Enroll. No._________

MARWADI UNIVERSITY
MU-FOT
CE-FOT1 (MU), IT-FOT1 (MU)
Semester 6 - Summer

Subject : COMPILER DESIGN (01CE0601)

Date : 26-Apr-2022 **Time : 3 Hours** **Total Marks : 100**

Instructions :

1. **Attempt all questions.**
2. **Make suitable assumptions wherever necessary.**
3. **Figures to the right indicate full marks.**

Que.1 (A) Answer the following objectives **[10]**

(1) Which one is follow Top down Parsing methodolgy?
a) SLR
b) Predictive Parsing
c) Operator Precedence Parsing
d) Shift Reduce parsing

Ans. b) Predictive Parsing

(2) Which information are stored in symbol table?
(a) Variable names
(b) Function names
(c) Label Names
(d) All options are correct

Ans. (d) All options are correct

(3) Which of the following depends on target machine and do not depend on source program?
(a) Front End
(b) Back End
(c) Both
(d) None of these

Ans. (b) Back End

(4) Common Sub Expression elimination is one of the code ____________ technique.
a) Optimization
b) Debugger
c) Generation
d) a and b both

Ans. a) Optimization

(5) Input for Code Generation is ______
a) Assembly Language

b) Higher Level Language
c) Binary language
d) Optimized Intermediate Code

Ans. d) Optimized Intermediate Code

(6) Transition function (δ) of DFA machine maps
(a) Q x ∑ → ∑
(b) Q x Q → ∑
(c) ∑x ∑ → ∑
(d) Q x ∑ → Q

Ans. (d) Q x ∑ → Q

(7) Identify the task carried out by Lexical analyzer.
(a) read the input charater and produces set of tokens
(b) Ignore space, tab
(c) Ignore comments
(d) All of the Above

Ans. (d) All of the Above

(8) Finite Automata M= (Q, ∑, q0, A, δ). In which ∑ stands for________
(a) Starting State
(b) Final State
(c) Input Alphabet
(d) Transition function

Ans. (c) Input Alphabet

(9) In triple, we have three fields named as ______
a) Operand, arg1, arg2
b) Operator, arg1, arg2
c) arg1, arg2, arg3
d) result, arg1, arg2

Ans. b) Operator, arg1, arg2

(10) From the following options, which one is Left Recursive Grammar?
a) M→js | b
b) M→Ma | b
c) M→ia | nK
K→m
d) M→a | b | nL
L→n

Ans. b) M→Ma | b

Que.1 (B) Answer the following questions. **[10]**

(1) State quadruple

Ans. A quadruple is a record structure with four fields, which we call op, arg1, arg2 and result.

(2) Write down any two source language issues.

Ans. Procedure call
Activation tree
Control stack
scope of declaration
Bindings of names

(3) Enlist the storage allocation strategies.

Ans. Static Allocation
Stack Allocation
Heap Allocation

(4) What is the meaning of Shift action in Shift Reduce Parser?

Ans. Can Shift any terminal or Non Terminal from buffer to the stack for the parsing purpose

(5) What is Compiler?

Ans. Compiler is a program that converts program written in high-level language into machine code understood by the computer.
It can compile Whole code at a time.

(6) Write down the general formula to remove the left recursion from the grammar.

Ans. $A \rightarrow A\alpha \,/\, \beta$

$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \,/\, \in$

(7) List down the main four operations of Shift Reduce Parser.

Ans. Shift
Reduce
Accept
Error

(8) What is Augmented Grammar?

Ans. If G is a grammar with start symbol S then G', the augmented grammar for G, is the grammar with new start symbol S' and a production S' -> S. The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of input.
Example : Let a grammar be S -> AA
A -> aA | b
The augmented grammar for the above grammar will be
S' -> S

S -> AA
A -> aA | b

(9) What is the major work of Preprocessor?

Ans. A preprocessor is a program that processes its input data to produce output that is used as input to another program

Eg. It can add all the header files for the program from beginning of the program compiles.

(10) What is the ouput for Lexical Anlayzer?

Ans. Generate Tokes

Que.2

(A) List out phases in Analysis phase and also Write down output for the given statement : a = b + c * 10 - 5 (where a,b,c are real) [8]

Ans.

The Analysis Phases

As translation progresses, the compiler's internal representation of the source program changes. We illustrate these representations by considering the translation of the statement

```
position := initial + rate * 60        (1.1)
```

Figure 1.10 shows the representation of this statement after each phase.

The lexical analysis phase reads the characters in the source program and groups them into a stream of tokens in which each token represents a logically cohesive sequence of characters, such as an identifier, a keyword (`if`, `while`, etc.), a punctuation character, or a multi-character operator like `:=`. The character sequence forming a token is called the *lexeme* for the token.

Certain tokens will be augmented by a "lexical value." For example, when an identifier like `rate` is found, the lexical analyzer not only generates a token, say **id**, but also enters the lexeme `rate` into the symbol table, if it is not already there. The lexical value associated with this occurrence of **id** points to the symbol-table entry for `rate`.

In this section, we shall use $\mathbf{id}_1$, $\mathbf{id}_2$, and $\mathbf{id}_3$ for `position`, `initial`, and `rate`, respectively, to emphasize that the internal representation of an identifier is different from the character sequence forming the identifier. The representation of (1.1) after lexical analysis is therefore suggested by:

$$\mathbf{id}_1 \ := \ \mathbf{id}_2 \ + \ \mathbf{id}_3 \ * \ 60 \qquad (1.2)$$

We should also make up tokens for the multi-character operator `:=` and the number `60` to reflect their internal representation, but we defer that until Chapter 2. Lexical analysis is covered in detail in Chapter 3.

The second and third phases, syntax and semantic analysis, have also been introduced in Section 1.2. Syntax analysis imposes a hierarchical structure on the token stream, which we shall portray by syntax trees as in Fig. 1.11(a). A typical data structure for the tree is shown in Fig. 1.11(b) in which an interior node is a record with a field for the operator and two fields containing pointers to the records for the left and right children. A leaf is a record with two or more fields, one to identify the token at the leaf, and the others to record information about the token. Additional information about language constructs can be kept by adding more fields to the records for nodes. We discuss syntax and semantic analysis in Chapters 4 and 6, respectively.

(B) What is Syntax Directed Definition? Explain Synthesized and inherited Attributes with small example. [8]

Ans. Syntax directed definition is a generalization of context free grammar in which each grammar symbol has an associated set of attributes.

Types of attributes are:

1. Synthesized attribute
2. Inherited attribute

◦Value of synthesized attribute at a node can be computed from the value of attributes at the children of that node in the parse tree.

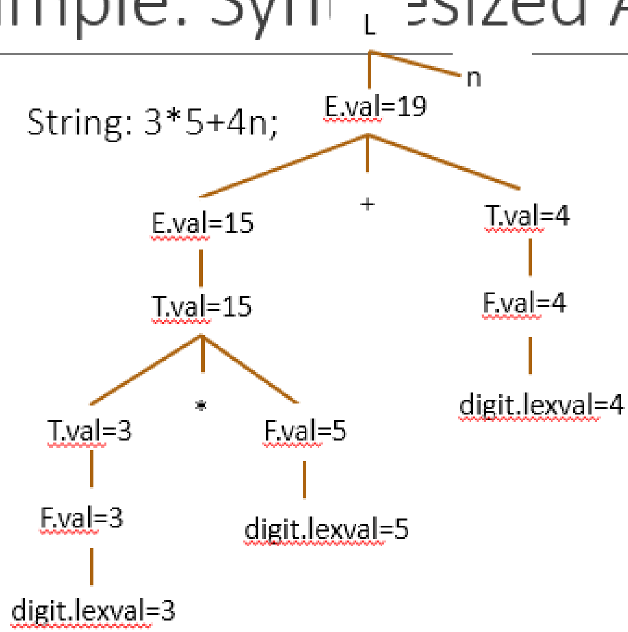
◦Syntax directed definition that uses synthesized attribute exclusively is said to be S-attribute definition.

