**Assembler:**

An assembler is a program that accepts as input an assembly language program &amp;

produces its machine language equivalent along with information for the loader.

7.1.1 Functions of Assembler:

- Convert mnemonic operation codes to their machine language equivalents

- Convert symbolic operands to their equivalent machine addresses

- Decide the proper instruction format

- Convert the data constants to internal machine representations

- Write the object program and the assembly listing

- Convert symbolic operands to their equivalent machine addresses

What is a FRP?

-The function of the assembler is to replace each symbol by its m/c address &amp;

if we refer to that symbol before it is defined , its address is not known by the

assembler. Such a problem is called FRP.

Solution to FRP:-

- Forward reference problem (For IBM 360) is solved by making two passes

over the assembly code.

Pass1:-

Purpose- Define Symbols &amp; Literals.

1) Keep track of LC.

2) Determine length of m/c instructions.

3) Remember the value of symbols until pass2.

4) Process some Pseudo-ops e.g EQU, DS, DC.

5) Remember Literals.

Pass2:

Purpose- Generate Object Program.

1) Lookup values of symbols.

2) Generate instructions.

3) Generate data.

4) Process some Pseudo-ops e.g. USING, DROP.

a) POT (Pseudo Opcode Table):-

POT is a fixed length table i.e. the contents of these table are altered during the assembly

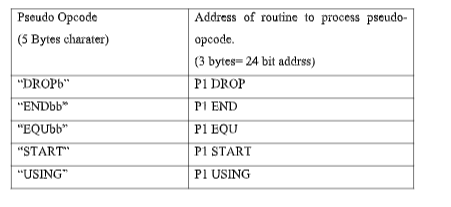
process.

The table will actually contain the physical addresses.

- POT is a predefined table.

- In PASS1 , POT is consulted to process some pseudo opcodes like-DS,DC,EQU

- In PASS2, POT is consulted to process some pseudo opcodes like DS,DC,USING,DROP



b) MOT (Mnemonic Opcode Table):-

MOT is a fixed length table i.e. the contents of these tables are altered during the

assembly process.

MOT is a predefined table.

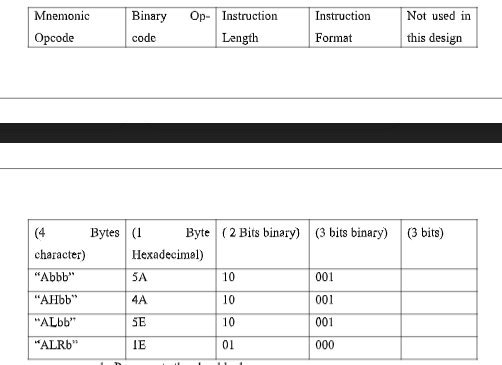
- In PASS1 , MOT is consulted to obtain the instruction length.(to Update LC)

- In PASS2, MOT is consulted to obtain:

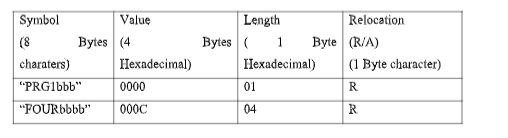
a) Binary Opcode (to generate instruction)

b) Instruction length ( to update LC)

c) Instruction Format (to assemble the instruction).



C) Symbol table (ST):

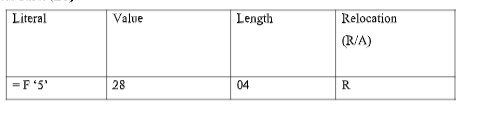


- ST is used to keep a track on the symbol defined in the program.

- In pass1- whenever the symbol is defined an entry is made in the ST.

- In pass2- Symbol table is used to generate the address of the symbol.

D) Literal Table (LT):

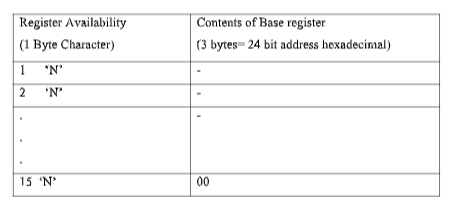


- LT is used to keep a track on the Literals encountered in the program.

