(1) Given a linked list whose typical node consists of INFO and LINK field. Formulate an Algorithm which will count the number of nodes in list.

(2) Formulate an Algorithm that will change the INFO field of the k^{th} node to the value given by y.

(3) Given a simple linked list whose first node is denoted by pointer variable FIRST. It is required to split this list into simply linked lists. The node denoted by pointer variable split is to be first element in the second list. Formulate a step by step algorithm to program this task.

(4) Formulate an Algorithm which concatenates a linear linked list to another linked list.

(5) Formulate an Algorithm which will ~~remove~~ reverse a singly linked list. Assume a typical node consist of an INFO & LINK field. The parameter to the function should be pointer to the Original list, and the function should return a pointer to the Reversed list.

(1) FIRST = It is a pointer variable which points first node of the linked list.

Temp = It is a temporary node to store data.

INFO = It is a part of node to store data.

LINK = It points to the next node.

count = It is counter variable to count the node and the it's initial value is zero.

→ function:- count (struct node * FIRST),

(1) IF LINK(FIRST) = NULL
THEN RETURN (count = 1).
(2) ELSE TEMP ← FIRST.
(3) LOOP: while(Temp != NULL)
(4) count ++.
(5) Temp = LINK(Temp).
(6) END LOOP.
(7) RETURN (count).

(2) FIRST = It is a pointer variable which points first node of the linked list
Temp = It is a temporary node to store data.
INFO = It is a part of node to store data.
LINK = It points to the next node
count = It counts the no. of Node (initia
k = It shows the potition of Node

→ function :- change (struct node *~~Node~~ FIRST)

(1) Temp ← FIRST.
(2) LOOP: while (Temp != NULL)
(3) count ++.
(4) START IF (count == k).
(5) THEN INFO(TEMP) = Y
(6) Return (FIRST)
(7) ELSE TEMP ← LINK (Temp).
(8) END LOOP:
(9) IF (count < k)
(10) Then write (k^{th} node is not Available

(2) FIRST = It is a pointer variable which points first node of the linked list.
Temp = It is a temporary node to store data
INFO = It is a part of node to store data.
LINK = It points to the next node
count = It counts the no. of Node (initial=0)
k = It shows the potition of Node

→ function :- change (struct node * ~~Node~~ FIRST)

(1) Temp ← FIRST.
(2) LOOP: while (Temp != NULL)
(3) count ++
(4) START IF (count == k)
(5) THEN INFO(TEMP) = Y
(6) Return (FIRST)
(7) ELSE TEMP ← LINK (Temp)
(8) END LOOP:
(9) IF (count < k)
(10) Then write (k^{th} node is not Available).

(3) FIRST = It is a pointer variable which points first node of linked list.
Temp = It is a temporary node to store data
INFO = It is a part of node to store data
LINK = It points to the next node
SPLIT = It points a node in a linked list.

→ function: split (Struct node * FIRST)

(1) Temp ← FIRST.
(2) LOOP: While (Temp != NULL)
(3) IF (LINK(TEMP) = SPLIT)
(4) THEN LINK(TEMP) = NULL.
(5) RETURN (FIRST), (SPLIT)
~~(6) RETURN (SPLIT).~~
(6) END IF.
(7) END LOOP.
(8) Write (SPLIT node not found.)
(9) IF (FIRST == SPLIT)
(10) RETURN (SPLIT).

(4) FIRST = It is a pointer variable which points first node of linked list.
Temp = It is a temporary node to store data
INFO = It is a part of node to store data
LINK = It points to the next node
SECOND = It is a pointer variable which points a node in a linked list.

→ function : concate (struct node * FIRST)

(1) Temp ← FIRST.
(2) LOOP : While (Temp != NULL)
(3) Temp = LINK (Temp).
(4) END LOOP :
(5) LINK (Temp) ← SECOND.
(6) RETURN (FIRST).
(7) IF (FIRST == NULL)
(8) THEN RETURN (SECOND).
(9) IF (SECOND == NULL)
(10) THEN RETURN (FIRST).

(5) FIRST = It is a pointer variable which points first node of a linked list
INFO = It is data part of node.
LINK = It points next node.
Temp = It is a temporary pointer variable to point nodes.
PREV = It points previous node
next = It points the address of next node.