◦An Annotated parse tree is a parse tree showing the value of the attributes at each node.

◦The process of computing the attribute values at the node is called Annotating or Decorating the parse tree.

Example: Synthesized Attributes

- String: 3*5+4n;



Annotated parse tree for 3*5+4n

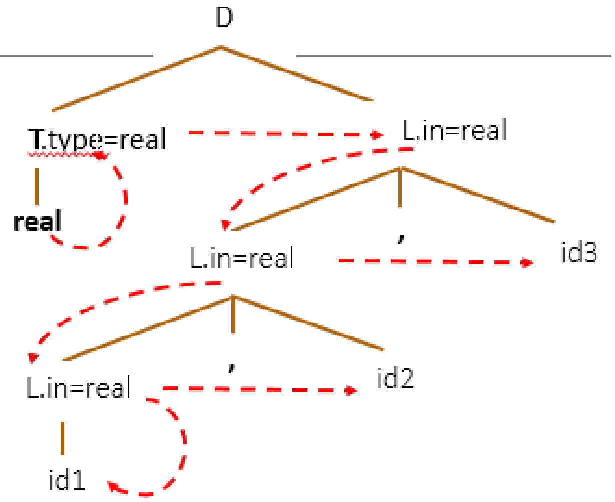
Production	Semantic Rules
L → E_n	Print (E.val)
E → E_1+T	E.Val = E1.val + T.val
E → T	E.Val = T.val
T → T_1*F	T.Val = T1.val * F.val
T → F	T.Val = F.val
F → (E)	F.Val = E.val
F → digit	F.Val = digit . lexval

An *inherited attribute* is a property of a symbol (node) that is determined by its parent node and its siblings in the parse tree.

An inherited value at a node in a parse tree is computed from the value of attributes at the parent and/or siblings of the node.

Example: Inherited Attribute

real id1,id2,id3



Production	Semantic Rules
D → T L	L.in = T.type
T → int	T.type = integer
T → real	T.type = real
L → L_1 , id	L_1.in = L.in, addtype(id.entry,L.in)
L → id	addtype(id.entry,L.in)

D→TL$_1$d, id

OR

(B) Discuss various code optimization techniques with examples. (Minimum 4 Techniques are required) [8]

Ans.

Code Optimization Technique.

→ Common Sub Expr. elimination

folding

length = (22/7) * d

pi = 3.14

r = 5

A = pi * r * r

Will replaced with 3.14 * 5 * 5

→ t1 := 4 * i

t2 := a[t1]

t3 := 4 * j

t4 := 4 * i

t5 := n

t6 := b[t4] + t5

⇒

t1 := 4 * i

t2 := a[t1]

t3 := 4 * j

t5 := n

t6 := b[t1] + t5

Scanned by CamScanner

→ Variable propagation :

use one vari. instead of another

x = pi ;

area : x * r * r ;

⇒ a = pi * r * r

→ Code movement :

i) To reduce the size of the code.
ii) To reduce the frequency of execu. of code.

```
for (i=0; i<=10; i++)          t = y*5
{ x = y*5;           =>        for (i=0; i<=10; i++)
  k = (y*5)+50;                { x = t;
                                 k = (t)+50;
```

↳ Loop invariant computation
to place some amount of code from the loop to outside the loop.

```
                               N = min-1;
while (i<=min-1)               while (i<=N)
{                    =>        {
  sum = sum+a[i];                sum = sum+a[i];
}                              }.
```

It is also called as code motion.

Scanned by CamScanner

Date
Page

→ Strength Reduction
- Strength of certain operator is higher than others.
- for instance strength of * is higher than +.

```
                               t = 7;
for (i=1; i<=50; i++)          for (i=1; i<=50; i++)
{                    =>        {
  count = i*7;                   count = t;
}                                t = t+7;
                               }
```

→ Dead code elimination
- A variable is said to be live in a program if the value contained into is subsequently.
- A variable is said to be dead at a point in a program if the value contained into it is never been used.
- And an optimization can be perform by eliminating such a dead code.

Eg.
```
i = 0;
if (i==1)
{
  a = x+5;
}
```

Scanned by CamScanner

Que.3

(A) Solve the given question using LALR(1) and construct the table for LALR(1) [8]

S→ AA

$A \rightarrow aA \mid b$

Ans.

Canonical LR **CLR** (look ahead)

Q. ⇒ $S \rightarrow CC$
$C \rightarrow cC \mid d$

0 = $S' \rightarrow .S, \$$
① $S \rightarrow .CC, \$$ first of \$ is \$
③ $C \rightarrow .cC, c|d$ first of C
③ $C \rightarrow .d, c|d$

I_1 = goto(I_0, S)
$S' \rightarrow S., \$$

I_2 = goto(I_0, C)
$S \rightarrow C.C, \$$
$C \rightarrow .cC, \$$ first of \$ = \$
$C \rightarrow .d, \$$

I_3 = goto(I_0, c)
$C \rightarrow c.C, c|d$
$C \rightarrow .cC, c|d$ first of c|d = c|d
$C \rightarrow .d, c|d$

I_4 = goto(I_0, d)
$C \rightarrow d., c|d$

I_5 = goto(I_2, C)
$S \rightarrow CC., \$$

I_6 = goto(I_2, c) when lookahead is different then item set is different
$C \rightarrow c.C, \$$
$C \rightarrow .cC, \$$ first of \$ = \$
$C \rightarrow .d, \$$

I_7 = goto(I_2, d)
$C \rightarrow d., \$$

I_8 = goto(I_3, C)
$C \rightarrow cC., c|d$

I_3 = goto(I_3, c)
$C \rightarrow c.C, c|d$
$C \rightarrow .cC, c|d$
$C \rightarrow .d, c|d$

I_4 = goto(I_3, d)
$C \rightarrow d., c|d$

I_9 = goto(I_6, C)
$C \rightarrow cC., \$$

I_6 = goto(I_6, c)

I_7 = goto(I_6, d)

Scanned by CamScanner

LALR :-

Item Set Same like CLR

String Parsing :-

cdd $

0	dd $	dd $
0d4	d $	S4
0C2	d$	R3
0C2d7	$	S7
0C2C5	$	R3
051	$	R1
		Accept

0	cdd $
0c3	dd $
0c3d4	d $
0c3C8	d $
0C2	d $
0C2d7	$
0C2C5	$
051	$

Laxmi Page No.
Date / /

Action			Goto	
c	d	$	S	C
			1	2

0	s3	s4			
1			Accept		5
2	s6	s7			8
3	s3	s4			
4	r3	r3			
5			r1		9
6	s6	s7			
7			r3		
8	r2	r2	r2		
9					

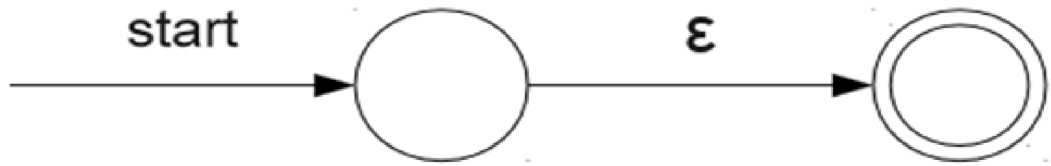
Scanned by CamScanner

(B) Write down rules for converting Regular expression into NFA using Thompson's Rule. [4]

