a) Write an Algorithm to build an Huffman Tree. Construct a Huffman tree for the following data item and Frequency 3+5

Data item	A	B	C	D	E	F	G	H
Frequency	22	5	11	19	2	11	25	5

→

Algorithm to build an Huffman Tree:

i) get the string
ii) Get the frequency occurance of each symbol.
iii) Select the symbol having lowest probability of occurance.
iv) Combine the two trees into a single & again choose two symbols with smallest frequency & then combine them again.

v) Repeat again the process until the forest consist of one (single) tree.

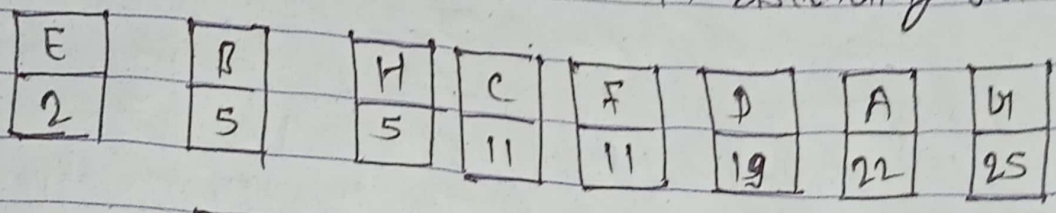
2nd part,

Soln

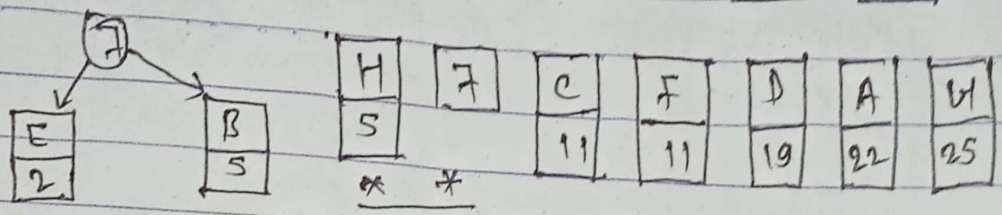
Here,

A	B	C	D	E	F	G	H
22	5	11	19	2	11	25	5

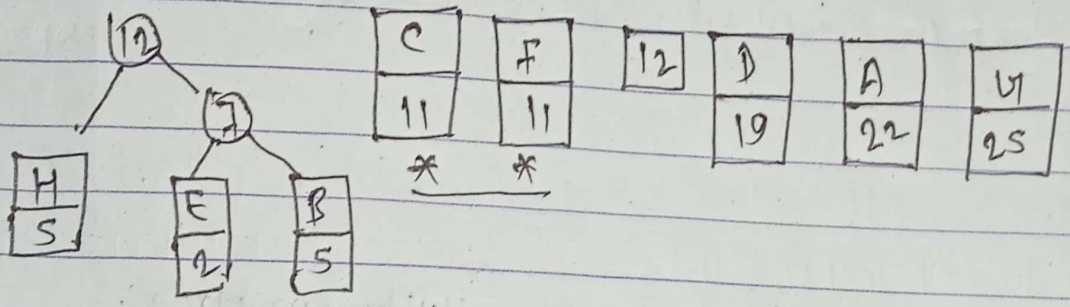
Step 1: Sort the above data item in ascending order.



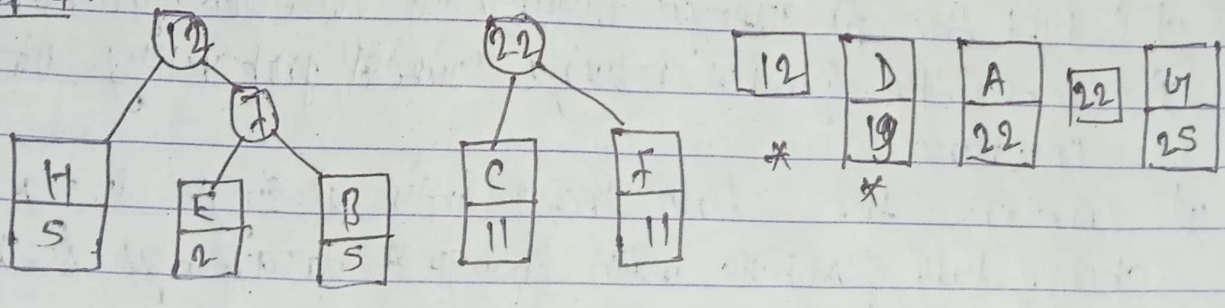
Step 2:



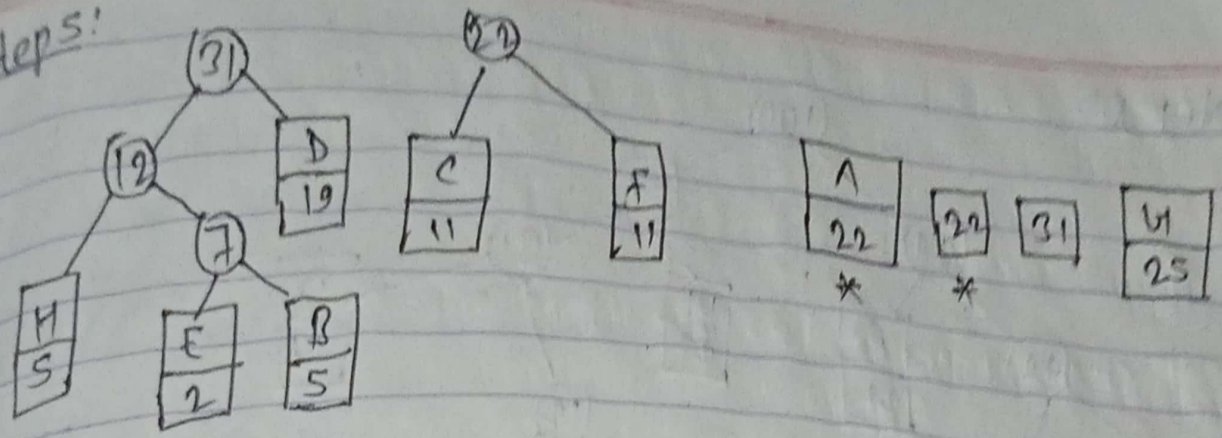
Step 3:



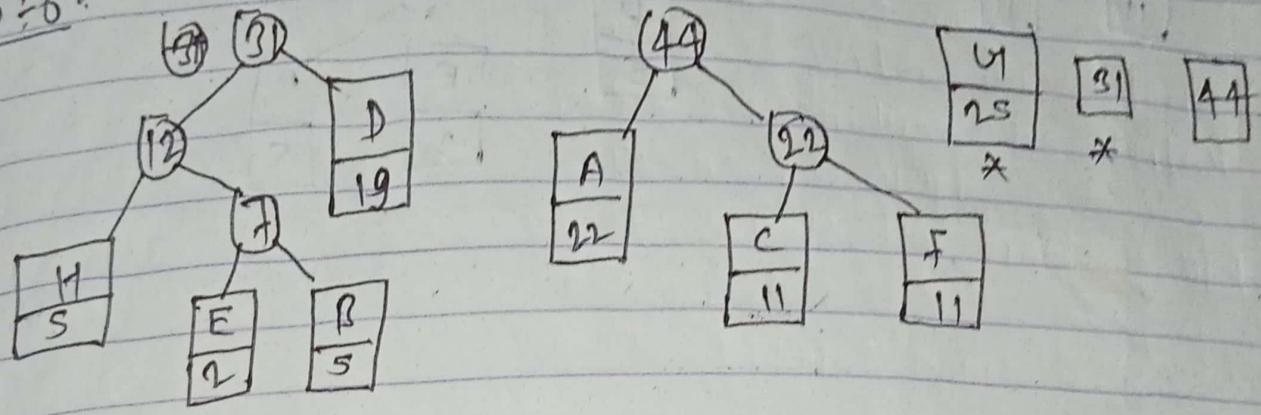
Step 4:



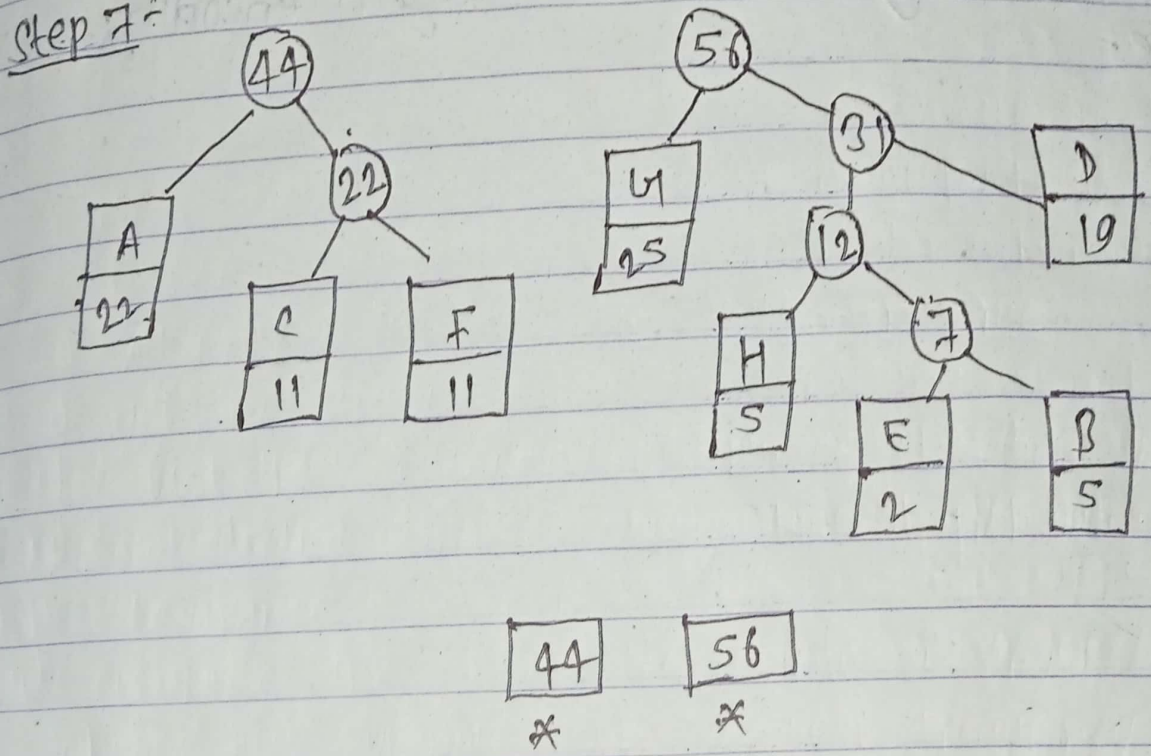
Steps:



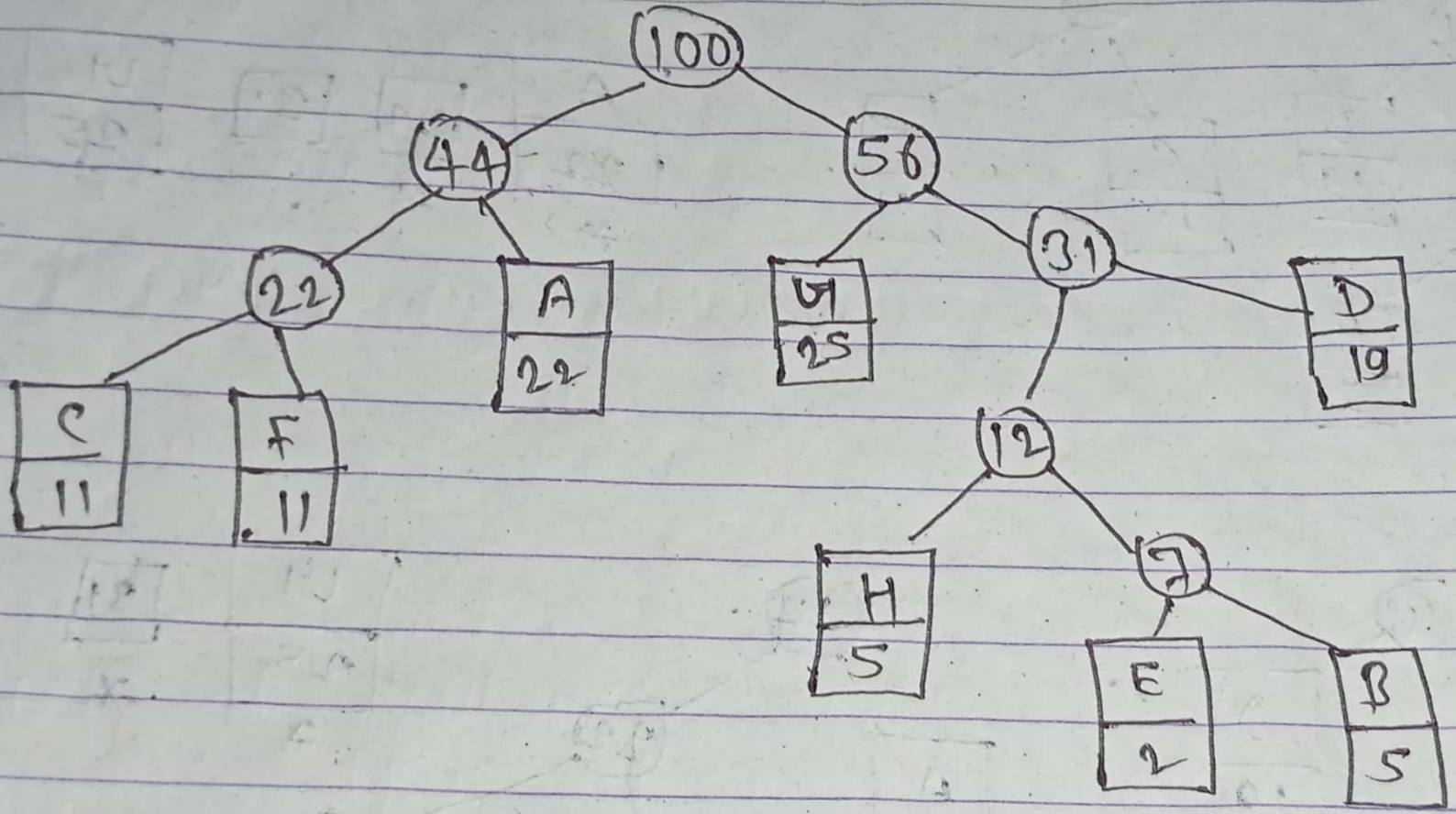
Step :6:



Step 7:



step 8:



b) What is the advantage of variable length encoding? Construct Huffman tree and generate code for the following symbol with their 7

b) Write the algorithm to convert the infix expression to postfix expression ~~to postfix expression~~ using stack implementation. Evaluate the following expression 8

A B C /- D E* F -*(Where A=6, B=5, C=2, D=3, E=4 and F=1)

→ Algorithm to convert the infix expression to postfix expression.

1) Start
2) Scan one character at a time of an infix expression from left to right.
3) Opstack = the empty stack
4) Repeat till there is data in infix ~~exp~~ expression.
4.1) If scanned character is '(' then push it to opstack.
4.2) If scanned character is operand then push it to poststack.
4.3) If scanned character is operator then.
if(opstack != -1)
while (predence(opstack[Otos]) > precedence (scan character) then
pop & push it into poststack.
Otherwise
push scanned character into opstack
4.4) If scanned character is ')' then
pop & push into poststack until '(' is not found & ~~give~~ ignore both symbols.
5) pop & push into poststack until opstack is not empty
6) Stop.