→ function :- Reverse (struct node * FIRST)

(1) IF (FIRST == NULL)
(2) THEN WRITE (Linked list is NULL).
(3) ELSE IF (LINK(FIRST) = NULL)
(4) THEN RETURN (FIRST).
(5) temp ← FIRST.
(6) PREV ← NULL.
(7) LOOP: While (temp != NULL)
(8) next ← LINK(Temp).
(9) LINK(Temp) ← PREV.
(10) PREV ← Temp.
(11) Temp ← next.
(12) End LOOP:
(13) FIRST ← PREV.
(14) Return (FIRST).

Tutorial-3

(1) Consider the following c code:

```
int f(int x)
{
    int y;
    if (x==0)
        return 1;
    else
    {
      y = 2 * f(x-1)
      return y+1;
    }

}
```

→ Show how a call to function f(2) will be executed with the help of program stack. Show the push and pop sequence of all activation records.
What will be the final result f(2)?

~~Ans:~~

(2) Explain any three difficulties that can arise if any infix expression is evalution by computer.

(3) convert the following INFIX expression to their postfix and prefix equivalent

(a) A + B * C + D
(b) (A+B) * (C+D)
(c) A * B + C * D
(d) A + B + C + D
(e) (A+B) * C - (D-E) * (F+G)

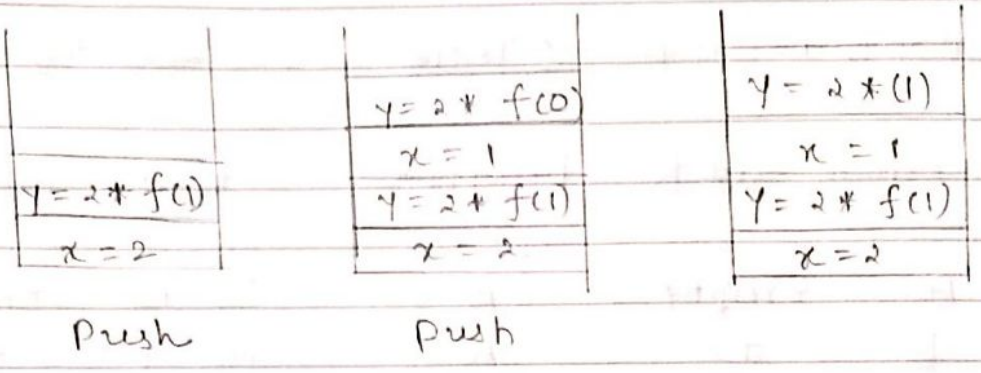
(4) consider the following INFIX expressions-

(a) (4+8) * (6-5) / ((3-2) * (2+2))
(b) (((1+2) * 3) + 6) / (2+3)

convert the above expression to postfix notation by following the Algorithm. Also, evaluate the postfix expression by following the algorithm. Clearly show the content of the stack and output string in each iteration.

Answers :-

(1)



(2)

(i) Infix expressions are harder to evaluate by computer because computer has to decide the precedence of operators.

(ii) Second difficulty is that if operators with same precedence come to the picture than computer has to decide "Associativity" of Operators.

(3)

(a) A + B * C + D. (Infix)

Expression	Stack	Postfix
A	Empty	A
+	+	A
B	+	AB
*	+ *	AB
C	+ *	ABC
+	+	ABC * +
D	+	ABC * + D
	Empty	ABC * + D +

Expre.	stack	Prefix
D	Empty	D
+	+	D
C	+	DC
*	+ *	DC
B	+ *	DCB
+	+ +	DCB *
A	~~+ +~~	DCB * +
	Empty	DCB * +

Postfix Expression :- ABC * + D +.
Prefix Expression :- + + A * BCD.

(b)

Expression	Stack	Postfix
C	C	-
A	C	A
+	C +	A
B	C +	AB
)	Empty	AB +
*	*	AB +
C	* C	AB +
c	* (	AB + C.
+	* C +	AB + C.
D	* C +	AB + CD.
)	*	AB + CD +
	Empty	AB + CD + *

Expre	Stack	Prefix
)	)	-
D	)	D
+	) +	D
C	) +	DC
C	Empty	DC +
*	*	DC +
)	*)	DC +
B	*)	DC + B
+	*) +	DC + B
A	*) +	DC + BA
(	*	DC + BA +
	Empty	DC + BA + *

Postfix expression :- AB + CD + * Prefix :- * + AB + CD.

(c) A * B + C * D.