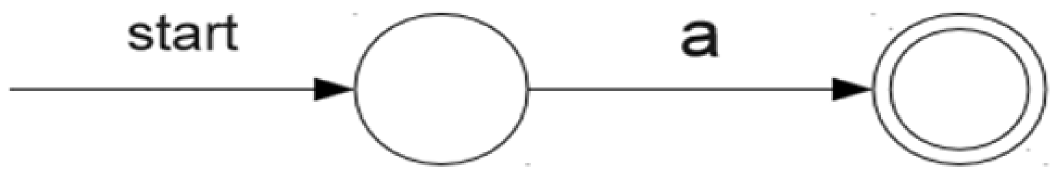
Ans. Rules for following data

a

Epsilon

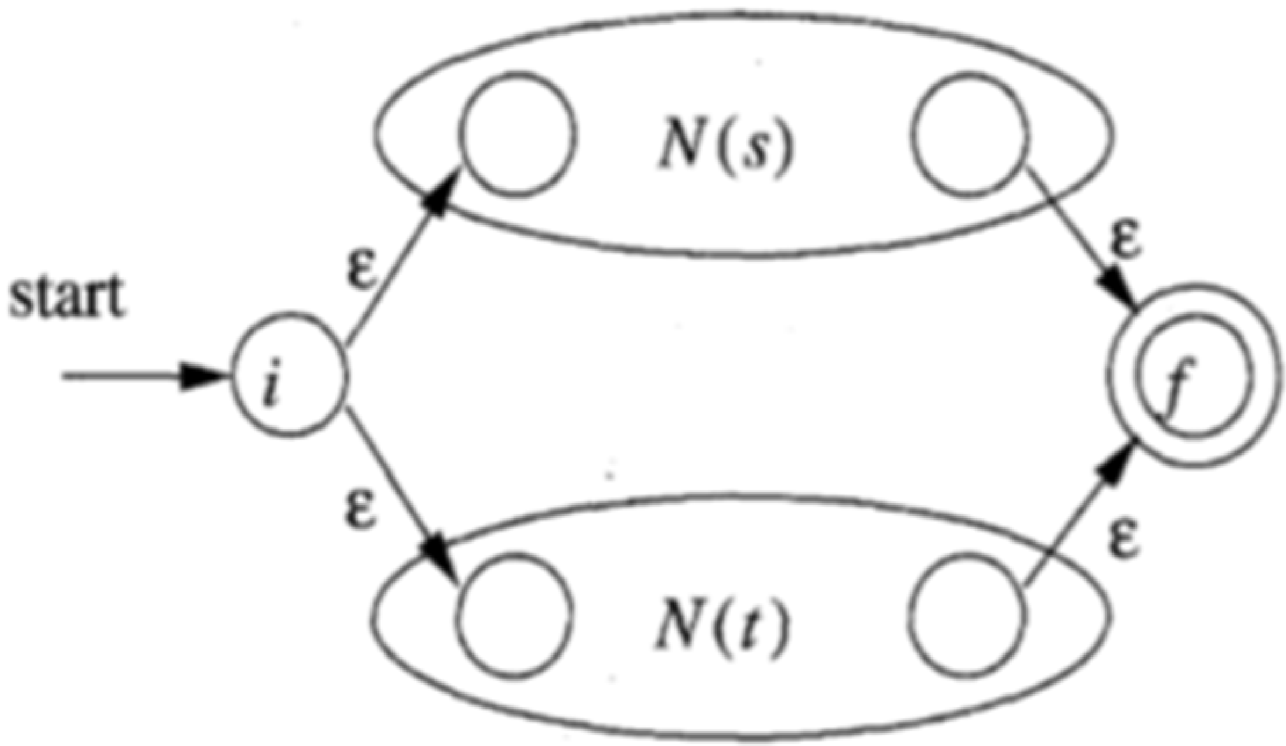


Automaton for ε

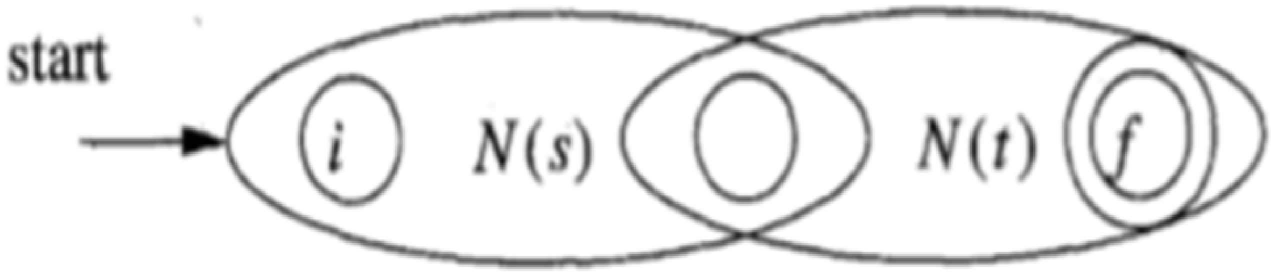


Automaton for single character a

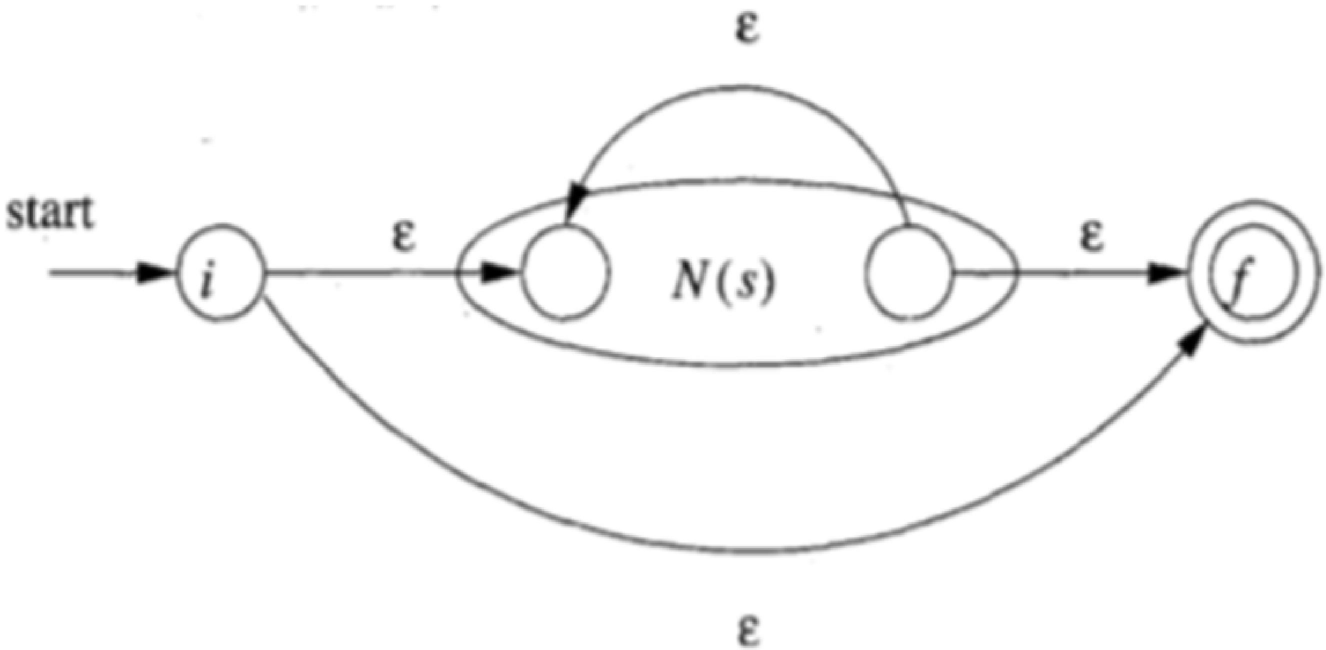
s|t



st



s*

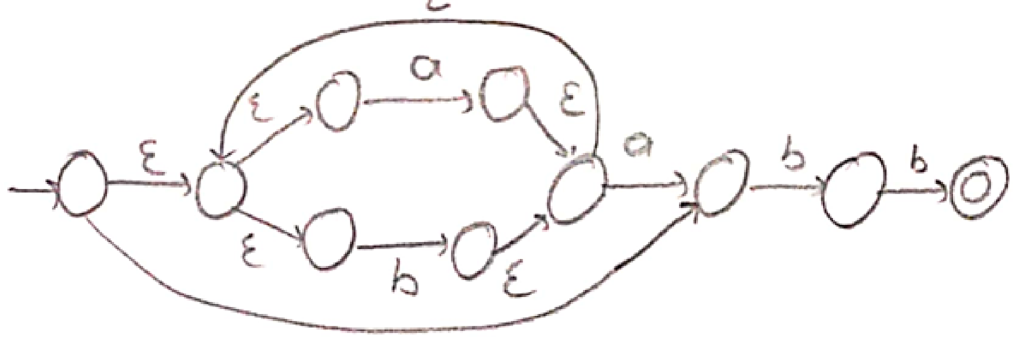


(C) Draw the ϵ-NFA using Thompson's Rule for given two regular expression [4]

