Page No.:
Date: / /

Paper

Q.1. explain different kind of system software.

- program development environment
- Run-time environment

☆ program development environment

- text editor: software that permits the creation and editing of text files.
- compiler: translate Program written in a high level language to object code or machine code
- Assembler: translates programs written in assembly language to object code or machine code
- static linker: combined and resolve reference between object code
- Debugger: It is used to debug executable program and their related object code and source code

☆ Run-time environment

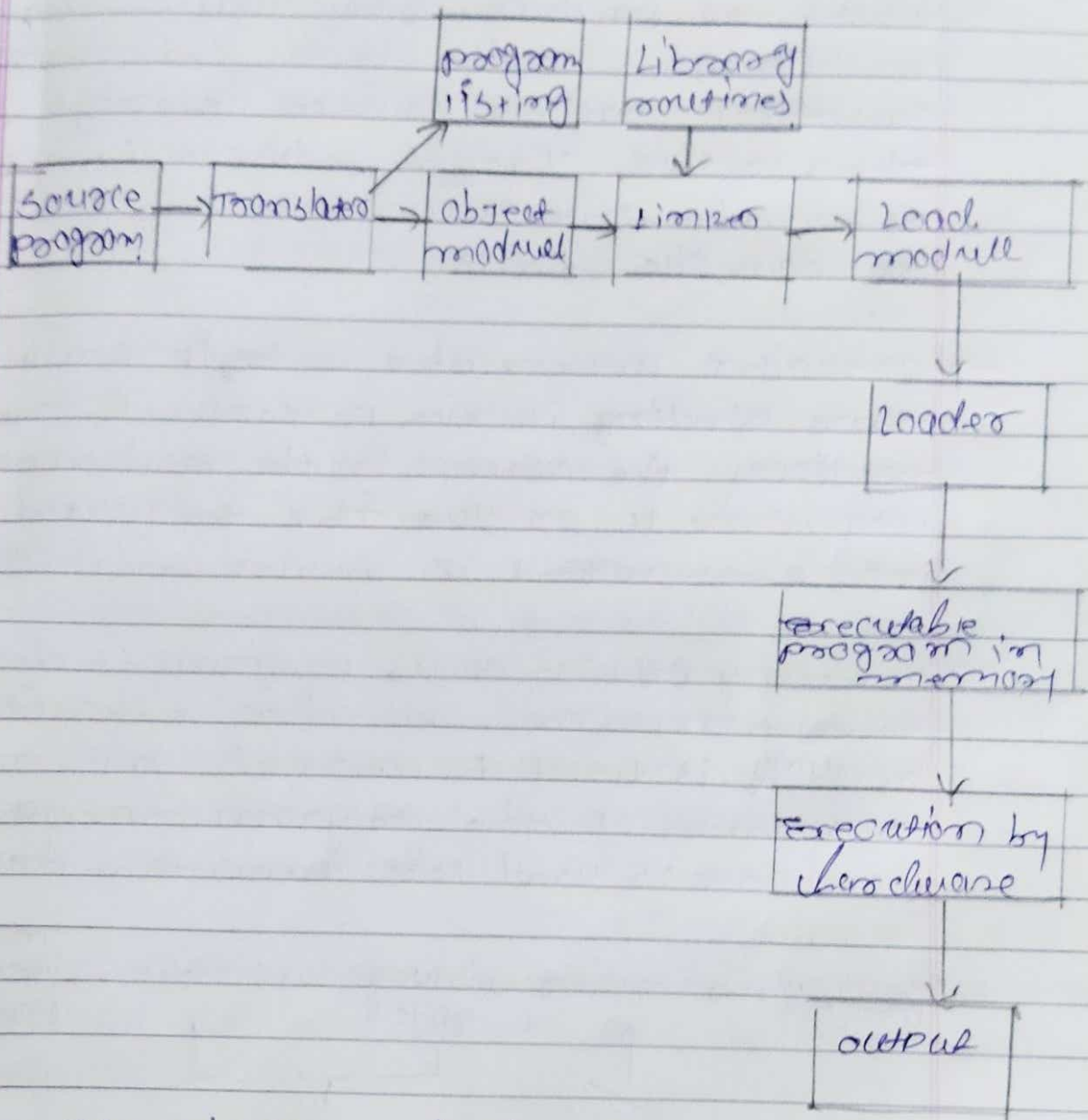
- loader: load an executable code and starts it execution
- libraries: precompiled program that create a set of functions for use by other program
- dynamic linker: load and links shared libraries at runtime
- operating system: An event driven program that make an abstraction of the computer for application system. The operating system handle bys all resources efficiently

create an environment for application program to run and provides a friendly interface between the user and the computer system

b] Compare problem oriented and procedure oriented language

Problem oriented	procedure oriental language
→ source language is a problem oriented.	→ A procedure oriental program language is the target language.
→ Execution gap is larg because of problem oriented language	→ specification gap is large because of procedure oriented language
→ problem oriented language is also known as Fourth generation language	→ Procedure oriented language is also known as third generation languages
→ Ex: Query language and application generat	→ ex: COBOL, FORTRAN, Pascal, Ada & HTML
→ problem oriented language were designed to solve problems.	→ Procedural language are intended to solve general problems.