Expression	stack	Postfix	Expre.	stack	prefix
A	Empty	A	D	Empty	D
*	*	A	*	*	D
B	*	AB	C	*	DC
+	+	AB *	+	+	DC *
C	+	AB * C.	B	+	DC * B
*	+*	AB * C	*	+*	DC * B
D	+*	AB * CD.	A	+*	DC * BA
	Empty	AB * CD * +		Empty	DC * BA * +

Postfix expression :- AB * CD * +.
Prefix expression :- +* AB * CD.

(d)

Expression	stack	Postfix	Expre.	Stack	prefix.
A	Empty	A	D	Empty	D
+	+	A	+	+	D
B	+	AB	C	+	DC
+	+	AB +	+	++	DC +
C	+	AB + C.	B	++	DC + B
+	+	AB + C +	+	+++	DC + B +
D	+	AB + C + D	A	+++	DC + B + A
	Empty	AB + C + D +		Empty	DC + B + A + +

postfix expression :- AB + C + D +.
prefix expression :- ~~+ A + B + C D~~.
+++ ABCD.

(e) $(A+B)*C-(D-E)*(F+G)$

Expression	stack	Output	Expression	stack	Output
	~~Empty~~ (	-	)	)	-
A	(	A	G	)	G
+	(+	A	+	)+	G
B	(+	AB	F	)+	GF
)	Empty	AB+	(	Empty	GF+
*	*	AB+	*	*	GF+
C	*	AB+C	)	*)	GF+
-	-	AB+C*	E	*)	GF+E
(	-(	AB+C*	-	*)-	GF+E
D	-(	AB+C*D	D	*)-	GF+ED
-	-(-	AB+C*D	(	*	GF+ED-
E	-(-	AB+C*DE	-	-	GF+ED
)	-	AB+C*DE-	C	-	GF+ED-
*	-*	AB+C*DE-	*	-*	GE+ED-
C	-*(	AB+C*DE-	)	-*)	GF+ED-
F	-*(	AB+C*DE-F	B	-*)	GF+ED-
+	-*(+	AB+C*DE-F	+	-*)+	GF+ED-
G	-*(+	AB+C*DE-FG	A	-*)+	GF+ED-
)	-*	AB+C*DE-FG+	(	-*	GF+ED-
	Empty	AB+C*DE-FG+*-		Empty	GF+ED- +*-

postfix Expression :- AB+C*DE-FG+*-

prefix Expression :- -*+ABC*-DE+FG

(4)

(a) $(4+8)*(6-5)/((3-2)*(2+2))$

Expression	stack	Output
(	(	-
4	(	4
+	(+	4
8	(+	48
)	Empty	48 +
*	*	48 +
(	*(	48 +
6	*(	48 + 6
-	*(-	48 + 6
5	*(-	48 + 65
)	*	48 + 65 -
/	/	48 + 65 - *
(	/(	48 + 65 - *
(	/((	48 + 65 - *
3	/((	48 + 65 - * 3
-	/((-	48 + 65 - * 3
2	/((-	48 + 65 - * 32
)	/(	48 + 65 - * 32 -
*	/(*	48 + 65 - * 32 -
(	/(*(	48 + 65 - * 32 -
2	/(*(	48 + 65 - * 32 - 2
+	/(*(+	48 + 65 - * 32 - 2
2	/(*(+	48 + 65 - * 32 - 22
)	/(*	48 + 65 - * 32 - 22 +
)	/	48 + 65 - * 32 - 22 + *

= 48 + 65 - * 32 - 22 + * /.

Expression	stack	Output
)	)	-
)	))	-
2	))	2
+	))+	2
2	))+	22
(	)	22 +
*	)*	22 +
)	)*)	22 +
2	)*)	22 + 2
-	)*)-	22 + 2
3	)*)-	22 + 23
(	)*	22 + 23 -
(	Empty	22 + 23 - *
/	/	22 + 23 - *
5	/)	22 + 23 - *
5	/)	22 + 23 - * 5
-	/)-	22 + 23 - * 5
6	/)-	22 + 23 - * 56
(	/	22 + 23 - * 56 -
*	*	22 + 23 - * 56 - /
)	*)	22 + 23 - * 56 - /
8	*)	22 + 23 - * 56 - / 8
+	*)+	22 + 23 - * 56 - / 8
4	*)+	22 + 23 - * 56 - / 84
(	*	22 + 23 - * 56 - / 84 +

= 22 + 23 - * 56 - / 84 + *

= * + 48 / - 65 * - 32 + 22

∴ Postfix expression :- 48+65-*32-22+*/

∴ Prefix expression :- */+48/-65*-32+22

(b)