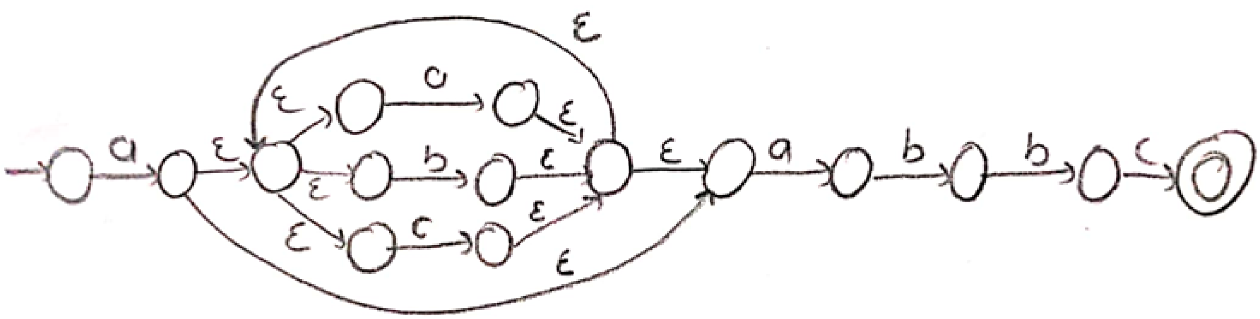
i) (a|b)* abb ii) (a|b|c)*

Ans.

1) (a|b)*abb (Using Thompson's)



2) a(a|b|c)*abbc (Using Thompson's)



OR

(A) Solve the given question using CLR and construct the table for CLR(1) [8]

S→ CC

C→ cC | d

Also check that string: dd is parsed or not.

Ans.

Canonical LR CLR (look ahead)

Q. ⇒ $S \rightarrow CC$

$C \rightarrow cC \mid d$

I_0 =

① $S' \rightarrow .S, \$$

② $S \rightarrow .CC, \$$ first of \$ is \$

③ $C \rightarrow .cC, c|d$ First of C

③ $C \rightarrow .d, c|d$

I_1 = goto(I_0, S)

$S' \rightarrow S., \$$

I_2 = goto(I_0, C)

$S \rightarrow C.C, \$$

$C \rightarrow .cC, \$$ first of \$ = \$

$C \rightarrow .d, \$$

I_3 = goto(I_0, c)

$C \rightarrow c.C, c|d$

$C \rightarrow .cC, c|d$ first of c|d = c|d

$C \rightarrow .d, c|d$

I_4 = goto(I_0, d)

$C \rightarrow d., c|d$

I_5 = goto(I_2, C)

$S \rightarrow CC., \$$

I_6 = goto(I_2, c) when lookahead is different then item set is different

$C \rightarrow c.C, \$$

$C \rightarrow .cC, \$$ first of \$ = \$

$C \rightarrow .d, \$$

I_7 = goto(I_2, d)

$C \rightarrow d., \$$

I_8 = goto(I_3, C)

$C \rightarrow cC., c|d$

I_3 = goto(I_3, c)

$C \rightarrow c.C, c|d$

$C \rightarrow .cC, c|d$

$C \rightarrow .d, c|d$

I_4 = goto(I_3, d)

$C \rightarrow d., c|d$

I_9 = goto(I_6, C)

$C \rightarrow cC., \$$

I_6 = goto(I_6, c)

I_7 = goto(I_6, d)

LALR :-

Item Set Same like CLR

String Parsing :-

cdd$

0	dd$	dd$
0d4	d$	S4
0C2	d$	R3
0C2d7	$	S7
0C2C5	$	R3
0S1	$	R1
		Accept

0	cdd$
0c3	dd$
0c3d4	d$
0c3C8	d$
0C2	d$
0C2d7	$
0C2C5	$
0S1	$

Scanned by CamScanner

	Action			Goto	
	c	d	$	S	C
0	s3	s4		1	2
1			Accept		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2	r2		
9					

Scanned by CamScanner

(B) What is Dominators in loops in flow graph? Explain with small example. [4]

Ans.

1. Dominators

- In a flow graph, a node d dominates n if every path to node n from initial node goes through d only. This can be denoted as'd dom n'.
- Every initial node dominates all the remaining nodes in the flow graph. Similarly every node dominates itself.

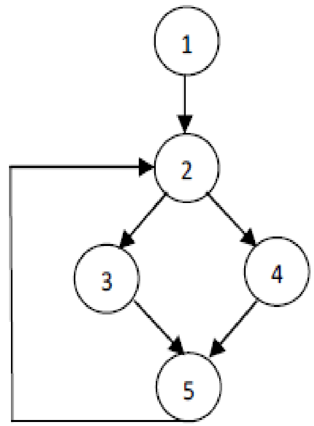


Fig.7.1. Dominators

- Node 1 is initial node and it dominates every node as it is initial node.
- Node 2 dominates 3, 4 and 5.
- Node 3 dominates itself similarly node 4 dominates itself.

(C) Explain Global Data Flow Analysis. [4]

Ans. - In order to do code optimization and a good job of code generation , a compiler needs to collect information about the program as a whole and to distribute this information to each block in the flow graph.

- Data flow equations are the equations representing the expressions that are appearing in the flow graph.

- Data flow information can be collected by setting up and solving systems of equations that relate information at various point in a program.

- A typical equation has the form

Out[S] = gen[S] U (in[S] – kill[S])

And can be read as

“the information at the end of a statement is either generated within the statement, or enters at the beginning and is not killed as control flows through the statement”.

- The details of how data-flow analysis are set up and solved depend on three factors.

1.The notion of generating and killing depend on the desired information. i.e. for some problem, instead of proceeding along with the flow of control and defining out[S] in terms of in[S], we need to proceed backwards and defines in[S] in terms of out[S].

2.Since data flows along control paths, data-flow analysis is affected by the control constructs in a program.

3.There are subtleties that go along with such statements as procedure calls, assignment through pointer and even assignments to array variables.

Que.4

(A) Explain Issues in design of Code generator. [8]

Ans. Input to Code Generator

Target Program

Memory Management

Instruction Selection

Register Allocation
Choice of Evaluation
Approached to Code Geneation

1) Input to Code Generator

The input to the code generator consists of intermediate representation of source program produced by front end.

Intermediate languages can be

Linear representation such as Postfix Notation

Three Address representation such as Quadruples

Virtual Machine representation such as Stack Machine Code

Graphical representation such as Syntax Tree and DAGs.

The Semantic errors should be done before submitting input to the Code Generator

The Code Generation Phase can therefore proceed on the assumption that its input is free of Errors.

2) Target Program

Target program may take on variety of forms :

Absolute Machine Language : It can be placed in a fixed location in memory and immediately executed.

Relocatable Machine Language : A set of relocatable object modules can be linked together and loaded for execution.

Assembly Language : We can generate symbolic instructions and use the macro facilities of the assembler to help generate code.