6 Explain life cycle of source program with neat sketch



→ The assembler read the source code program in Assembly language and generates the object program in binary form. The object program is passed to the linker

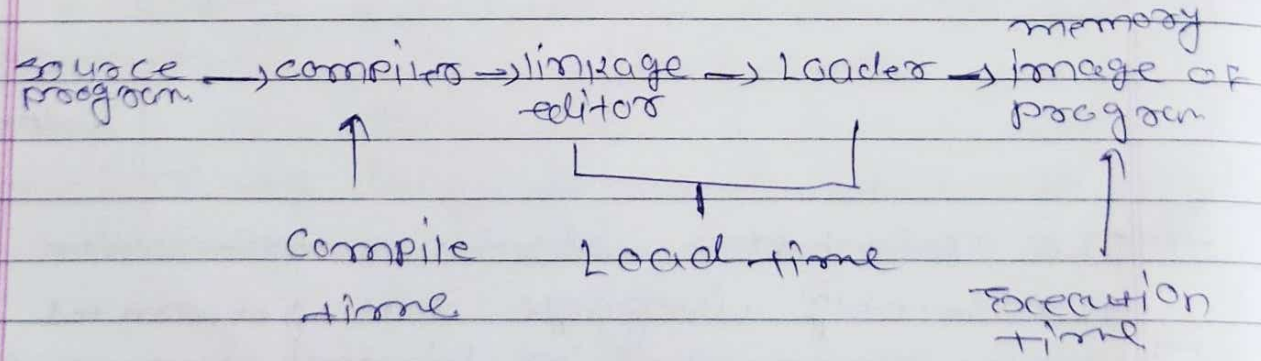
→ The linker will be combined the required procedure from the linker library with the object program and produce the executable program. The loader load the executable program into memory and branches the CPU

to the starting program.

→ source program is translate at Compile time to produce a relocatable object module. At compile time, the translator generates code to allocate storage for the variable. Target address is unknown at compile time, it cannot be found at compile time.

→ compiler generatable code if compile time binding is not performed. The loader modifies the address in the load module at load time to produce the executable image stored in main memory.

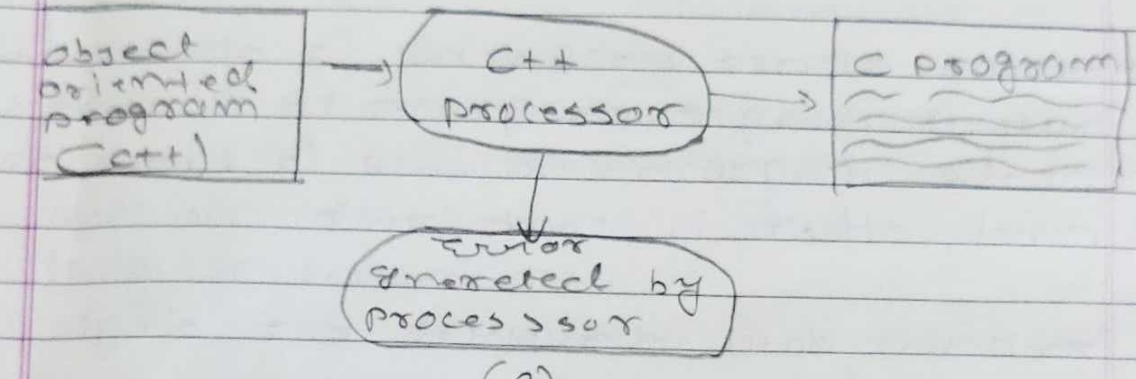
→ memory address of the program is changed at execution time, and then execution time binding is used is delayed until the run time of the program. special hardware is used for execution time binding.



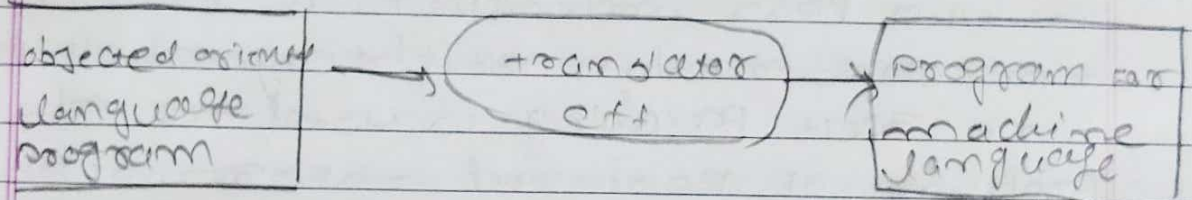
→ memory allocation and deallocation is done using run-time support of the programming language in which a program is coded. Allocation and deallocation request coded by calling appropriate routines of the run time library.

Page No. :
Date : / /

Q.2 a Give example of language processor.



(a)



→ source program is input to a language Processor and target program is output language Processor. language used for writing a source program and target program is called language and target language respectively.

→ In a computer system, language translator bridges an execution gap to the machine language

→ A translator bridges an Execution gap from the machine language of a system

→ processor also bridge an execution gap, but in is not a language translator.

→ language migrator bridges the specification gap between two programming languages

→ Each language Processor uses its own set of command

→ Input source program is same but output target program is different in the diagram. So one is processor and other is translator.

b) Explain data structure for single pass assembler.