Given, Express. = ABC/-DE*F-*

(A=6, B=5, C=2, D=3, E=4 & F=1)

Scan character	Value	operator 2	operator 1	Result	Vstack.
A	6				6
B	5				6, 5
C	2				6, 5, 2
/		2	5	5/2 = 2.5	6, 2.5
-		2.5	6		3.5
D	3				3.5, 3
E	4				3.5, 3, 4
*		4	3	12	3.5, 12
F	1				3.5, 12, 1
-		1	12	11	3.5, 11
*		11	3.5	38.5	38.5

Thus the output expression is 38.5 ✓

a) What is infix, prefix and postfix expression? Convert the following infix expression into postfix expression showing the content of stack in each step. 7

P = A + (B / C - (D * E $ F) + G) * H

→ Infix expression :- It is a ordinary mathematical notation of expression where operator is written in between the operands. eg :- A+B, there + is an operator & 'A' & 'B' are called operands.

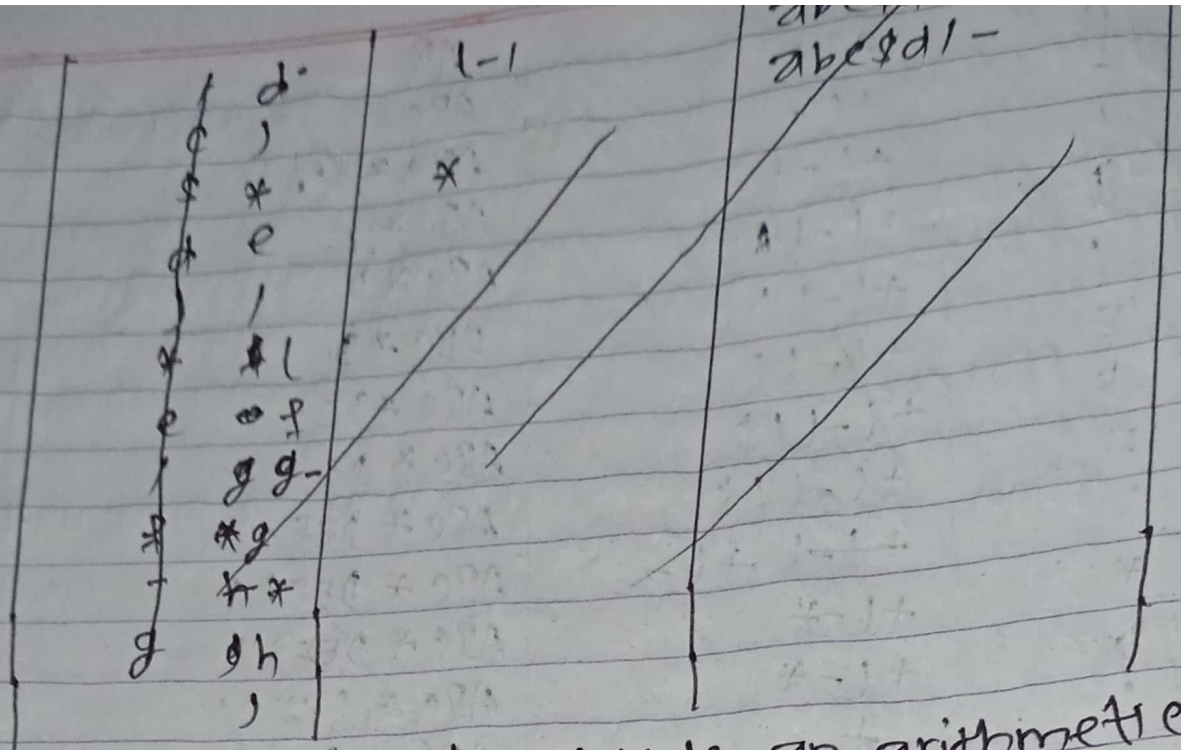
Prefix expression :- In prefix expression the operator precedes the two operands. That is the operator is written before the operands. It is also called Polish

expression. eg: The equivalent prefix expression of given infix expression A+B is +AB

postfix expression : In postfix expression the operators are written after the operands. So, it is called the postfix expression. In this expression the operator follows the two operands.

eg: The equivalent postfix expression of given infix expression A+B is *AB+.