Expression	Stack	Output	Expression	Stack	Output
(	(	-	)	)	-
(	((	-	3	)	3
(	(((	-	+	)+	3
1	(((	1	2	)+	32
+	(((+	1	(	Empty	32+
2	(((+	12	/	/	32+
)	((	12+	)	/)	32+
*	((*	12+	6	/)	32+6
3	((*	12+3	+	/)+	32+6
)	(	12+3*	)	/)+)	32+6
+	(+	12+3*	3	/)+)	32+63
6	(+	12+3*6	*	/)+)*	32+63
)	Empty	12+3*6+	)	/)+)*)	32+63
/	/	12+3*6+/	2	/)+)*)	32+632
(	/(	12+3*6+/	+	/)+)*)+	32+632
2	/(	12+3*6+/2	1	/)+)*)+	32+6321
+	/(+	12+3*6+/2	(	/)+)*	32+6321+
3	/(+	12+3*6+/23	(	/)+	32+6321+*
)	/	12+3*6+/23+	(	/	32+6321+*/
	Empty	12+3*6+/23+/		Empty	32+6321+*/

Postfix expression :- 12+3*6+/23+/

Prefix expression :- */+*+1236+23.

Tutorial - 2

(1) Formulate an algorithm which will perform an insertion to the immediate left of the k^{th} node in the list.

(2) You're given the pointer to the head node of a doubly linked list. Reverse the order of the nodes in the list. The head node might be null to indicate the list is empty.

(3) You're given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order. Delete as few nodes as possible so that the list does not contain any value more than once. The given head pointer may be null indicating that the list is empty.

(4) You're given the pointer to the head node of a sorted doubly linked list and an integer to insert into the list. Create a node and insert it into the appropriate position in the list. The head node might be null to indicate that the list is empty.

(5) You're given the pointer to the head nodes of two sorted linked list. The data in both lists will be sorted in ascending order. Change the next pointer

to obtain a single, merged linked list which also has data in ascending order. Either head pointer given may be null meaning that the corresponding list is empty.

Answers: head = It is a pointer which points to the first node of the link list.

(1) temp = It is a temporary pointer to store address of nodes.
K = It is a k^{th} node of the link list.
INFO = It is a data part in a node.
LINK = It is a pointer which points next node in the linked list.
NEW = It is a new node.
AVAIL = It is a Available free node in memory.

InsertFirst (struct Node * head, x)

(1) [IS Linklist empty ?]
IF head = NULL.
Then write ("Link list is empty"), Return
(2) NEW ← AVAIL.
AVAIL ← LINK(AVAIL)
INFO (NEW) ← x.
(3) [IS LINKLIST containing only one node]
IF head = k.
Then LINK (NEW) ← head
NEW ← head
Return (head).
(4) temp ← head.
(5) Repeat while link(temp) != k || link(temp) != NU
temp ← link (temp)
(6) IF link (temp) = NULL.
Then write ("Node not found").
Return
(7) LINK (NEW) ← LINK (temp)
LINK (temp) ← NEW. (8) Return (head).

head = It is a pointer which points to the first node of the link list.

temp, temp1, temp2 = They are temporary pointers which holds the address of nodes.

LPTR = It points the left node in the link list.

RPTR = It points the right node in the link list.

Reverse (struct node * head).

(1) [Is Linklist empty ?].
IF head = NULL.
Then write ("Link list is empty"), Retu
(2) temp ← head
(3) Repeat while LINK(temp) != NULL.
temp ← LINK(temp).
(4) temp2 ← temp.
(5) [Reversing both the links]
Repeat while (temp) != head.

RPTR (temp1) ← RPTR (temp)
RPTR (temp) ← LPTR (temp)
LPTR (temp) ← RPTR (temp1)

temp ← RPTR (temp).

(5) head ← temp 2
(6) Return (head).

(3) head = It is a pointer which points to the first node of the link list.
temp = It is a temporary pointer which hold the address of node.
INFO = It is a data part of a node.
LINK = It is pointer which points next node in the linklist.

Delete Repeat (struct Node *head).

(1) [Is Linklist empty ?]
IF head = NULL.
Then write ("Link list is empty").
Return.
(2) ~~[Is Link list containing only one node]~~

(2) temp ← head.
(3) Repeat while link(temp) != NULL.
IF (INFO(temp) == INFO (LINK (temp))
Then LINK(temp) ← LINK (LINK(temp))

ELSE temp ← LINK(temp).