→ In two pass translator, PC processing and construction of the symbol table done. Back Patching is used to the problem of Forward reference. The operand field of an instruction containing a forward reference is left blank initially. The address of the forward reference symbol is put into this field when its definition is encountered for ex

MOVER BREG, ONE

In the above statement, ONE is a Forward reference. The memory location (LC) contains the instruction opcode and address of BREG. Table of Incomplete Instruction (TII) is used inserting the second operand address. TII contains the (LC instruction address, <symbol>)

→ When assembler reads END statement the symbol table would contain the addresses of all symbols defined in the source program and table of

incomplete instruction would contain information describing all forward references. The assembler can now process each entry in TII to complete the concerned instruction

C Explain symbol table data structure for language processing

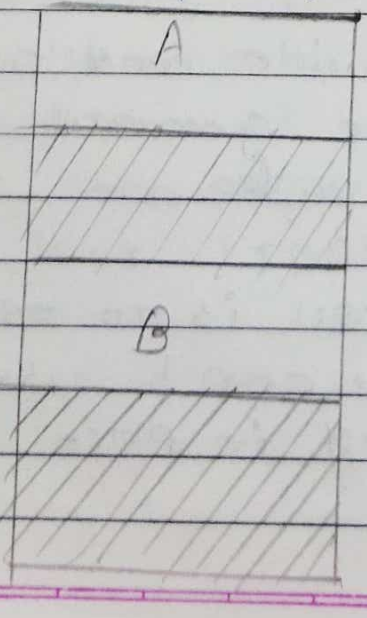
→ Data structures are classified on the basis of the following criteria

* Linear or nonlinear data structure.

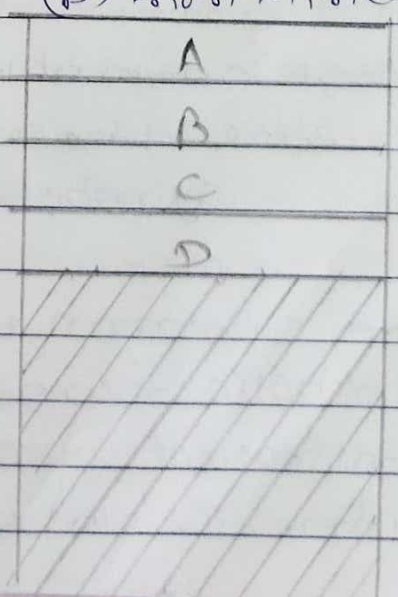
→ A linear data structure consists of a linear arrangement of element in the memory for its elements. This leads wastage memory.

→ The element of a non linear data structure are accessed using pointers The elements need to occupy contiguous area of memory. There is no wastage of memory. But it leads to lower search efficiency.

(a) Linear



(b) non-linear



• → Search data structure are used during language processing to maintain various information concerning different entities in the source program.

→ This type of data structures are characterized by using entry for an entity is created only once but may be searched for a large number of times

• Allocation data structures are characterized by the nature of the memory area allocation to an entity is known to the user of that entity. In this method, search operation are not conducted. The important points are allocation or deallocation speed and efficiency of memory utilization in this type of data structures.

2. Explain in detail any two advanced assembler Directive

★ EQU

→ most assembler provide an assembler directive the allows the general form

symbol EQU value

→ one common use of EQU is to establish symbolic names that can be used for improved readability in place of numeric values

→ Another common use of EQU is in defining mnemonic names for registers, example:

```
A   EQU  0
X   EQU  1
L   EQU  2
```

* LTORG.

→ LTORG allow placing literals into a pool at some other location in the object program

→ Directive LTORG creates a literal pool that contains all of the literal operands used since the previous LTORG or the beginning of the program

→ Literals placed in a pool by LTORG will not be repeated in the end of the program

→ The LTRG statement permits a programmer to specify where literals should be placed. By default, assembler place the literals after the END statement.

→ Assembler allocates memory to the literals of a pool. The pool contains all literals used in the program since the start of the program or since the last LTORG statement.

Q.3. A Define macro, macro expansion with syntax of macro.

☆ macro expansion:

macro expansion is a macro name with a set of formal parameters is replaced by some code

☆ a) macro expansion:

a) macro call lead to macro expansion.

b) macro call statement is replaced by a sequence of assembly statement in macro expansion

c) Two key motions are used in macro expansion

a) expansion time control flow
b) lexical substitution

d) Algorithm for macro expansion

1.] Initialize the macro expansion counter
2.] Check the statement which is pointed by MEC is not a MEND statement.

a) if the statment is model statement then

expand the statment and increment the MEC by 1

else

MEC := new value specified in the statment;

b] positional parameters, keyword parameters and default value parameter for macro

→ If the parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement, then these parameters in macro definitions are called as positional parameters.

- Syntax : written as,
  & <parameter name>
  Example : & ROLLNO.

where ROLLNO is the name of a parameter in positional parameter
<parameter kind> is omitted. The <actual parameter spec> is a call on a macro using positional parameter is simply an <ordinary string>

A keyword Parameters.
→ For keyword Parameters,
<parameter name> ⇒ ordinary string
<parameter kind> ⇒ string '='
<actual Parameter spec> ⇒ written as
<Formal parameter name> = <ordinary string>

→ Value of Formal Parameter is determined as follows.

1. Find the actual parameter specification which looks like XYZ = <ordinary string>
2. <ordinary string> is in the specification of the string ABC. XYZ = ABC

b] Positional parameters, keyword parameters and default value parameter for macro

→ If the parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement, then these parameters in macro definitions are called as positional parameters.

- Syntax : written as,
  & <parameter name>
  Example : & ROLLNO.

where ROLLNO is the name of a parameter in positional parameter
<parameter kind> is omitted. The <actual parameter spec> is a call on a macro using positional parameter is simply an <ordinary string>

A Keyword Parameters.

→ For keyword Parameters,
<parameter name> ⇒ ordinary string
<parameter kind> ⇒ string '='
<actual parameter spec> ⇒ written as
<Formal parameter name> = <ordinary string>

→ Value of Formal Parameter is determined as follows.