Scanned character	OPstack	Poststack
	(	
A	(	A
+	(+	A
B	(+(	AB
/	(+(/	AB
C	(+(/	ABC
-	(+(/-	ABC/
D	~~(+(/-~~ (+(-	ABC/D
*	(+(-(*	ABC/D
E	(+(-(*	ABC/DE
$	(+(-(*	ABC/DE$
F	(+(-(*	ABC/DE$F
)	(+(-	ABC/DE$F*
+	(+(+	~~ABC~~ ABC/DE$F*-
G	(+(+	ABC/DE$F*-G
)	(+	ABC/DE$F*-G+
*	(+*	ABC/DE$F*-G+
H	(+*	ABC/DE$F*-G+H
)		ABC/DE$F*-G+H*+



write an algorithm to evaluate an arithmetic expression in postfix string. Apply the algorithm to evaluate : AB + C - BA + C$ - (Assume A=1, B=2, C=3).

→ Algorithm to evaluate the postfix expression

Here we use only one stack called Vstack (value stack).

1) Scan one character at a time from left to right of give postfix expression.

1.1) if scanned symbol is operand then
- read it's corresponding value & push it into Vstack.

1.2) if scanned symbol is operator then.
- pop & place into Op2 operator 2.
- pop & place into Op1 operator 1.
- compute result according to given operator & push result into Vstack.

2) pop & display which is required value of the given postfix expression.

3) stop.

2nd part,

Given, Expression = AB+C-BA+C$-

Now, (A=1, B=2, C=3)

scan character	Value	operator 2	operator 1	Result	Vstack
A	1	–	–	–	1
B	2	–	–	–	1,2
+	+	2	1	3	3
C	3	–	–	–	3,3
–	–	3	3	0	0
B	2	–	–	–	0,2
A	1	–	–	–	0,2,1
+	+	1	2	3	0,3
C	3	–	–	–	0,3,3
$	$	3	3	27	0,27
–	–	27	0	27	27

Explain how do you use stack to convert the following infix expression into postfix:

A+(B*C-(D/E^F)*G)*H , where ^ is power operator.

→ let there two stak opstack & poststack are used & otos & potos represent the opstack top & poststack top respectively. The opstack is used to store operators of given expression & poststack is used to store converted corresponding

Write the advantages of Postfix expression over the Infix expression. 7
Convert the given expression into Postfix expression showing the content of stack at each step. (A+B*C/D)+E*F-(G*H+I-J).

→ The advantages of postfix expression over the Infix expression.

The given expression,

(A+B*C/D)+E+F-(G*H+I-J).

scanned character	opstack	poststack
(	(	
	((	

A		A
+	((+	A
B		AB
*	((+ *	AB
C		ABC
/	((+ /	ABC*
D		ABC*D
)	(	ABC*D/+
+	(+	
E		ABC*D/+E
*	(+ *	ABC*D/+E
F		ABC*D/+EF
-	(-	ABC*D/+EF*+
(	(- (	
G		ABC*D/+EF*+G
*	(- (*	
H		ABC*D/+EF*+GH
+	(- (+	ABC*D/+EF*+GH*
I		ABC*D/+EF*+GH*I
-	(- (-	ABC*D/+EF*+GH*I+
J		ABC*D/+EF*+GH*I+J
)	(-	ABC*D/+EF*+GH*I+J-
)		ABC*D/+EF*+GH*I+J--

what are the difference beween stack & queue ? write ~~en~~ enqueue & dequeue algorithm of circular queue.