- In pass1- whenever the literals are encountered an entry is made in the LT.

- In pass2- Literal table is used to generate the address of the literal.

E) Base Table (BT):



- Code availability-

- Y- Register specified in USING pseudo-opcode.

- N--Register never specified in USING pseudo-opcode.

- BT is used to keep a track on the Register availability.

- In pass1- BT is not used.

- In pass2- In pass2, BT is consulted to find which register can be used as base registers

along with their contents.

F) Location Counter (LC):

- LC is used to assign addresses to each instruction &amp; address to the symbol defined in

the program.

- LC is updated only in two cases:-

a) If it is an instruction then it is updated by instruction length.

b) If it is a data representation (DS, DC) then it is updated by length of data field.

7.7 Pass1 Databases:

- The purpose of pass1 is to define the symbols &amp; literals.

- The various databases maintained are as follows.

a) Original source cards:

It contains the original program.

a) Location counter (LC):

It is used to assign addresses to the instruction &amp; addresses to the symbols

defined in the program.

c)Mnemonic Opcode Table (MOT):

MOT is consulted to obtain the instruction length ( to Update LC).

d)Pseudo Opcode Table (POT):

POT is consulted to process some pseudo opcodes like DS, DC, EQU.

e) Symbol Table (ST):

Whenever a symbol is defined entry is made in ST.

f) Literal Table (LT):

Whenever literal are encountered entry is made in LT.

g) Copy File:

It is prepared to be used by pass2.

Pass2 Databases:

- The purpose of pass2 is to generate the instruction &amp; data.

- The various databases maintained are as follows.

a) Copy File:

It is prepared by pass1.

b) Location counter (LC):

It is used to assign addresses to the instruction &amp; addresses to the symbols

defined in the program.

c)Mnemonic Opcode Table (MOT):

MOT is consulted to obtain the binary opcode, instruction length &amp; instruction

format.

d)Pseudo Opcode Table (POT):

POT is consulted to process some Pseudo opcodes like DS, DC, USING, DROP.

e)Symbol Table (ST):

It is used to generate the address for the symbols.

f) Literal Table (LT):

It is used to generate the address of the literals.

g)Base Table (BT):

It is consulted to find the registers which are available to be used as base

registers.

h) Instruction workspace:

It is used to hold the instruction while its various parts are getting assembled.

i) PUNCH workspace:

It is used to punch the assembled instruction onto the cards.

j) PRINT workspace:

It is used to generate a printed assembly listing.

k) Assembled object cards:

Which contain the object program in a format required by the loader.

**Code Optimization**

The code optimization in the synthesis phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources (i.e. CPU, Memory) so that faster-running machine code will result.

Optimizing an algorithm is beyond the scope of the code optimization phase. So the program is optimized. And it may involve reducing the size of the code. So optimization helps to:

* Reduce the space consumed and increases the speed of compilation.
* Manually analyzing datasets involves a lot of time. Hence we make use of software like Tableau for data analysis. Similarly manually performing the optimization is also tedious and is better done using a code optimizer.
* An optimized code often promotes re-usability.

**Dead Code Elimination:**

* Copy propagation often leads to making assignment statements into dead code.
* A variable is said to be dead if it is never used after its last definition.
* In order to find the dead variables, a data flow analysis should be done.

***Example:***

* C

|  |
| --- |
| c = a \* b  x = a  till  d = a \* b + 4    //After elimination :  c = a \* b  till  d = a \* b + 4 |

**Constant Propagation-**

In this technique,

* If some variable has been assigned some constant value, then it replaces that variable with its constant value in the further program during compilation.
* The condition is that the value of variable must not get alter in between.