1. Find the actual parameter specification which looks like XYZ = <ordinary string>
2. <ordinary string> is in the specification of the string ABC. XYZ = ABC

Page No. :
Date : / /

* Default specification of parameters :

→ Standard assumption value is a default value. This is suitable in such condition the parameter has the same value in most cases. when the described value is different from the default value the desired value can be specified explicitly in a macro call. Syntax for for formal parameter specification is as follows

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

c) explain relocation program with example

→ program relocation is the process of changing the addresses used in the address sensitive instruction of a program so that the program can execute correctly from designed area of memory

→ The relocation factor can be calculated using following formula.

Relocation factor = Load origin - Translated origin

The translator put the address of particular symbol which is used in the program. This is called translation time address

Translated time address = Translated origin + OFFset OF a

similarly the linked time address can be denoted as.

Linked time address = Linked origin + OFFset OF A

another way of denoting linked time address is

Linked time address = Translated time address + Relocation Factor.

* Example of Relocation

1. suppose there is a command linker 500 A,3 the linker command 500 denotes load origin
2. Let for the linker set of instructions X that need to be relocated the relocation table is as Follow.

Translated Address
400
438

3. The instruction with translated address 400 containing the address 430 ~~the~~ 330 in operand Field

4. The instruction with translated address 438 contain the address 302 in operand Field translated origin Factor

5. Calculate relocation factor

Relocation Factor = Load origin - translated origin

= 800 - 500.

Relocation Factor = 300

OR

Q.3. A Explain Pass-2 data base of macro

1. The copy of the input macro source deck
2. output expand source deck to be used as input to the assembler.
3. MDT is created by Pass I
4. MDT is created by pass I
5. MDTP is used to indicate the next line of text to be used during macro expansion
6. ALSA is used to ~~indicate~~ substitute macro call argument for the index numbers, in the stored macro definition

b) Illustrate expansion of nested macro call by giving example

→ macro may call another macro in nested macro call. called macro is called as inner macro whereas macro containing the nested call as a outer macro

→ Expansion of nested macro calls follows the last in first out rule.

For example

```
MOVEM    BREG, TMP
MOVER    BREG, P
ADD      BREG, P
MOVEM    BREG, P
MOVER    BREG, TMP
```

```
MACRO
COMPUTE  &FIRST &SECOND
MOVEM    BREG, TMP
INCR_D   &FIRST, &SECOND, REG=BREG
MOVER    BREG, TMP
MEND
```

- In the above program the macro COMPUTE contains a nested call on macro INCR_D. The Expanded Code for the call

COMPUTE P,Q

```
COMPUTE P,Q   1. MOVEM BREG, TEMP [1]
              2. INCR_D P, Q
              3. MOVER BREG, TMP [5]

              MOVER BREG, P [2]
              ADD   BREG, Q [3]
              MOVEM BREG, P [4]
```

→ Expansion is performed after the lexical expansion and leads to the generation of marked statement 2, 3 and 4. Then the third model statement of COMPUTE is expanded.

c) What are advanced macro programming facilities ? explain with examples

→ 1. For alteration of flow of control during expansion
2. Expansion time variable
3. Attributes of parameter.

★ AIF (<expression>) <sequencing symbol>.

where <expression> = Relational expression involving ordinary stuff

→ Relational expression is true then only control is transferred to the statement containing <sequencing symbol> in it label field

→ • An ANOP statement is written as.
<sequencing symbol> ANOP
and simply has the effect of defining the sequencing symbol.

★ expansion time variable (EV)

→ These variables are used only during the expansion of macro call. Expansion time variables are local or global. Local EV is created for use only during a particular macro call. Syntax of local and global EV is as follows:

LCL <EV specification> [, EV specification]...
GBL <EV specification> [, EV specification]...
<EV specification> has the syntax & <EV name>

but <EV name> is an ordinary string. SET statement is used to manipulate the value of EV.
<EV specification> SET <SET expression>
where

<EV specification> SET <SET Expression> only in the label field. SET is in the mnemonic field

* Attributes of Formal parameter syntax

→ <attribute name> <Formal parameter spec> it also represents information about the value of the Formal parameter. The TYPE length, and size attributes have the names T, L and S.

1 Expansion Time LOOPS
→ Expansion time LOOPS can be written using Expansion Time variable and expansion time control transfer statements AIF and AGO
→ Example

```
MACRO
   CLEAR    &A
   MOVEM    AREG, ='0'
   MOVEM    AREG, &A
   MOVEM    AREG, &A + 1
   MEND     AREG, &A + 2
```

→ CLEAR B is called, the MOVEM statement puts the value '0' in AREG. The remaining three MOVEM statements store this value in three consecutive bytes with the address B, B+1, B+2.

Page No. :
Date : / /

Q.4 A. what is loader? enlist basics Functions of loader.

→ loader is utility program which TAKES object code on input prepares it FO execution and loads the executable code into the memory. This loader is basically responsible FOR initiating the execution process.

☆ Functions of loader

→ It ~~responsible~~ allocates the space FOR the activities such as allocation, linking, relocation and loading

1 It allocates the space FOR program in the memory, by calculating the size OF the program. This activity is called allocation.

2 It resolved the symbolic references between the object modules by assigning all the user subroutine and library subroutine address. This activity is called linking.

3 There are some address dependent location in the program, such such constant must be adjusted according to allocated space, such activity done by loader is called loading.

→ Finally it places all the machine instruction and data of corresponding program and

subroutines in the memory. This program now becomes ready for execution This Activity is called loading.

b Explain term of self relocating program.

