(csf).deleteCharAt(0).toString();
                    SSM++;
                }

                if(newRule.equals("e"))
                    csf=new
StringBuilder(csf).deleteCharAt(0).toString();

                if(csf.equals("")){

     System.out.println("\n_____________________\n EXPRESSION
IS VALID \n_____________________");
                    return;
                }
            }
            System.out.println("!!! INVALID EXPRESSION !!!");
        }
        else{
            System.out.println("!!! INVALID EXPRESSION !!!");
        }
    }
    String getNextRule(String r,String c){
        boolean row=false,col=false;
        int i,j;
        for(i=1;i<ruleArray.length;i++){
            if(ruleArray[i][0].equals(r)){
                row=true;
                break;
            }
        }
        for(j=1;j<ruleArray[0].length;j++){
            if(ruleArray[0][j].equals(c)){
                col=true;
                break;
            }
        }
        if(row && col)
```

```
                return ruleArray[i][j];
           else
                return null;
      }
}
```

```
**********************************************************************
**********************************************************************
 OUTPUT
**********************************************************************
**********************************************************************
D:\MCA-III\SS File>javac myLL1.java

D:\MCA-III\SS File>java myLL1

_____________RULE TABLE____________________
NT      a       +       *       |
E       TF      -       -       -
F       -       +TF     -       e
T       VU      -       -       -
U       -       e       *VU     e
V       a       -       -       -

_____________________________________________
Enter Expression : a+a*a

_____________________________________________

CSF               Symbol          Prediction
_____________________________________________________

TF                a               TF
VUF               a               VU
UF                +               a
F                 +               e
TF                a               +TF
VUF               a               VU
UF                *               a
VUF               a               *VU
UF                |               a
F                 |               e

_____________________________
 EXPRESSION IS VALID
_____________________________

D:\MCA-III\SS File>java myLL1
```

```
_____________RULE TABLE_____________________
NT      a       +       *       |
E       TF      -       -       -
F       -       +TF     -       e
T       VU      -       -       -
U       -       e       *VU     e
V       a       -       -       -

_____________________________________________
Enter Expression : a+a*

_____________________________________________

CSF             Symbol          Prediction
_____________________________________________

TF              a               TF
VUF             a               VU
UF              +               a
F               +               e
TF              a               +TF
VUF             a               VU
UF              *               a
VUF             |               *VU
-UF             |               -
!!! INVALID EXPRESSION !!!

D:\MCA-III\SS File>
```

```
**************************************************************
**************************************************************
OPERATOR PRECEDENCE PARSER:-
**************************************************************
**************************************************************
import java.io.*;
import java.util.*;
class TreeNode{
     String value;
     TreeNode leftChild,rightChild;
     TreeNode(){
          value="";
          leftChild=null;
          rightChild=null;
     }
     TreeNode(String val){
          value=val;
          leftChild=null;
          rightChild=null;
```

```
	}
	TreeNode(String val,TreeNode lchild,TreeNode rchild){
		value=val;
		leftChild=lchild;
		rightChild=rchild;
	}
	void postOrder(){
		if(leftChild!=null)
			leftChild.postOrder();
		if(rightChild!=null)
			rightChild.postOrder();
		System.out.print(value);
	}
}
class Stack{
	String operator;
	TreeNode operand;
	static int TOS =-1;
	void push(String operator,TreeNode operand){
		this.operand=operand;
		this.operator=operator;
	}
}
class OpePreParser{
	static String precedenceTable[][];
	static TreeNode rootNode;

	public static void main(String args[]) throws Exception{
		fillTable();
		showPrecTable();
		Scanner scanner=new Scanner(System.in);

	System.out.println("_____________________________________");
		System.out.print("Enter Expression : ");
		String expression=scanner.next();

	System.out.println("_____________________________________");
		expression="|"+expression+"|";

		int ssm=0,row=0,col=0;
		Stack stack[]=new Stack[10];
		String operator=expression.charAt(ssm++)+"";
		String operand=expression.charAt(ssm)+"";