**Example-**

pi = 3.14

radius = 10

Area of circle = pi x radius x radius

### ****Common Sub-expression Elimination:****

The expression or sub-expression that has been appeared and computed before and appears again during the computation of the code is the common sub-expression. Elimination of that sub-expression is known as**Common sub-expression elimination.**

**Example 1:**

*Before elimination* –

*a = 10;*

*b = a + 1 \* 2;*

*c = a + 1 \* 2;*

*//’c’ has common expression as ‘b’*

*d = c + a;*

*After elimination*–

*a = 10;*

*b = a + 1 \* 2;*

*d = b + a;*

Algebraic simplification

More general form of constant folding, e.g.,

x + 0 ⇒ x

x – 0 ⇒ x

x \* 1 ⇒ x

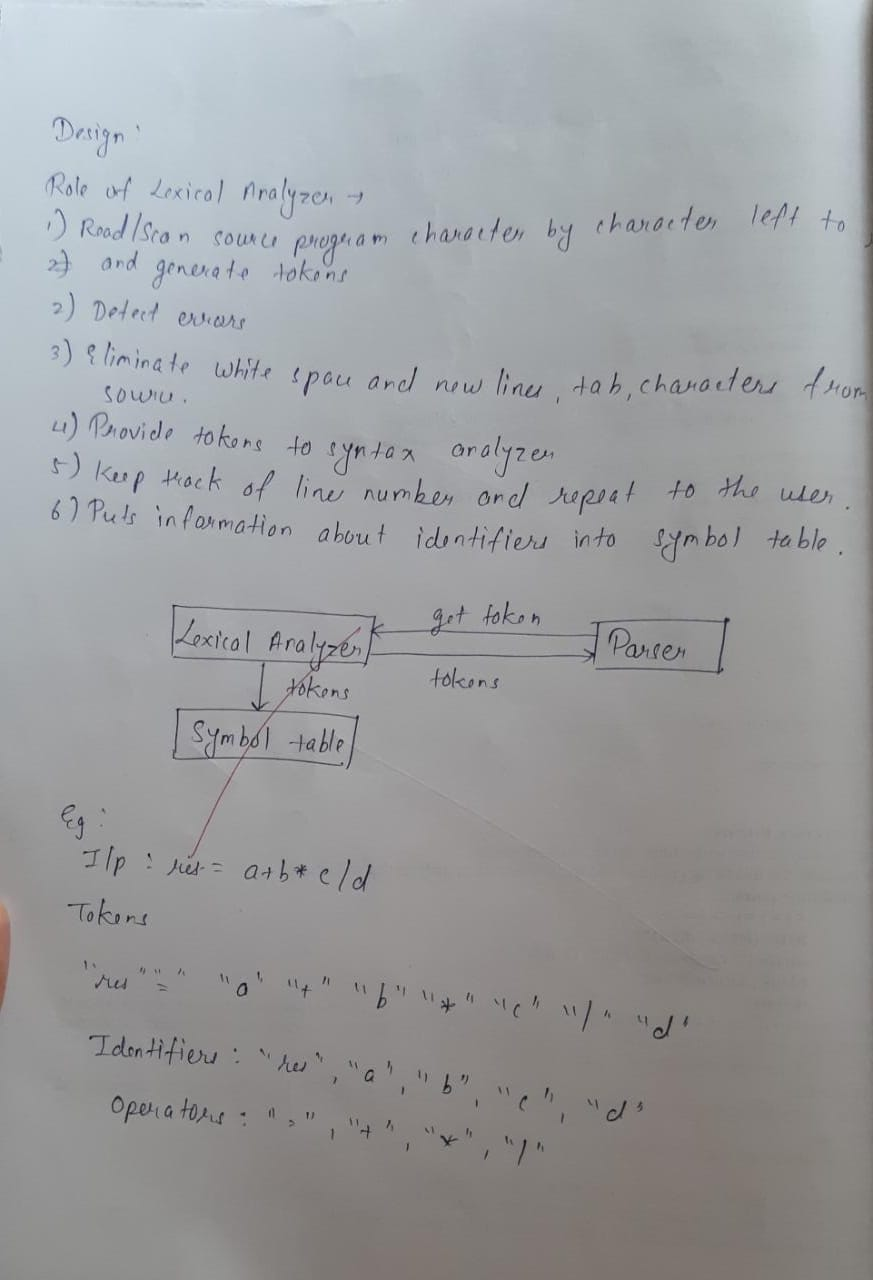
x / 1 ⇒ x

x \* 0 ⇒ 0

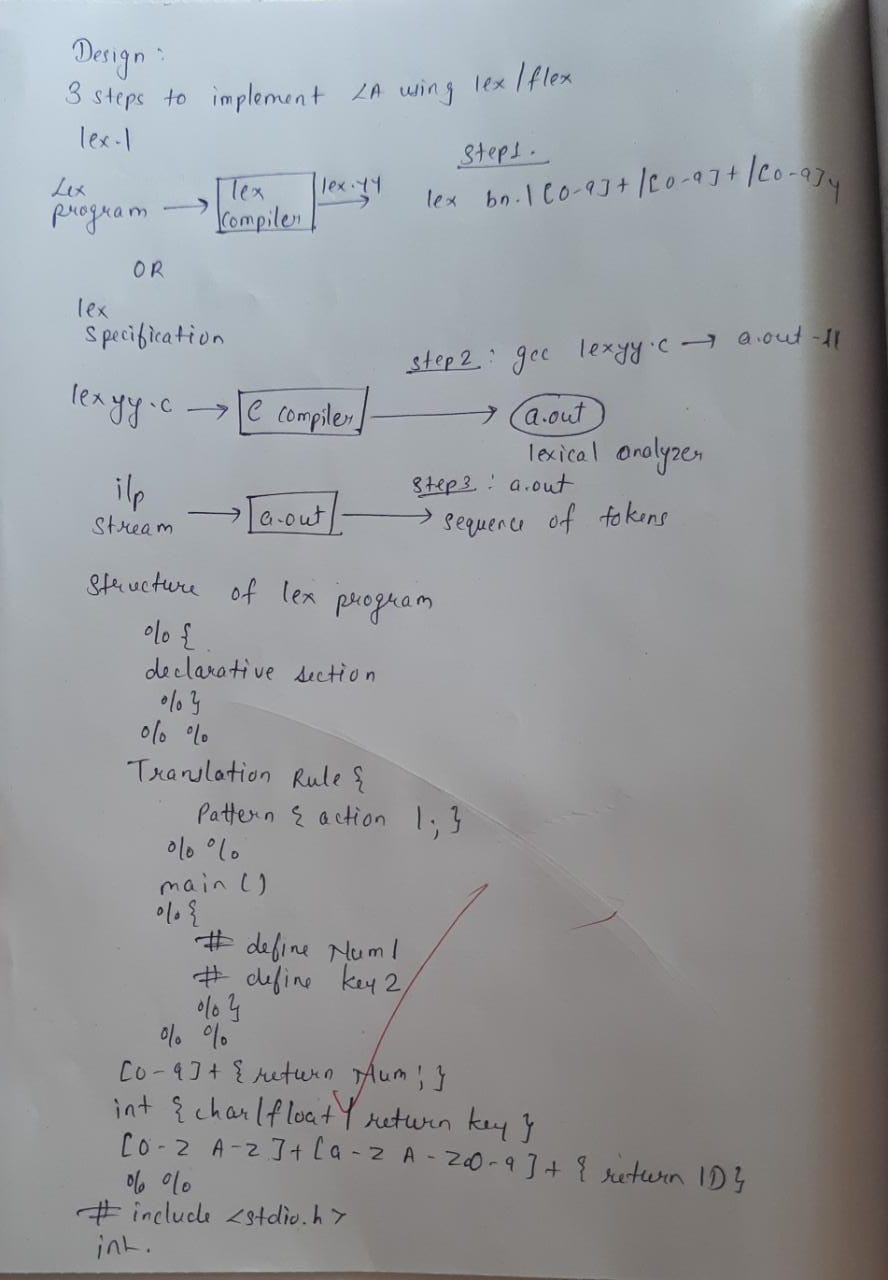