→ non Relocatable program is a program that can be executed from the area starting from its translated origin

→ Relocatable Program is program which executes from the memory location designated by relocating information rather than be the location specified at translated time. The relocatable program in the form of object module and a linking editor or relocating loader loads the program at appropriate location by using relocation information. The relocating information accurately identifies the address sensitive instruction in the code and then with the help of this information the relocation is made

→ The self relocatable program is a program is a program which itself performs relocation. It does no require linkage editor to Perform relocation. The self relocation program cannot load itself but relocated itself. The self relocation program does not bound to a specific memory area of its execution. For preparing a self according from of a program special techniques are needed

Page No. :
Date : / /

c Explain Linking or overlay structured Program

→ During execution of the program, sometimes it is not necessary to load all the parts of the program

→ The memory requirment of such program can be reduced by loading during linking those parts in the memory that are required

→ Hence some part of the program can be given same load address during linking. This way only one of these part can be in the memory at any time.

→ when other other part has to be loaded Then it is given with the same address so it will overwrite the previous part of execute. Due to this arrangment the total memory requirment for loading the program segments get reduced to much extent

→ Definition of overlay structure: The Program containing multipe overlay is called overlay structured program

→ There might be some part of the program they need to be loaded

permanently in the memory for execution of other part of the program such a permanent residing part of the program is called root other parts of the program can be loaded in the memory as per the requirment.

→ The program which controls the loading of the overlays when required is called overlay manager. This overlay manager is limked with the root.

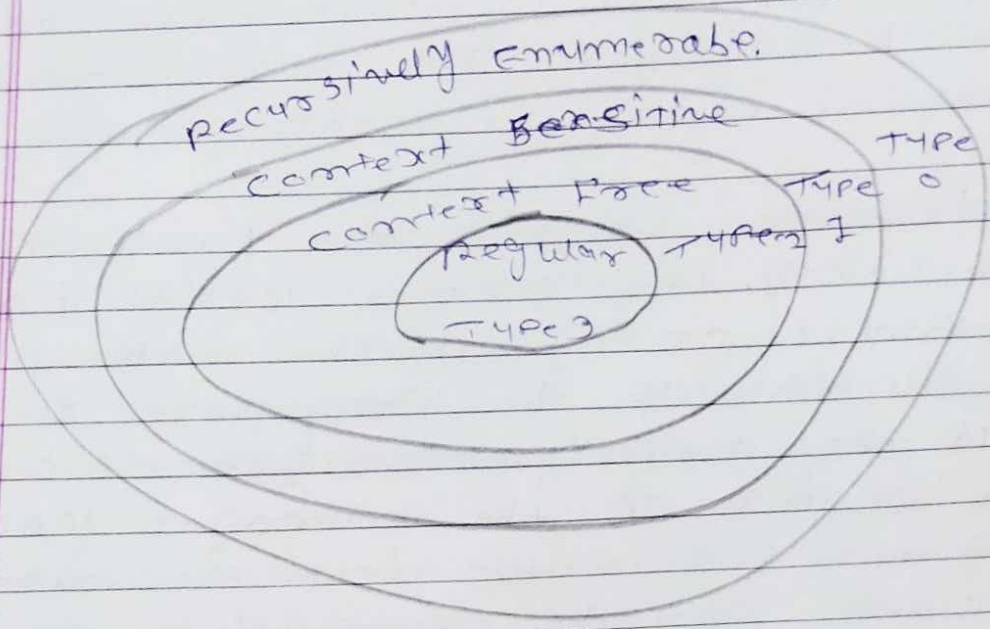
→ At the initial stage of the program execution, the root is loaded in the memory and given the control for execution of the program. It then calls the overlay manager to control the loading of the manager loads the program segments that are currently required. For the execution. It would overwrite the previously overloaded overlay using the same load begin

→ It is clear from the figure that same part of program can reside in the memory simultaneously. Thus the total memory allocated is 7k with overlay structure whereas the memory requirment of entire program is 13k. Thus lot of space can be saved using overlay structuring.

OR

Q.4.A List and explain types of grammar.

- Recursively Enumerable (Type 0)
- context Sensitive (Type 1)
- context Free (Type 2)
- Regular (Type 3)



Any Type 0 Grammar

- This Type of grammar is of the form

$\alpha \rightarrow \beta$

where α and β can be string of terminal and non-terminal symbol

It is called phrase structure grammar. This form of grammar allow arbitary substitution of strings during derivation or reduction

Page No. :
Date : / /

Type 1 Grammar

→ In this type of grammar the derivation or reduction the strings have place only specific context, This type of grammar is called context sensitive grammar. The form of this grammar is

$$\alpha A \beta \rightarrow \alpha \pi \beta$$

→ This shows that the string π can be replace by A if it is enclosed within α and β.

Type 2 Grammar:

→ This type of grammar in which there is no context requirment on derivation or reduction

This form of grammar is,

$$S \rightarrow \pi$$

This type of grammar is called Context Free grammer

Type 3 Grammar.

This type of grammar is represented by the following form

$$A \rightarrow tB \mid t$$

or

$$A \rightarrow Bt \mid t$$

b) Define simple phrase and handle. Using handle and simple phrase trace the bottom up parsing algorithm
Grammar is: E; E → T + E | T - E | T
T → T * V | T / V | V
V → a | b | c | d
string is b * c + d

Solution Tracing of a a-b*c+d

Stack	Input BUFFER	Parsing Action
$	a-b*c+d$	shift
$a	-b*c+d$	Reduce V→a
$V	-b*c+d$	Reduce T→V
$T	-b*c+d$	shift
$T-	*c+d$	shift
$T-b	*c+d$	Reduce by V→b
$T-V	*c+d$	Reduce T→V
$T-T	+d$	shift
$T-T*	+d$	shift
$T-T*c	+d$	Reduce by V→c
$T-T*V	d$	Reduce by T→T*V
$T-T	$	shift
$T-T+	$	Shift
$T-T+d	$	Reduce by V→d
$T-T+V	$	Reduce by T→V
$T-T+T	$	Reduce by E→T
$T-T+E	$	Reduce by E→T+E
$T-E	$	Reduce by E→T-E
$E	$	Accept