3) Memory Management

Mapping Names in source program to address of data objects in run time memory is done cooperatively by front end and code generator.

We assume that a name in a three address statement refers to a symbol table entry for the name.

From Symbol table information, a relative address can be determined for the name in a data area.

4) Instruction Selection

Example :

a = b + c

d = a + e

Would be translated into

```
MOV   b,R0
ADD    c,R0
MOV     R0,a
MOV     a,R0    //its redundant
ADD       e,R0
MOV     R0,d
```

5) Register Allocation

Register Usage can be done like :

During *Register Allocation*, We select set of variables that will reside in registers at a point in the program.

During a subsequent *Register Assignment* phase, we pick specific register that a variable will reside in.

6) Choice of Evaluation Order

The order in which computations are performed can affect the efficiency of the target code.

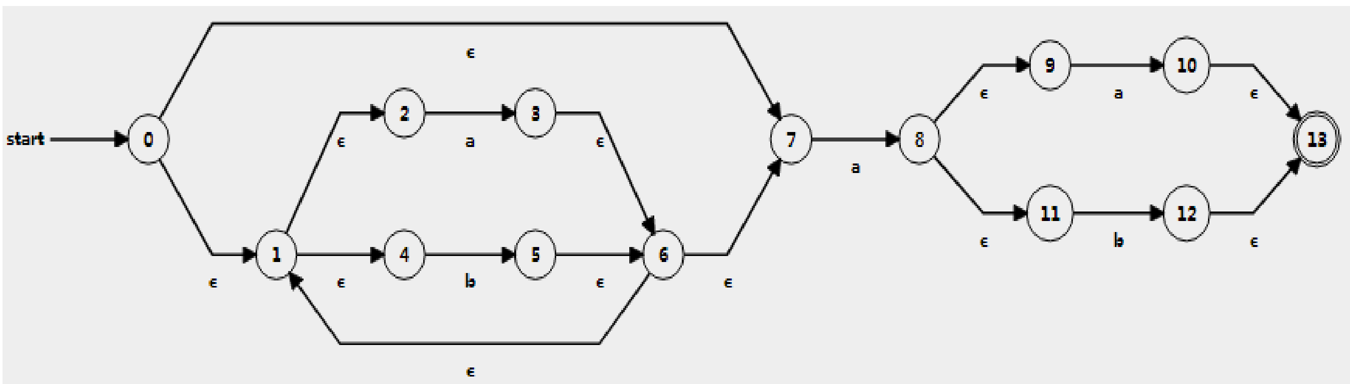
Some computation orders require fewer registers to hold intermediate results than others.

Picking a best order is difficult, NP-complete problem.

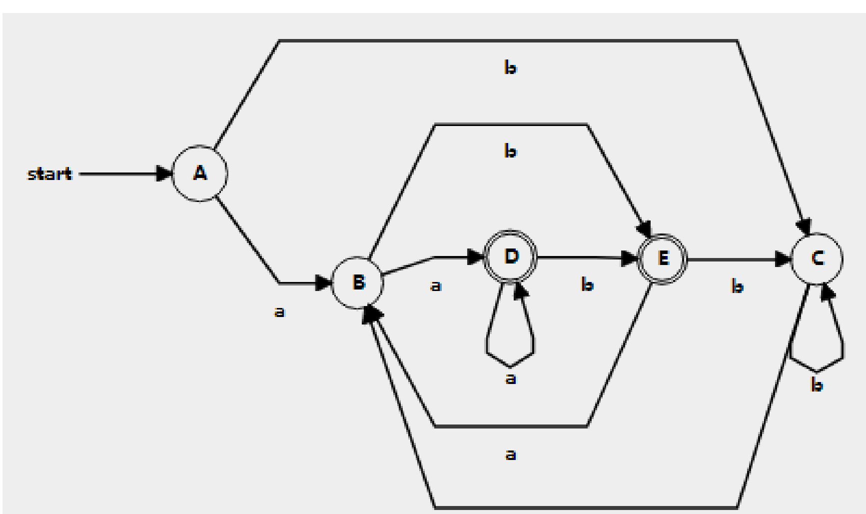
7) Approach to Code Generation
The most important criterion for a code generator is that it produces correct code.
The design of code generator should be in such a way so it can be implemented, tested and maintained easily.

(B) Convert following regular expression to DFA using subset construction method: [8]
(a|b)*a(a|b)

Ans.



NFA STATE	DFA STATE	TYPE	a	b
{0,1,2,4,7}	A		B	C
{1,2,3,4,6,7,8,9,11}	B		D	E
{1,2,4,5,6,7}	C		B	C
{1,2,3,4,6,7,8,9,10,11,13}	D	accept	D	E
{1,2,4,5,6,7,12,13}	E	accept	B	C



OR

(A) Create Regular Expression for following Languages over $\Sigma = \{0, 1\}$ [8]

(a) String starting and ending with same character
(b) All binary strings where 2nd symbol from starting is 0
(c) Language consisting of exactly two 0's
(d) Strings having Even length
(e) String length atleast 2
(f) String length atmost 2
(g) String starting with 0 and having odd length
(h) String starting or ending with 01 or 111

Ans. a) 1 (0|1)* 1 or 0 (0|1)* 0 or 1 or 0 or null

b) (0|1) 0 (0|1)*

c) 1* 01*01*

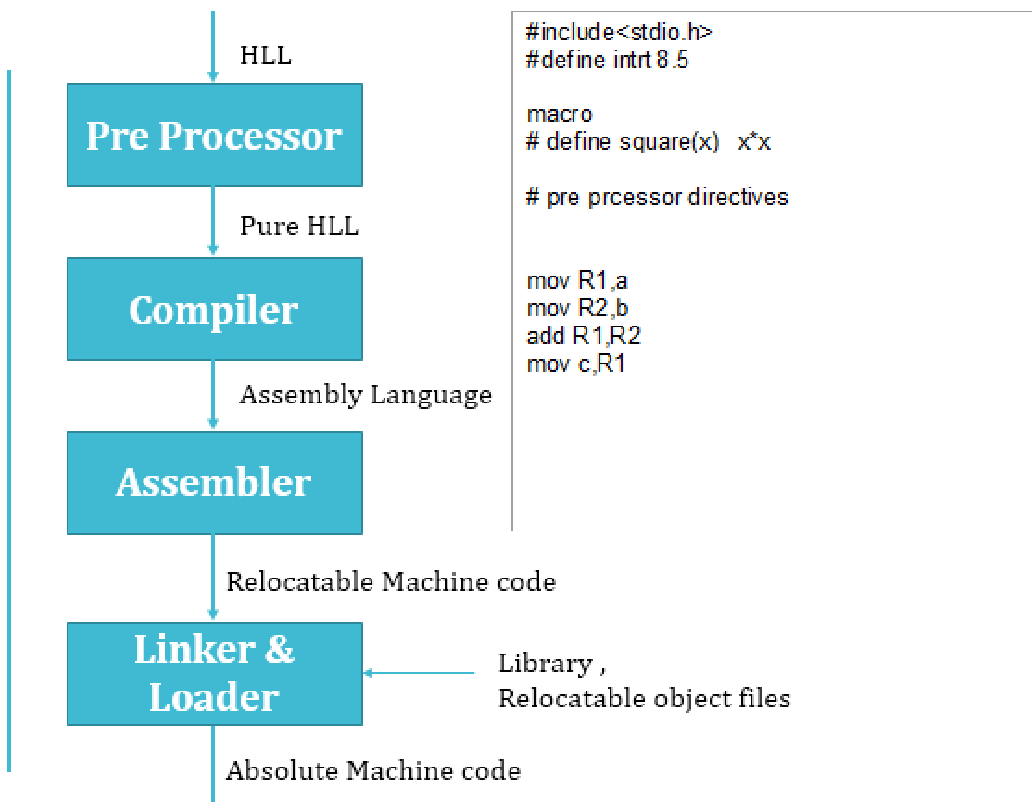
d) ((0|1)(0|1))*

g) 0 ((0|1)(0|1))*

h) (01|111)(0|1)*| (0|1)*(01|111)

(B) Explain Language Processing process in detail. List out cousins of Compilers [8]

Ans.