(4) Return (head).

(4) head = It is a pointer which points the first node in the linklist.
temp = It is a temporary pointer to store address of node.
INFO = It is a data part in the node
RPTR = It is a pointer which points right most node.
LPTR = It is a pointer which points left most node.
Pred = It is a temporary pointer which holds the address of node.

INSERTORDERED (struct Node * head).

(1) [Is Linklist empty ?]
IF head = NULL
Then write ("Linklist is empty").
Return.

(2) NEW ← AVAIL.
AVAIL ← RPTR (AVAIL)
INFO (NEW) ← x.

(3) [Insertion at first].
IF (INFO (head) ≥ INFO (NEW))
Then RPTR (NEW) ← head
LPTR (NEW) ← NULL.
LPTR (head) ← NEW.
~~NEW~~ ← head. ← new
Return (head).

(4) temp ← head.
pred ← head.

(ii) [Insertion at end or inbetween].

Repeat while temp != NULL.

IF (INFO(temp) ≤ INFO(NEW)).
Then pred ← temp
temp ← RPTR(temp).

ELSE IF (INFO(temp) ≤ INFO(NEW) && LINK(temp) = NULL).

Then RPTR(temp) ← new.
RPTR(NEW) ← NULL.
LPTR(NEW) ← temp.
Return (head).

Else RPTR(NEW) ← temp.
LPTR(NEW) ← pred.
RPTR(Pred) ← new.
LPTR(temp) ← new.
Return (head).

(5) [Insertion at end or Inbetween].

Repeat while temp != NULL.

IF (INFO(temp) ≤ INFO(NEW)).
Then pred ← temp.
temp ← RPTR(temp).

ELSE IF (INFO(temp) ≤ INFO(NEW) && LINK(temp) = NULL).

Then RPTR(temp) ← new.
RPTR(NEW) ← NULL.
LPTR(NEW) ← temp.
Return (head).

ELSE RPTR(NEW) ← temp.
LPTR(NEW) ← pred.
RPTR(pred) ← new.
LPTR(temp) ← new.
Return (head).

(5) head = It is a points which points to the first node in the link list.
temp, i, j = These are the temporary pointers which stores the address of nodes.
INFO = It is a data part of the node
LINK = It points to the next node in link list.
Begin = It is first node of 2nd link list.

Mergesort (head, Begin).

(1) [Is LINKlist Empty ?]
IF head = NULL.
Then head ← Begin.
Follow steps

(2) ELSE temp ← head
Repeat while LINK (temp) != NULL
temp ← LINK (temp).

(3) LINK (temp) ← Begin [Merge].

(4). FOR (i = head; LINK(i) != NULL; i ← LINK
FOR (j = LINK(i); LINK(j) != NULL; j ← LIN
IF (INFO(i) > INFO(j))
Then IF (LINK(i) = j)
Then LINK(i) ← LINK(j)
LINK(j) ← i
m ← i
i = j
j = m

```
        ELSE    LINK(i) ← LINK(j)
                LINK(i) ← LINK(j)
                m = i
                i = j
                j = m
        END IF
    END FOR
    END FOR.
```

(5) Return (head).

```
        Then    INFO(temp1) ← INFO(i)
                INFO(i) ← INFO(j)
                INFO(j) ← INFO(temp1)
        END IF.
    END FOR
    END FOR.
```

(5) Return (head).

Tutorial - 4.

Choose the appropriate data structure and write algorithm for converting a decimal integer to binary with proper reason and at the end display the result.

In a CD pack, the CDs are placed one over the other through a central axis. Assume that there are 35 CDs in a pack.

a. Name the data structure that resembles with the CD Pack.

b. Consider the CD pack as an array in C, how will you name the position of the last CD and what will be its value.

c. Write an algorithm to insert and remove the CD from the pack and name the operations performed.

While waiting for a ticket at a railway station ticket counter, you are following the principle as that of a data structure.

a. Name the data structure and the principle.

b. Write an algorithm to add new element in this data structure.

c. Name the situation when there is no space for adding the new element.

(4) Set of 50 books is available which consists of Java books and c++ books. There is only one book pack open from both the ends, of size 50 which is empty. Books are to be arranged in a book pack in such a way that Java books and c++ books are segregated (condition: At a time only one book can be selected).

a. Name the data structure that resembles the book pack.
b. Write an algorithm to segregate the books.
c. Use LIFO structure.