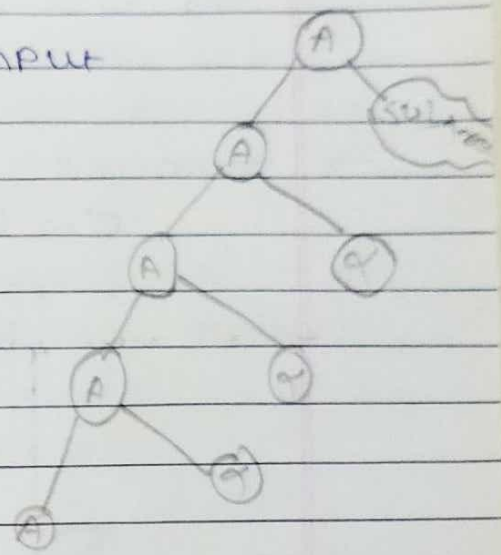
Page No. :
Date : / /

Q. Explain left recursion and ~~left~~ left factoring with example

→ The left recursive grammar is a grammar which is as given

$A \xrightarrow{+} A\alpha$

→ The $\xrightarrow{+}$ means deriving the input in one or more steps. The A can be non terminal and α denotes some input string. If left recursion is present in the grammar then it creates serious problem. Because of left recursion the top-down parser can enter in Finite loop. This is as shown in Fig.



→ Thus, expansion of A causes Further expansion of A only and due to generating of A, $A\alpha$, $A\alpha\alpha$, $A\alpha\alpha\alpha$ ... The input will no be advanced. This causes major problem in top-down parsing and therefore elimination of left recursion is a must

→ To eliminate left recursion we need To modify the programmar. Let G_1 be a Context Free grammar having a production rule with left recursion

$A \to A\alpha$ |
$A \to B$. |

Then we eliminate left recursion by re-writting the production rules

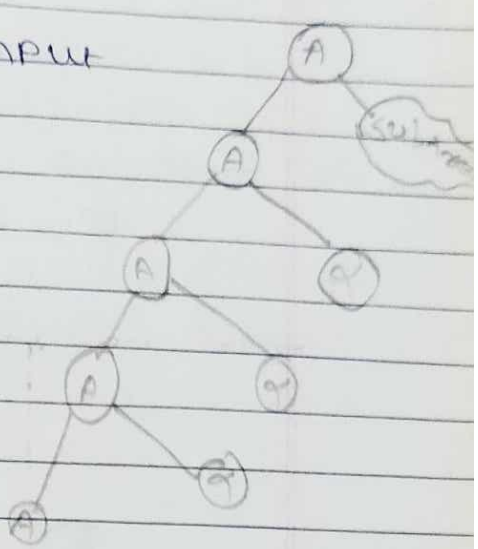
Page No. :
Date : / /

Q]. Explain left recursion and ~~left factoring~~ with example

→ The left recursive grammar is a grammar which is as given

$A \xrightarrow{+} A\alpha$

→ The $\xrightarrow{+}$ means deriving the input in one or more steps. The A can be non terminal and α denotes some input string. If left recursion is present in the grammar then it creates serious problem. Because of left recursion top-down parser can enter in Finite loop. This is as shown in Fig.



→ Thus, expansion of A causes Further expansion of A only and due to generation of A, $A\alpha$, $A\alpha\alpha$, $A\alpha\alpha\alpha$ ... The input will no be advanced. This causes major problem in top-down parsing and therefore elimination of left recursion is a must

→ To eliminate left recursion we need to modify the programmar. Let, G be a Context Free grammar having a production rule with left recursion

$A \rightarrow A\alpha$ |
$A \rightarrow B.$ |.

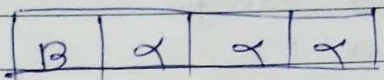
Then we eliminate left recursion by re-writting the production rule as