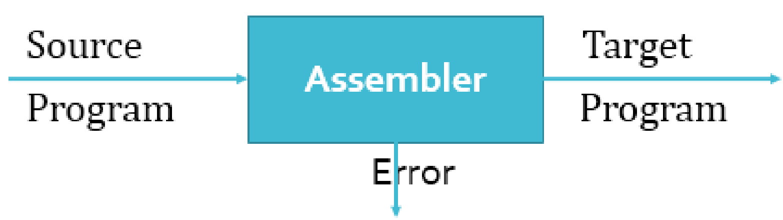
1) Preprocessor

- Preprocessor produces input to compilers.
- They may perform the following functions

- **Macro Processing** : A preprocessor may allow a user to define macros that are short hands for longer constructs.
- **File Inclusion** : A preprocessor may include header files into program text.
- **"Rational" Preprocessor** : These processors augment older languages with more modern flow of control and data structuring facilities. (Built in Macro for construct like While or If Statement)
- **Language Extension** : These processor attempt to add capabilities to the language by what amount to built in macros. the language equal is a database query language embedded in C. statement beginning with ## are taken by preprocessor to be database access statement unrelated to C and translated into procedure call on routines that perform database access.

2) Assembler

- An assembler is a translator used to translate **assembly language to machine language.**
- An assembler translates a low-level language, an assembly language to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.



3) **Linker and Loader**

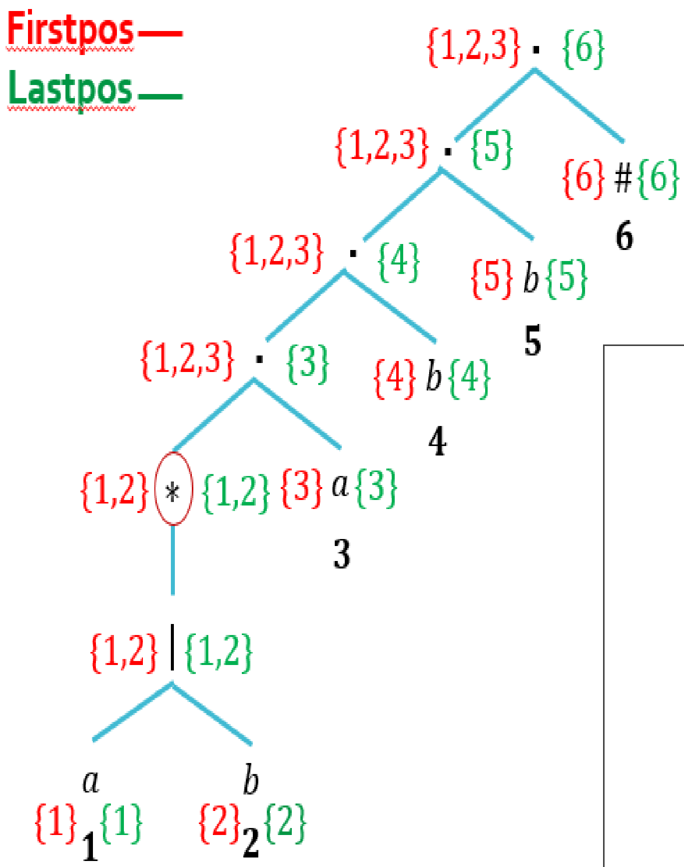
- A program called loader performs the two functions of loading and link editing
- The process of loading consist of taking relocatable machine code , altering the relocatable address and placing the altered instructions and data in memory at the proper location.
- Linker allows us to make a single program from several files of relocatable machine code.
- Types of Linking : Static and Dynamic
- Types of Loader : Compile and Go loader, Absolute Loader, Relocating Loader.

Que.5

(A) Draw the DFA using Syntax tree method with the Firstpos and Lastpos in the tree for given regular expression (a|b)*ab [6]

Ans.

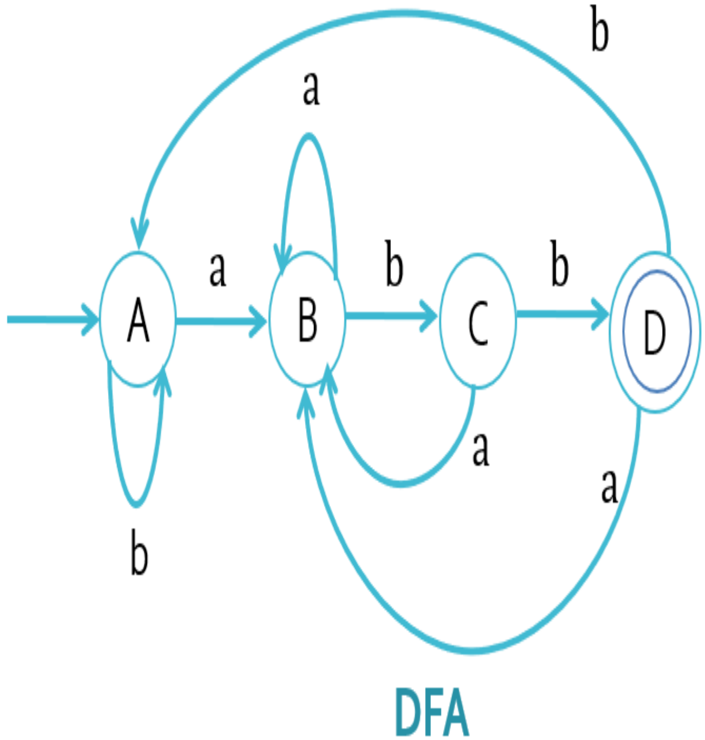
Step 4: Calculate followpos



Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3

$$\{1,2\} \ * \ \{1,2\}$$
$$\boldsymbol{n}$$

$$i = lastpos(n) = \{1,2\}$$
$$firstpos(n) = \{1,2\}$$
$$followpos(1) = \{1,2\}$$
$$followpos(2) = \{1,2\}$$



Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3

States	a	b
A={1,2,3}	B	A
B={1,2,3,4}	B	C
C={1,2,3,5}	B	D
D={1,2,3,6}	B	A

(B) Differentiate Static and Dynamic Memory Allocation [6]

Ans.

*	Static Memory Allocation	Dynamic Memory Allocation
*	m/m is allocated before the execution of the program begins.	* M/m is allocated during the execution of the program.
*	No m/m allocation or Deallocation are performed during the compil execution.	* M/m bindings are established and destroyed during the execution.
*	Variables remain perme. allocated.	* Allocated only when program unit is active.
*		
*	Implementation using stack and heap.	* Implemented using data segments.
*	Pointers a is needed to accessing variables.	* No need of Dynamic allocated pointers.
*	faster than Dynamic	* Slower execution than stati
*	More m/m space required.	* Less m/m space required.

(C) Find out the first and follow of given grammar. [4]

S→1AB | ϵ

A→ 1AC | 0C

B→ 0S

C→ 1

Ans. First Set

S → {1, ϵ}

A → {1, 0}

B → {0}

C → {1}

Follow Set

S → {$}

A →{ 0, 1}

B → {$}

C → {0, 1}

OR

(A) Explain Peephole Optimization. [6]

Ans. It is a Simple and Effective technique for locally improving target code.

This technique is applied to improve performance of the target program by examining the short sequence of target instructions (peephole) and replacing these instructions by shorter or faster sequence whenever possible.

It is a small, moving window on the target program.

Redundant Load and Store

- Redundant Load and Store can be eliminated.
- Example :

MOV R0,x

MOV x,R0

Flow of Control Optimization

- Unnecessary jumps can be eliminated using following type of peephole optimization.

goto L1

..... → Goto L2

L1 : goto L2

Algebraic Simplification

- Statements such that

X = X + 0 or X = X * 1

Can be eliminated by peephole optimization.