		TreeNode node=new TreeNode(operand);
		stack[++Stack.TOS]=new Stack();
		stack[Stack.TOS].push(operator,node);
```

```
          String str;
          boolean error=false;

          while(expression.charAt(ssm)!='|'){
               ssm++;
               row=getRow(stack[Stack.TOS].operator);
               col=getColumn(expression.charAt(ssm)+"");
               if(row != -1 && col != -1)
                    str=precedenceTable[row][col];
               else{

     System.out.println("_____________________________________");
                    System.out.println("!!! INVALID EXPRESSION
!!!");

     System.out.println("_____________________________________");
                    error=true;
                    break;
               }
               if(str.equals("<")){
                    operator=expression.charAt(ssm++)+"";
                    operand=expression.charAt(ssm)+"";
                    node=new TreeNode(operand);
                    stack[++Stack.TOS]=new Stack();
                    stack[Stack.TOS].push(operator,node);
               }
               else if(str.equals(">")){
                    pop(stack);
                    ssm--;
               }
          }
          if(!error){
               rootNode=stack[Stack.TOS].operand;

     System.out.println("_____________________________________");
               System.out.print(">>Post Order Expression : ");
               rootNode.postOrder();

     System.out.println("\n_____________________________________");
          }
     }
     static void pop(Stack stack[]){
          TreeNode node1=stack[Stack.TOS].operand;
          String op1=stack[Stack.TOS--].operator;
          TreeNode node2=stack[Stack.TOS].operand;
          String op2=stack[Stack.TOS--].operator;
```

```
          TreeNode node=new TreeNode(op1,node2,node1);
          stack[++Stack.TOS].push(op2,node);
     }
     static void fillTable() throws Exception{
          BufferedReader bufRead=new BufferedReader(new
FileReader("OpePreTable.txt"));
          precedenceTable=new String[6][6];
          String arr[],line=bufRead.readLine();
          int i=0;
          while(line!=null){
               arr=line.split("\t");
               for(int j=0;j<arr.length;j++)
                    precedenceTable[i][j]=arr[j];
               i++;
               line=bufRead.readLine();
          }
          bufRead.close();
     }
     static void showPrecTable(){

     System.out.println("\n_____________________Precedence
Table_____________________________");
          for(int i=0;i<precedenceTable.length;i++){
               for(int j=0;j<precedenceTable[i].length;j++)

     System.out.print("\t"+precedenceTable[i][j]);
               System.out.println("");
          }
     }
     static int getRow(String str){
          for(int i=0;i<precedenceTable.length;i++)
               if(precedenceTable[i][0].equals(str))
                    return i;
          return -1;
     }
     static int getColumn(String str){
          for(int i=0;i<precedenceTable[0].length;i++)
               if(precedenceTable[0][i].equals(str))
                    return i;
          return -1;
     }
}
**************************************************************
**************************************************************
OUTPUT
**************************************************************
**************************************************************
```

```
D:\MCA-III\SS File>javac OpePreParser.java

D:\MCA-III\SS File>java OpePreParser

_____________________Precedence
Table_____________________________
        op      +       *       (       )       |
        +       >       <       <       >       >
        *       >       >       <       >       >
        (       <       <       <       =       $
        )       >       >       $       >       >
        |       <       <       <       $       =

_____________________________________
Enter Expression : a+b*c

_____________________________________
_____________________________________
>>Post Order Expression : abc*+

_____________________________________

D:\MCA-III\SS File>java OpePreParser

_____________________Precedence Table_____________________________
       op      +       *       (       )       |
       +       >       <       <       >       >
       *       >       >       <       >       >
       (       <       <       <       =       $
       )       >       >       $       >       >
       |       <       <       <       $       =

_____________________________________
Enter Expression : a*bc

_____________________________________
_____________________________________
!!! INVALID EXPRESSION !!!

_____________________________________

D:\MCA-III\SS File>
**********************************************************************
**********************************************************************
```