(5) Is it possible to implement a queue using two stacks? Justify your answer.

Answers.

(1) x = Any Decimal Number
Top = It is a top element of a stack s.

function – Binary (x).

(1) START. (2) TOP ← -1 [Initialization].
(3) Repeat while (x > 0).
S[TOP] ← x % 2
x ← x/2.
~~(4) Return (5).~~
(4) Repeat while (top != -1)
print ("S[TOP]")
top ← Top - 1
(5) End.

(2) There are 35 CDs are placed one over the other through a central axis.

(i) Stack Data structure resembles with the CD-Pack.

(ii) The Name of the position of the last CD will be C[TOP] and its value will give the name of last CD.

(iii) TOP = It is a top element of a stack C.
X = It is the name of the CD to be inserted in a pack.

Push (x).

(1) START.
(2) [Initialization]
Top ← -1.
(3) [Insertion].
Top ← Top + 1
S[TOP] ← X.
(4) Return (C)
(5) End.

Pop (X).

(1) START.
(2) [Is stack empty ?].
IF TOP ← -1
Then write ("Underflow").
Return.
(3) [Deletion].
Top ← Top - 1
(4) Return (C)
(5) End.

While waiting for a ticket at a Railway station is following the principle of "Queue" Data structure.

Principle: In Queue insertion can be performed

at one side and Deletion should from other side.

(ii) head = It is a pointer pointing to the first element of an queue
tail = It is a pointer pointing to the last element of an queue.
x = Element to be inserted.
q = Array of elements.
Max = Maximum elements that can be added in the queue.
Enqueue(x) -

(1) START.
(2) [Is queue full ?].
IF tail ← Max - 1
Then (Return)
(3) IF head ← tail ← -1
Then head ← tail ← 0,
q[head] ← x.
Return (q)
(4) ELSE tail ← tail + 1
q[tail] ← x.
Return (q).
(5) End.

(iii) When there is no space for adding a new element in queue.

(4)

(i) Here Book Pack is Open from both the ends of size 50. So, Book pack is Deque (Double ended queue).

(iii) Here, 50 Books are stored in a ~~queue~~ stack.

Head = It is a pointer pointing to the first element of book Pack.
tail = It is a pointer pointing to the last element of book pack.
Java = It is a Java book.
C++ = It is a C++ book.
Top = Top element of a stack S.
q = Array of elements of a deque

⇒ Seperate ~~(S,q)~~ (S, q[]) —

(1) START.
(2) [check for the first book in a stack].
IF (S[TOP] == Java)
Then
(2) [empty deque ?].
IF head ← tail ← 1
Then head ← tail ← 0
q[head] ← S[TOP].
Top ← Top - 1
(3) [check for the first element in a deque].
IF (q[head] == Java)
Then
IF (S[TOP] == Java)
enqueue_head (S[TOP]).

ELSE enqueue_tail (s[TOP]).

(4) ELSE IF (q[head] == C++)
Then
IF (s[TOP] == C++)
Then enqueue_head (s[TOP]).
ELSE enqueue_tail (s[TOP])

(5) Return (q).
(6) End.

(iii) head 1 = It is a head pointer pointing to the first element of a single ended queue.
tail 1 = It is a tail pointer pointing to the last element of a single ended queue.
Java = It is a Java book.
C++ = It is a C++ book.
q_1 = Array of elements of a queue.
q_2 = Array of elements of a deque.
head 2 = head pointer of a deque.
tail 2 = tail pointer of a deque.

⇒ Seperate (q_1[], q_2[]).

(1) START.
(2) [empty deque ?]
IF head 2 ← tail 2 ← -1.
Then ~~q2~~[head 2 ← tail 2 ← 0
q_2[head 2] ← ~~pop~~ dequeue (q_1)

(3) [check for a first element in a deque]
IF (q_2 [head$_2$] == Java)
Then
IF (dequeue (q_1) == Java)
Then enqueue_head (dequeue(q_1)).
ELSE
enqueue_tail (dequeue (q_1)).

(4) ELSE (q_2 [head$_2$] == c++).
Then
IF (dequeue(q_1) == c++)
Then enqueue_head. (dequeue (q_1))
ELSE
enqueue_tail (dequeue (q_1))

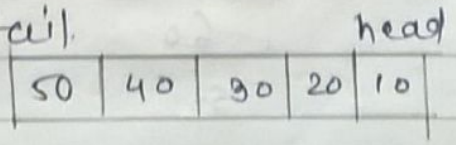
(5) Return. (q_2).
(6) End.