$A \rightarrow \beta A'$
$A' \rightarrow \alpha A'$
$A' \rightarrow \varepsilon$

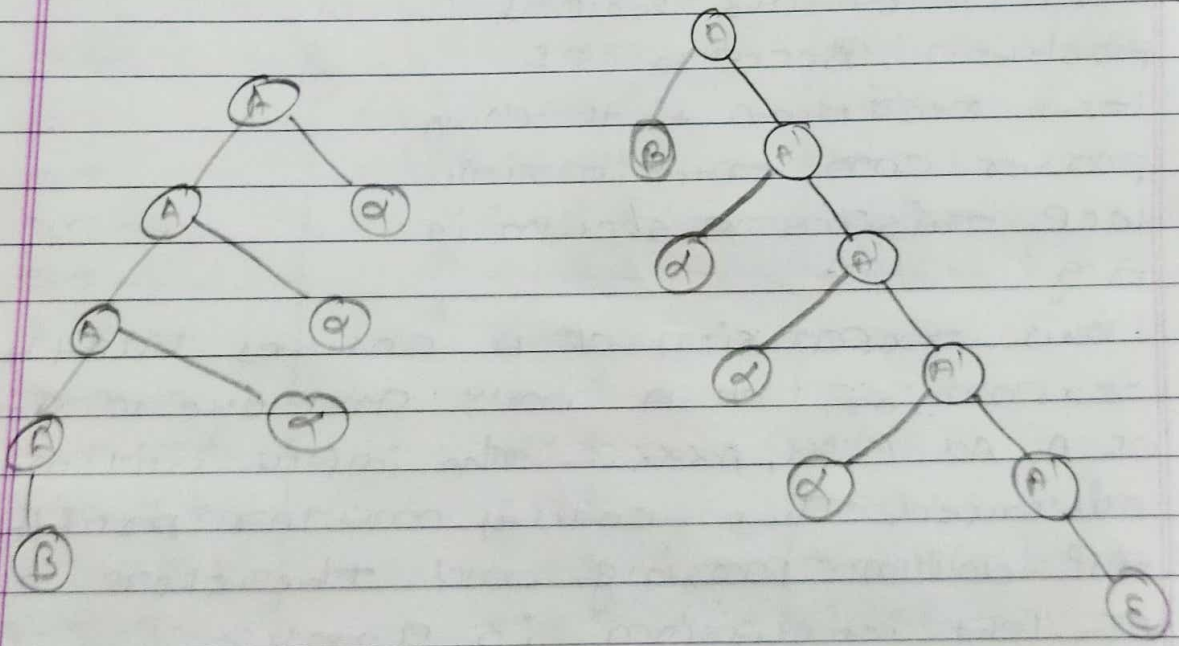
→ Thus a new symbol A' is introduced we can also verify, whether the modified grammar is equivalent to original or not

→ Example : consider the grammar

$E \rightarrow E+T \mid T$



Input string



Page No. :
Date : / /

Q.5.A) Explain the terms: Binding and binding times

→ These are various entities that occur in the program. These entities can be identifier, procedures, keywords and so on.

→ A variable has attributes such as TYPE, dimensionality, SCOPE, memory address and so on. These attributes can be assigned with some value

→ Binding can be defined as an association of attributes of program entity with some value

For example

```
int i;
```

here variable i is associated with the type int.

* Binding time
→ Binding time is the time at which the binding is actually performed
→ compiler had following binding times

→ why there is a need for Binding time

→ The binding time is determines the manner in which the compiler handles particular entity in the program. Sometimes TO code can be generated and it is tailored TO the programmin

entity. one to which the execution and access to that entity can be done efficiency.

b] write short note on - LEX

→ LEX or Flex is UNIX utility which can be used for generating the lexical analyzer

→ LEX make use of regular expression for recognizing the pattern of the token

→ The LEX specification file can be created using the file name extension .l (dot l)

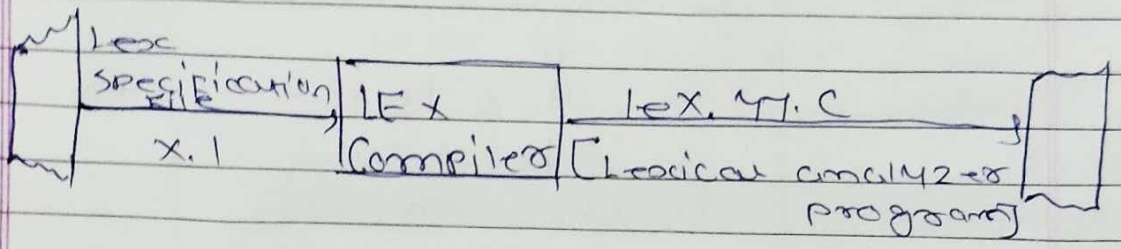
→ The lex.yy.c consists of tabular representation of the transition diagrams constructed for the regular expression of specification file. The lexeme can be recognized with the help of tabular transition diagram and some standard routines

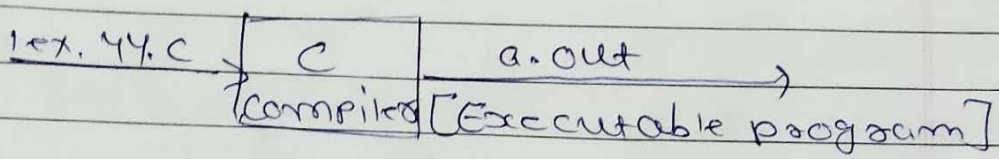
→ In specification file of LEX action each regular expression has specific action associated with it. These action are simply piece of C code

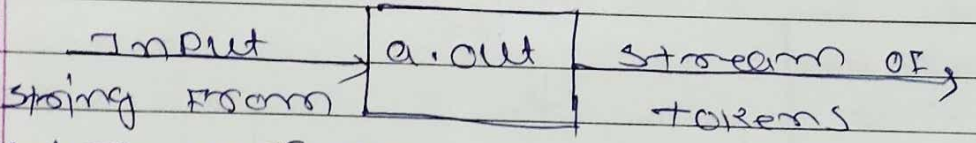
→ Finally C compiler compiles the generated lex.yy.c file and an output file say a.out is produced.

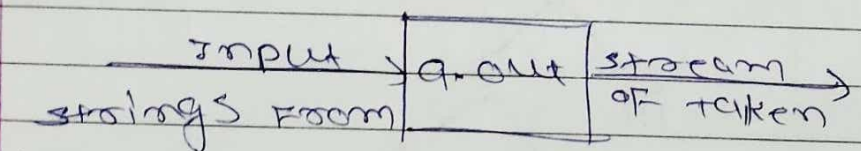
→ The output can be tested using some programming statements we should get corresponding token for these statement.