Strength Reduction

- Some instructions are cheaper than others.
- To improve performance, we can replace some instructions with equivalent cheaper instructions.
- For Example, addition and subtraction is cheaper than multiplication and division.

Machine Idioms

- Target instructions have equivalent machine instructions for computation purpose.
- Example : Some machines have auto increment and auto decrement addressing modes. So, it can be used in program for statements like i = i+1.

Define basic block and Draw DAG representation of Basic block.

T1 = 4 * I

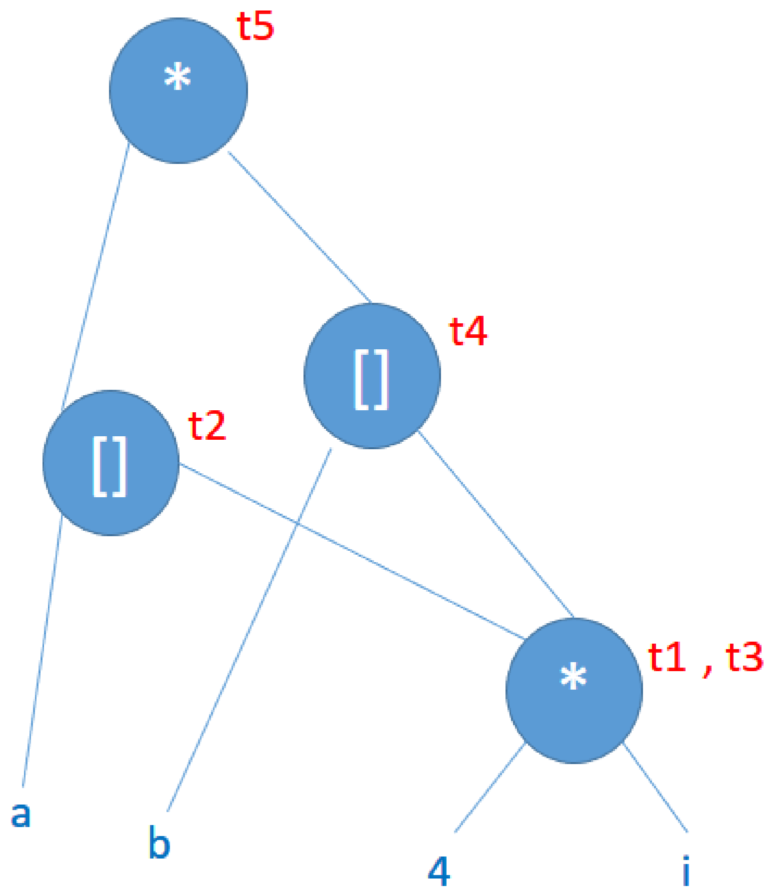
T2 = a[t1]

T3 = 4 * I

T4 = b[t3]

T5 = t2 * t4

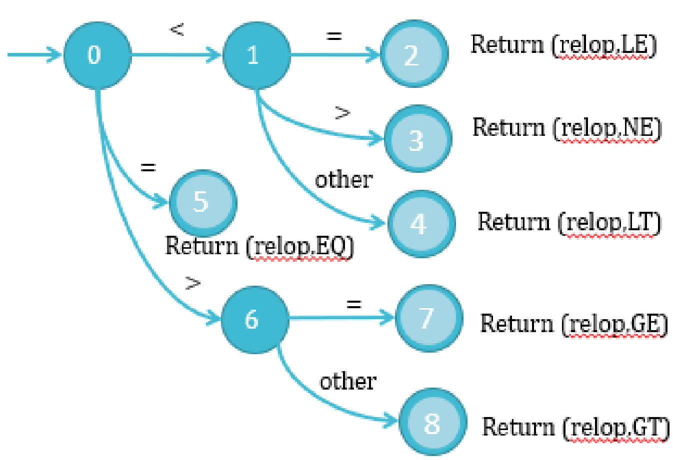
A Basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.



(B) Answer the Following : [6]
1) What is Transition diagram ? Draw notations used in transition diagram.
2) Draw transition diagram for relational operators.

Ans.

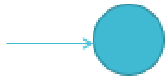
Transition Diagram for Relational Operators

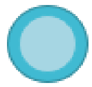


- Symbolized representation of transition diagram uses:


is a state

is a transition


is a start state


is a final state

(C) Differentiate Compiler and Assembler. [4]

Ans.

Compiler Vs Assembler

BASIS FOR COMPARISON	COMPILER	ASSEMBLER
Basic	Generates the assembly language code or directly the executable code.	Generates the relocatable machine code.
Input	Preprocessed source code.	Assembly language code.
Phases/ Passes	The compilation phases are lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generation, code optimization, code generation.	Assembler makes two passes over the given input.
Output	The assembly code generated by the compiler is a mnemonic version of machine code.	The relocatable machine code generated by an assembler is represented by binary code.

Compiler	Assembler
Translates high-level language into machine code	Translates assembly language into machine code
Translates all code at the same time	Uses the processors instruction set to convert
Only needed once to create an executable file	Runs quickly as conversation between two low level languages is just reliant on the processors instructions set
Will only inform you of the first error it finds	
Once compiled runs quickly but compiling can take a long time	

Que.6

(A) Enlist the parameter Passing Methods and explain all in detail with example. [8]

Ans. - All programming languages have a notion of a procedure, but they can differ in how these procedures get their arguments. In this section, we shall consider how the actual parameters are associated with the formal parameters.

1.Call by value

2.Call by reference

3.Copy restore

4.Call by name

1.Call by Value:

- In *call-by-value,* the actual parameter is evaluated (if it is an expression) or copied (if it is a variable). The value is placed in the location belonging to the corresponding formal parameter of the called procedure.
- The operation in formal parameter do not change its value in actual parameter.
- This method is used in C, Java and is a common option in C++, as well as other languages.

2.Call by Reference:

- This method is also called call by address or call by location.
- In *call-by-reference,* the address of the actual parameter is passed to the callee as the value of the corresponding formal parameter.
- Uses of the formal parameter in the code of the callee are implemented by following this pointer to the location indicated by the caller.
- Changes to the formal parameter thus appear as changes to the actual parameter.

3.Copy and restore:

- This method is hybrid between call by value and call by reference. This method is also called copy-in-copy-out or values result.

- The calling procedure calculates the value of actual parameter and it then copies to activation record for the called procedure.
- During execution of called procedure, the actual parameters value is not affected.
- If the actual parameter has X-values then, at return the value of formal parameter is copied to actual parameter.
-

•

4.Call by names:
- This is less popular method of parameter passing.
- Procedure is treated like macro. The procedure body is substituted for call in caller with actual parameters substituted for formulas.
- The actual parameters can be surrounded by parenthesis to preserve their integrity.
- The local names of called procedure and names of calling procedure are distinct.
-

•

(B) What is Basic block? Mention Three conditions to identify Leaders in basic block. [4]

Ans. - A basic block is sequence of 3-address statements where control enters at the beginning and leaves only at the end without any jumps or halts.
- In order to find the basic blocks, we need to find the leaders in the program. Then a basic block will start from one leader to the next leader but not including next leader.
-

Identifying leaders in basic block:
1.1st statement is leader.
2.Statement that is target of conditional or unconditional statement is a leader.
- if () goto 100 (Line number 100 is a leader)
- goto 200 (Line number 200 is a leader)
3.Statement that follows immediately a conditional or unconditional statement is a leader.