(5)

→ Yes, It is possible to implement queue using two stacks.

e.x. We have some elements in a queue such as, 10, 20, 30, 40 and 50.

→ First of all, We will push them in a stack 1.



50	TOP
40	
30	
20	
10	

→ Now, I want to insert an element "60" in a queue. So, I can directly push that element in a stack.

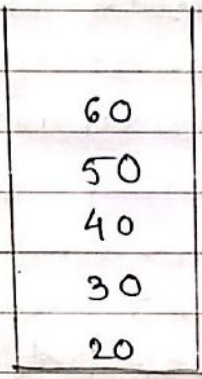
60	Top.
50	
40	
30	
20	
10	

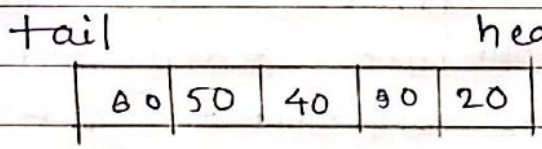
→ But If we want to delete an element from queue, it should be deleted from the head only. So, I'll use second stack and pop elements from S_1 and push them in stack S_2.

S_1	S_2	
	10	TOP
	20	
	30	
	40	
20	50	
10	60	

→ Then POP ⇔ an element from 32, 50, It will delete an element from head in a queue.

→ Then transfer all the elements in a stack and then in a queue.





queue.

Tutorial - 5.

(1) Give non-recursive algorithm that reverses a singly linked list of n elem An Algorithm should use no more than constant storage beyond that needed for the list itself.

(2) Identify the data structure and write an algorithm for addition of two quadratic equation that results third quadratic equation.

(3) The web-browser allows viewing a particular web-page using BACK and FORWARD button using the cache. Find appropriate data structure and write an algorithm for this task

(4) Describe an Algorithm which perform swapping two consecutive nodes in a singly linked list.

(5) To divide a circular linked list into two almost equal sized circular linked list in a single pass. Maintaining sequence of nodes in resultant lists is not necessary.

(1) temp, next, pred } temporary pointer variable which store the address in a memory.

head = head pointer pointing to the head node

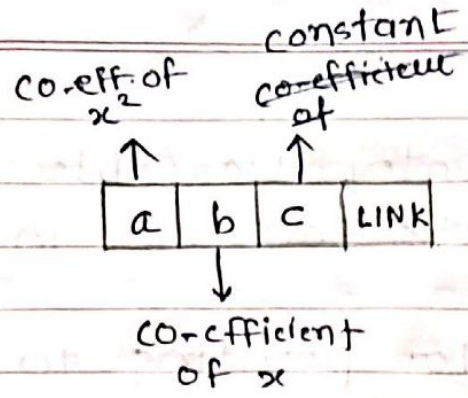
INFO = Info part of a node.

LINK = Stores the next address in a link-list.

function :- Reverse (struct node * head).

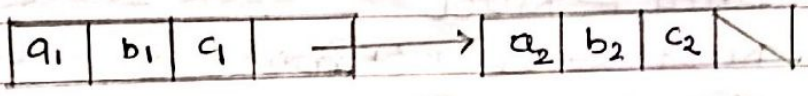
(1) START
(2) temp ← head.
(3) pred ← NULL
    Next ← LINK(temp).
(4) [Is only one node present?].
    IF (LINK(head) == NULL)
    Return head.
(5) While ((temp) != NULL)
    pred ← temp.
    temp ← head.
    next ← LINK(temp).
(6) head ← Pred.
(7) Return (head).
(8) End.

(2)



quadratic eq. $= ax^2 + bx + c$.

data 1 = It stores the co-efficient of x^2.
data 2 = It stores the co-efficient of x.
data 3 = It stores constant.
LINK = It is a pointer pointing to the next node in a Link-list.
head = It is a pointer pointing to the head node of a link-list



(1) START.
(2) temp ← LINK(head)
(3) [Addition of data]
data 1(head) ← data 1(head) + data 1(temp)
data 2(head) ← data 2(head) + data 2(temp)
data 3(head) ← data 3(head) + data 3(temp)

(4) LINK(head) ← NULL.
(5) Return (head).
(6) End.