→ The difference between stack & queue are,

Stack	Queue
i) Object are inserted & removed at the same time. end.	i) Object are inserted & removed from different end.
ii) In stack only one pointer is used. It points to the top of the stack.	ii) In queue two different pointer are used for front & rear ends.
iii) Stack follow the last in first out (LIFO) manner.	iii) Queue follow the first in first out (FIFO) order.
iv) Insert operation is called push operation.	iv) Insert operation is called enqueue operation.
v) Delete operation is called pop operation.	v) Delete operation is called dequeue operation.
vi) In stack we maintain only one pointer to access the list. (I → IV)	vi) In queue we maintain two pointer to access the list.
vii) Stack is used in solving problem works on recursion.	vii) Queue is used in solving problem having sequential processing.
viii) Stack are visualized as vertical collection.	viii) Queue are visualized as horizontal collection.
ix) eg: collection of dinner plates at a wedding reception.	ix) eg: people standing in a line to board a bus.

Algorithm for circular queue

(#) Enqueue (Insertion)

1. Check queue full condition
   if (front == (rear+1) % MAXSIZE)
   print Queue is full & exit
2. else
   rear = (rear+1) % MAXSIZE
   queue [rear] = value
3. Stop

(#) Dequeue (Delete)

1. Check empty condition,
   if (front == rear)
   print queue is empty & exit
2. ~~else~~ else
   ~~pri~~ front = (front+1) % MAXSIZE
   item = queue [front]
3. Stop.

What is doubly linked list (DLL) & circular linked list (CLL)? Represent the following DO

Define Queue. Mention the primitive operations of Queue & write the module for enqueuer & dequeuer of in circular queue.

→ Queue : A Queue is an ordered collection of items from which items may be deleted at one end (called the front of the queue) & into which items may be inserted at the other end (the rear of the queue).

The primitive operations of Queue are:

① Make Empty (q) : To make q as an empty queue.

② Enqueue (q, x) : To insert an item x at the rear of the queue. This is also called by names add, insert.

(iii) Dequeue (q) : To delete an item from the front of the queue q. This is also known as delete, Remove.

iv) IsFull(q) : To check whether the queue q is full.

v) IsEmpty (q) : To check whether the queue q is empty.

vi) Traverse (q) : To read entire queue that is display the content of the queue.

write algorithm for push & pop operations on a stack using linked list implementation.

→

Algorithm of push operation:

1. Input the data to be ~~publish~~ pushed
2. create a New node
3. New node -> info = data
4. New node -> next = top
5. Top = New node
6. Exit

Algorithm

pop operation

1) If (top == NULL)
   1.1) display "The stack is empty"
   1.2) Exit
2) Temp = top;
3) display "The popped element is top -> info";
4) top = top -> next;
5) temp -> next = NULL;
6) free the temp node;
7) exit

push operation :- Suppose, Top is a pointer, which is pointing towards the topmost element on stack. Top is NULL when stack is empty. Data is the data item to be ~~published~~ pushed.
item

1. Input the data to be pushed:
2. create a New node

3) Newnode -> info = data
4) Newnode -> next = top
5) TOP = Newnode
6) Exit

Define list. List down the operations performed in list. Explain dynamic implementation of list with suitable example.

→ A list is a collection of homogeneous set of elements or objects.

Operations performed in list

Let L be a list & p indicates the position of the list L, then the following basic operation can be performed on the list L.

* create (L) : create a list L.
* Insert (x, p, L) : Insert x at position p in the list L.
* Find (x, L) : Return the position of the first occurrence of element x in list L.
* Retrieve (p, L) : Return the element at position p in list L.
* Delete (p, L) : Delete the element at position p of list L.
* Next (p, L) : Return the element at position p+1 in list L.

* previous(p,L) :- Return the element at position p-1 in list L.

* Makeempty(L) :- Make a list L an empty list.

* First(L) :- Return the first position on list L.

* print(L) :- Print the element of list L.

Dynamic Implementation of list :-