Repeatedly apply the rules

(y \* 1 + 0) / 1 ⇒ y

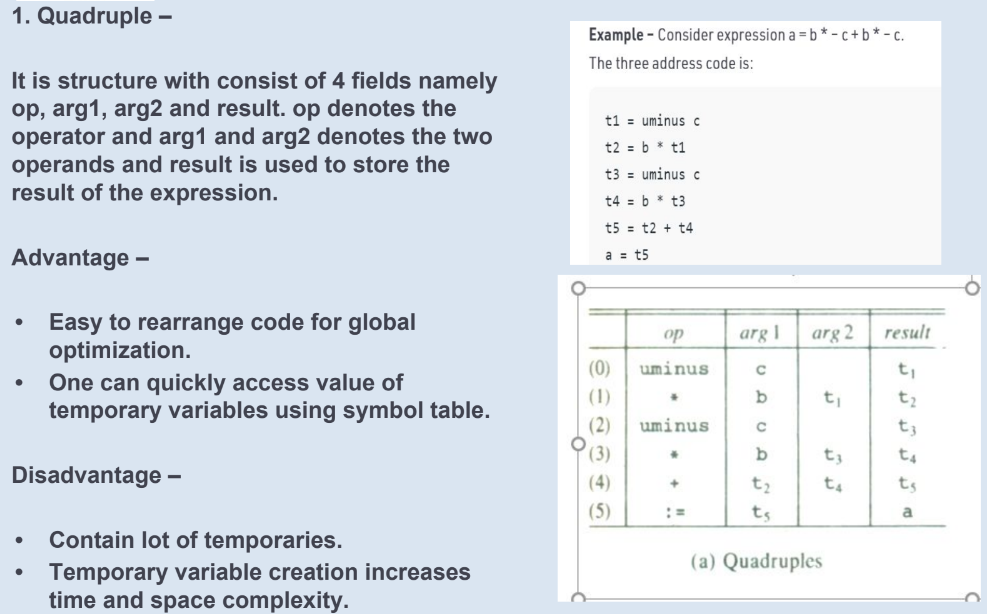
**Lexical analyzer**

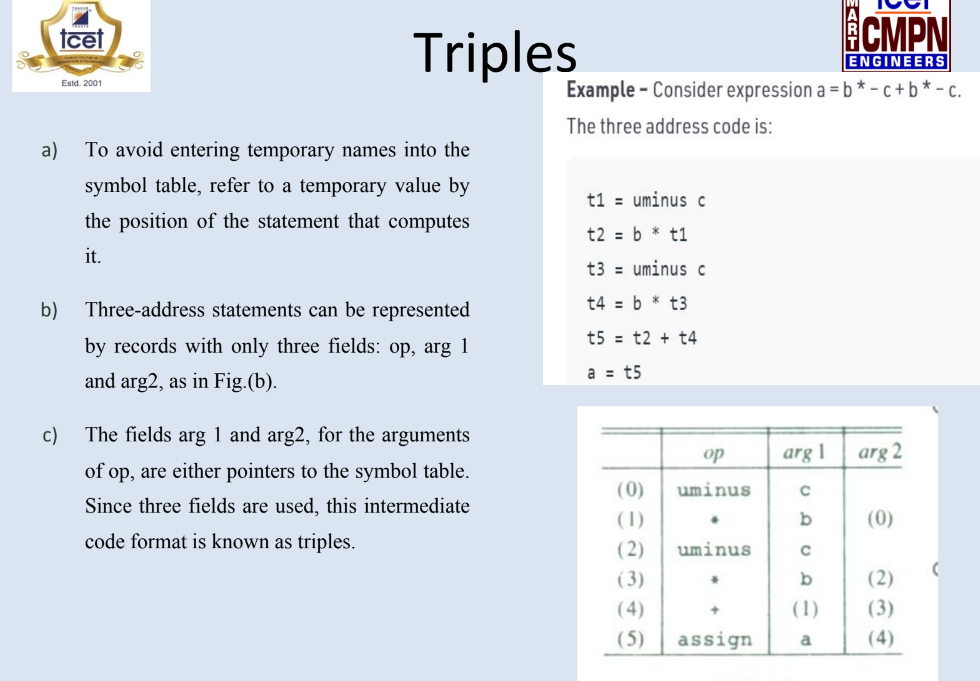


**Lex/flex tool**

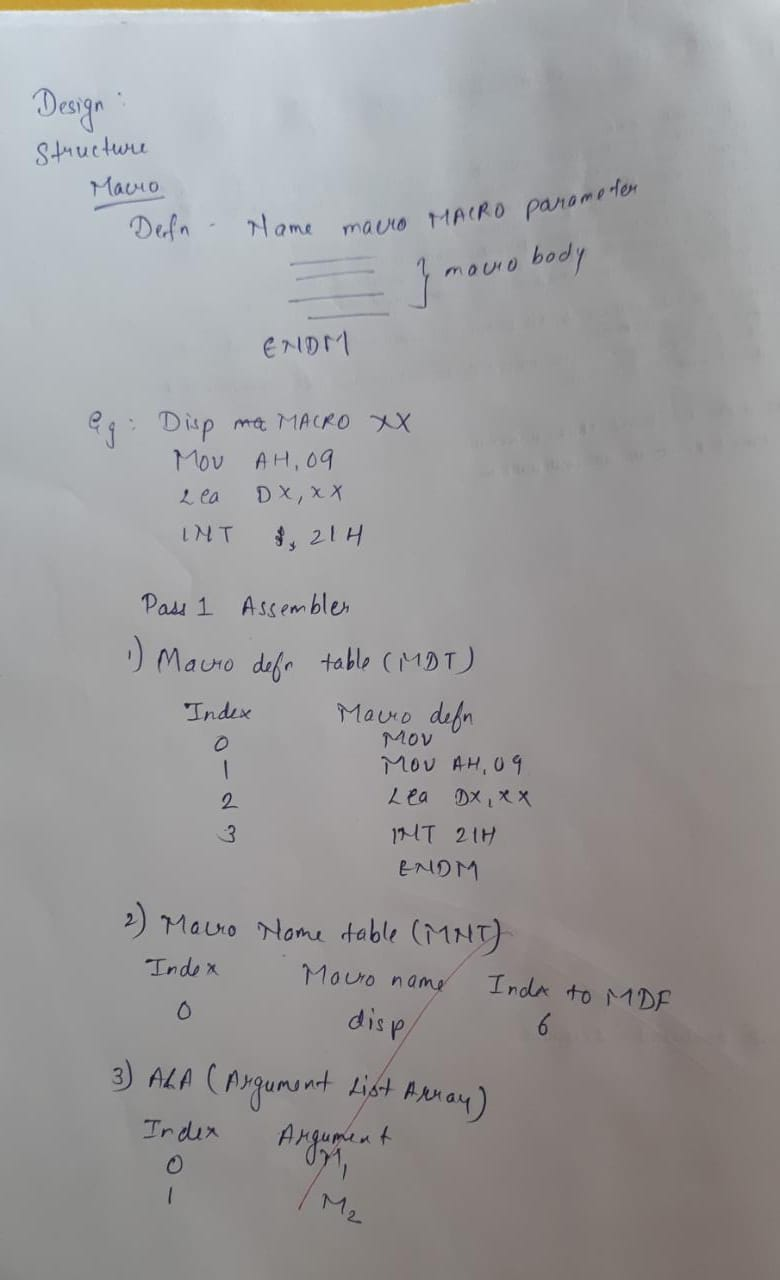


**3 Address code**

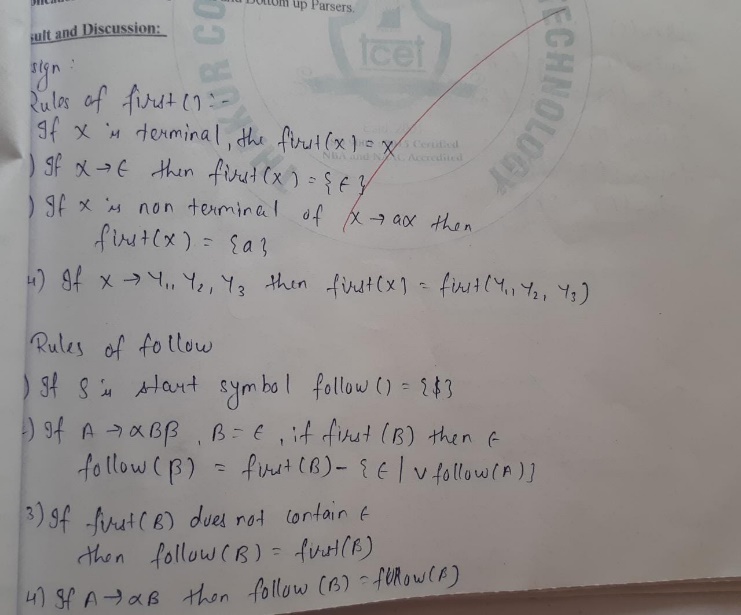


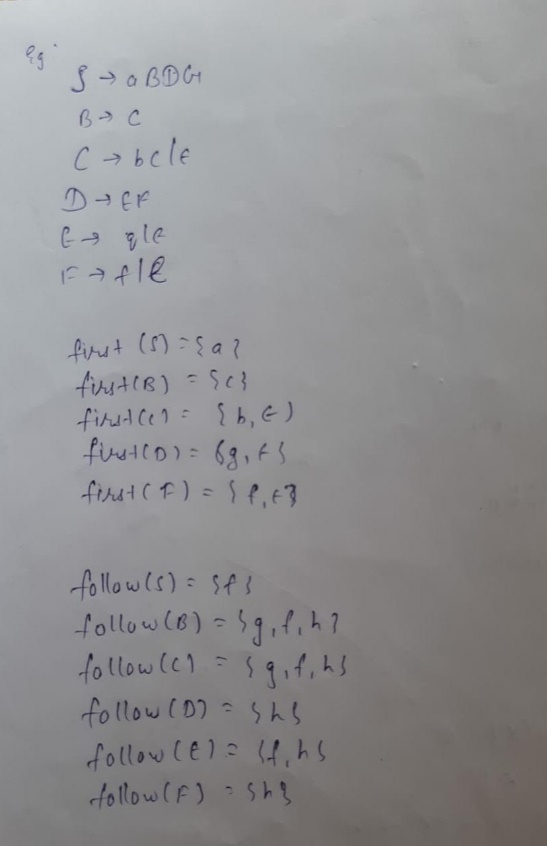


**Macro Processor**



First Follow





**If X is Grammar Symbol, then First (X) will be −**

* If X is a terminal symbol, then FIRST(X) = {X}
* If X → ε, then FIRST(X) = {ε}
* If X is non-terminal & X → a α, then FIRST (X) = {a}
* If X → Y1, Y2, Y3, then FIRST (X) will be

**Computation of FOLLOW**

**Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.**

FOLLOW(A) = {a|S ⇒\* αAaβ where α, β can be any strings}

**Rules to find FOLLOW**

* If S is the start symbol, FOLLOW (S) ={$}
* If production is of form A → α B β, β ≠ ε.

(a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

(b) If FIRST (β) contains ε (i. e. , β ⇒\* ε), then

        FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

∵ when β derives ε, then terminal after A will follow B.

* If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.