(C) Remove Left factoring from the given grammar (if present) [4]

S→ abc | abd | abf |abe |Ak
A→ xyz | xym | xyn | xyp | xy | Bg
B→ pqr | pl | pt | w

Ans.
S→abX |Ak
X→c|d|f|e
A→xyX' |Bg
X'→z|m|n|p|ϵ
B→pZ|w
Z→qr|l|t

OR

(A) Write down three Address Code for given example and draw the Quadruple, Triple and Indirect Triple table. [8]

z=a*-b+a*-b

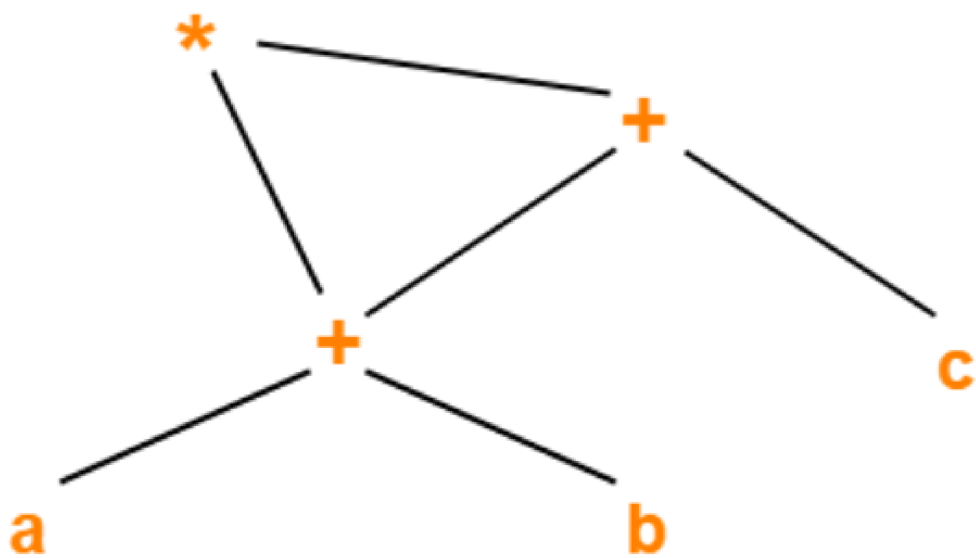
Ans.
t1:=Uminus b
t2:=a*t1
t3:=uminus b

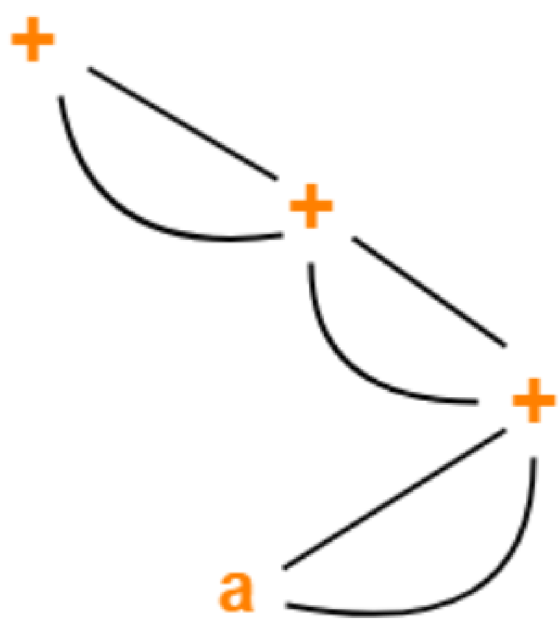
t4:=a*t3
t5:=t2+t4
z:=t5

(B) Draw DAG for following : [4]

1) (a+b) * (a+b+c)
2) (((a + a) + (a + a)) + ((a + a) + (a + a)))

Ans.





(C) Remove the Left recursion from the given grammar (if present) [4]

$A \rightarrow A + B \mid B$

$B \rightarrow B - C \mid C$

$C \rightarrow C * D \mid D$

$D \rightarrow id$

Ans.

$A \rightarrow BA'$

$A' \rightarrow +BA' \mid \epsilon$

$B \rightarrow CB'$

$B' \rightarrow -CB' \mid \epsilon$

$C \rightarrow DC'$

$C' \rightarrow *DC' \mid \epsilon$

$D \rightarrow id$

---Best of Luck---

MARWADI UNIVERSITY
MU-FOT
CE-FOT1 (MU), IT-FOT1 (MU)
Semester 6 - Summer

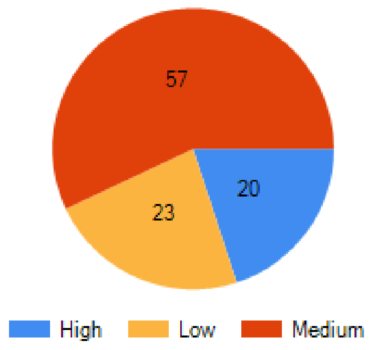
Subject : COMPILER DESIGN (01CE0601)

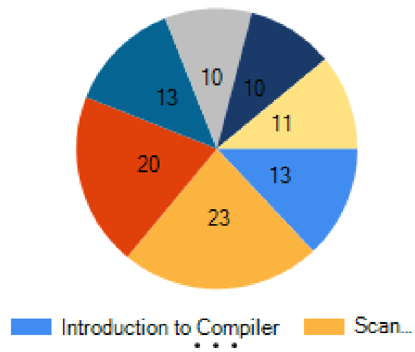
Date : 26-Apr-2022 **Time : 3 Hours** **Total Marks : 100**

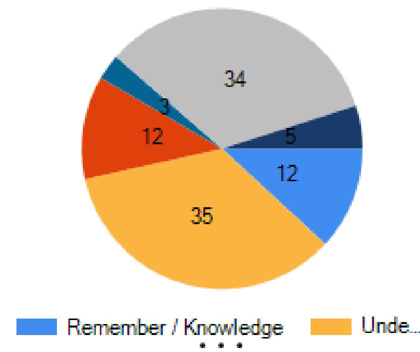
Difficulty Level	Weightage Recommended	Weightage Actual	No of Question	Total Marks	Question List
High	20	19.77	8	34	1(A), 3(A), 3(B), 5(A), 5(B), 6(B)
Low	20	23.26	14	40	1(A), 1(B), 3(B), 3(C), 4(A), 5(A), 5(B), 5(C)
Medium	60	56.98	23	98	1(A), 1(B), 2(A), 2(B), 3(A), 3(C), 4(A), 4(B), 5(C), 6(A), 6(C)

Module Name	Weightage Recommended	Weightage Actual	No of Question	Total Marks	Question List
Introduction to Compiler	15	13.37	6	23	1(A), 1(B), 2(A), 4(B), 5(C)
Scanner	15	23.26	10	40	1(A), 1(B), 3(B), 3(C), 4(A), 4(B), 5(A), 5(B)
Parsing	30	19.77	11	34	1(A), 1(B), 3(A), 5(C), 6(C)
Intermediate Code Generation	10	12.79	5	22	1(A), 1(B), 2(B), 6(A), 6(B)
Memory Management	10	9.88	5	17	1(A), 1(B), 5(B), 6(A)
Code Optimization	10	9.88	4	17	1(A), 2(B), 3(B), 3(C)
Code Generation	10	11.05	4	19	1(A), 4(A), 5(A), 6(B)

Blooms Taxonomy	Weightage Recommended	Weightage Actual	No of Question	Total Marks	Question List
Remember / Knowledge	10	11.63	10	20	1(A), 1(B), 3(B), 6(A)
Understand	20	34.88	18	60	1(A), 1(B), 3(B), 3(C), 4(A), 4(B), 5(A), 5(B)
Apply	25	11.63	4	20	3(C), 6(A), 6(C)
Analyze	25	3.49	3	6	1(B), 5(C)
Evaluate	10	33.72	8	58	2(A), 2(B), 3(A), 4(B), 5(A), 5(C)
Higher order Thinking	10	4.65	2	8	6(B)


57
23
20
High
Low
Medium


10
13
10
11
20
13
23
Introduction to Compiler
Scan...


34
3
12
5
12
35
Remember / Knowledge
Unde...