→ The above described scenario can be represented by following fig









c] show how an input a=b+c*60 get processed in compiler. show the output at each stage of compiler. Also show the contents of symbol table

Page No.
Date : / /

Input processing in compiler

$a = b + c * 60$

↓

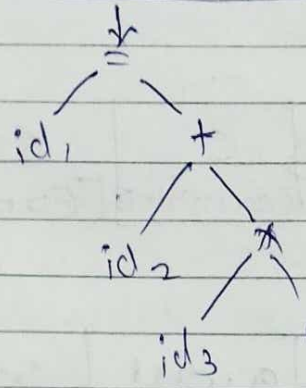
[Lexical analyzer]

↓

$id_1 = id_2 + id_3 * 60$ — Token stream

↓

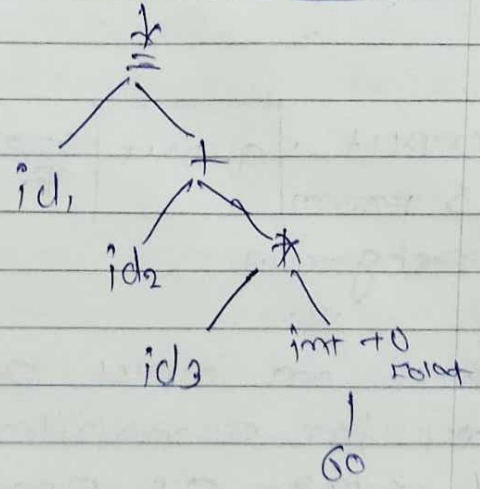
[Syntax analyzer]



Syntax tree

↓

[Semantic analyzer]



Semantic gap

↓

[Intermidiate code generator]

a	----	t_1 = int to float (60)
b	----	$t_2 = id_3 * t_1$
c	----	$t_3 = id_2 + t_2$
:	----	$id_1 = t_3$

Intermidiate code

↓

[code optimizer]

$t_1 := id_2 * 60.0$ optimize code
$id_1 := id_2 + t_1$

↓

[code generator]

↓

```
MOVF  id3, R2
MULF  #60.0, R2      machine
MOVF  id2, R1          code
ADDF  R2, R1
MOVF  R1, id1
```

OR

Q.5 A How compiler implements scope rules?

→ The block is sequence of statement containing the local data declaration and enclosed within the delimiters.

For Example

```
{
  Declaration statements
  ...
}
```

→ The determine marks the beginning and end of the blocks. The blocks can be in nesting Function that means block B2 completely can be inside the block B1.

→ The scope of declaration in block structured language is given by most closely nested loop or static rule. It is as given below

ii) The declaration are visible at a procedure

i The declarations that are made locally in the procedure

ii The names of all enclosing procedures

iii the declaration of names made immediately within such procedures

local variable: A variable declared in a block b is called local variable of block.

non local variable: A variable of an enclosing block that is accessible within block b is called non local variable of block b.

b) what are the facilities for dynamic debugging.

→ dynamic debugging: dynamic debugging is either software or hardware debugger

→ An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program

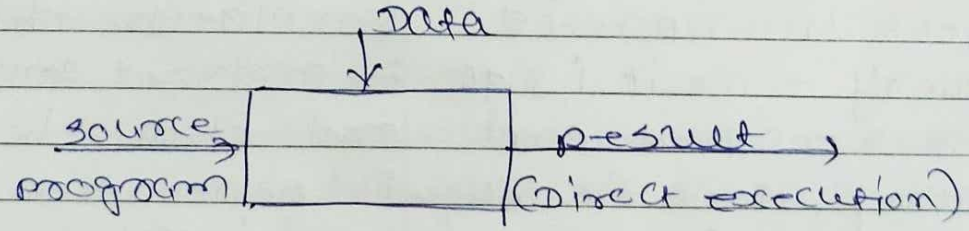
→ This is the most common method of debugging but is the least efficient method. In this method the program is loaded with print statements to print the intermediate values with the hope

hope that some of the printed values will help to identify the statement in error. This approach becomes more systematic with the use of a symbolic debugger

~~Features of~~

C] what is interpreter ? explain benefits of interpreter. Compare interpreter and compiler.

→ An interpreter is kind of translator which produces the result directly when the source language and data is given to it as input. It does not produce the object code rather each time program the needs execution. The model of Interpreter is here



Benefits of Interpreter

1. The interpreter does not generate a target program. Hence it is simple to develop interpreter for a programming language than to develop a compiler. The simplicity in design of interpreter is due to the fact that it does not generate the code.

2. For the languages or programs in which the commands are not executed repeatedly interpreter is supposed to be a good choice.

3. modification of user program can be easily made and implemented as execution proceeds.

4. Debugging a program and finding errors is simplified task for a program used for interpretation.

5 The interpreter for the language makes it machine independent.

Interpreter	Compiler
1. Demerit: The source program gets interpreted every time it is to be executed and every time the source program is analyzed, Hence interpretation is less efficient than Compiler.	Merit: In the process of compilation the program is analyzed only once and then the code is generated. Hence compiler is efficient than interpreter.
2. The interpreters do not produce object code	→ The compiler produce object code
3 Merit: The interpreters can be made portal because they do not produce object code	Demerit: The compiler has to be present on the host machine when particular program need to be compile

4	Merit: Interpreters are simpler and give us improved debugging environment	Demerit: The compiler is a complex program and it requires large amount of memory.
5	An interpreter is a kind of translator which produces the results directly when the source program/langug and data is given to it as input	An compiler is a kind of translator which takes only source program as input and converts it into object code. The loader performs. An executable code compiler takes this executable code and data as input and produces output
→	Example: A UPS ~~debugger~~ debugger is basically a graphical source level debugger but it contains built-in c interpreter which can handle multiple source files.	Example: Borland c compiler or Turbo C compiler compiles the program written in C or C++