(3) Homepage: It is a homepage of a browser.
Current-page: It is a Opened current page in a browser.
LPTR: It is a pointer, pointing to the previous URL in a browser.
RPTR: It is a pointer, pointing to the next URL (newURL) in a browser.

function: back-forward (Homepage)

(1) START.
(2) [only home-page is Open].
IF (RPTR (homepage) == NULL)
Then Return (homepage).
(3) RPTR (currentpage) = NULL.
(4) IF (NewURL != NULL)
Then RPTR(currentpage) ← New URL.
LPTR(newURL) ← CurrentURL
(5) IF (Back button is pressed)
Then currentpage ← LPTR(currentpage)
(6) ELSE IF (FORWARD button is pressed)
Then currenpage ← RPTR(currentpage)
(7) End.

temp ← It is a temporary pointer, pointing to the node.

next = It is a pointer pointing to the next node in a link-list.

head = It is pointer pointing to the head node of a link-list.

LINK = It is link part of a node.

function: Swap (struct node * head).

(1) START.
(2) temp ← head
    next ← LINK(temp).
(3) Repeat while throu 4 step. (next != NULL)
    LINK(temp) ← LINK(next)
    LINK(next) ← temp.
(4) temp ← LINK (temp).
    next ← LINK(temp).
(5) Return (head)
(6) End.

head = It is a head node of a link-list.

head1 = It is a head node of a new circular link-list.

head2 = It is a head node of second new circular link-list.

temp, temp1, temp2 = They are temporary pointer variables pointing to the node

count = It is a variable.

LINK = It is a link part of a node.

function: seperate (struct node * head).

(1) START.
(2) temp ← head. [Initialization].
temp1 ← temp.
temp2 ← LINK(temp).
~~head1 ← tem~~
LINK (temp1) ← NULL.
LINK (temp2) ← NULL.
head 1 ← temp 1
head 2 ← temp 2
temp ← LINK (LINK (temp).
count ← 3.
(3) Repeat thyou 4 steps while (temp! =head)

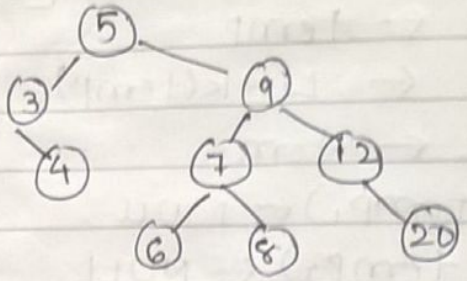
IF (count % 2 ! =0)
Then LINK (temp1) ← temp.
temp1 ← LINK (temp1)
LINK (temp1) ← NULL
temp ← LINK (temp)
count ← count + 1.

(4) ELSE IF (count % 2 = 0)
Then LINK (temp2) ← temp.
temp2 ← LINK (temp2)
LINK (temp2) ← NULL.
temp ← LINK (temp)
count ← count + 1.

(5) LINK (temp1) ← head 1.
LINK (temp2) ← head 2.
(6) Return head 1, head 2 (7) End.

Tutorial-6

(1) The binary search tree shown below was constructed by inserting a sequ of items into an empty tree.



which of the following input sequence will not produce this binary search tree:-

(a) 5 3 4 9 12 7 8 6 20
(b) 5 9 3 7 6 8 4 12 20
(c) 5 9 7 8 6 12 20 3 4
(d) 5 9 7 3 8 12 6 4 20
(e) 5 9 3 6 7 8 4 12 20

(2) Suppose the keys on the middle row a standard keyboard (A S D F G H J K L) inserted in succession into an init empty binary search tree. Draw th tree after this sequence of insert has been made.

(3) Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?

(a) 7 5 1 0 3 2 4 6 8 9
(b) 0 2 4 3 1 6 5 9 8 7
✓(c) 0 1 2 3 4 5 6 7 8 9
(d) 9 8 6 4 2 3 0 1 5 7

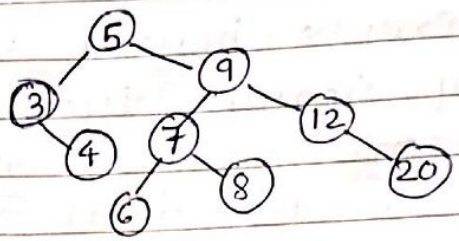
(4) The pre-order traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one ~~of~~ of the following is the post-order traversal sequence of the same tree.

(a) 10, 20, 15, 23, 25, 35, 42, 39, 30
(b) 15, 10, 25, 23, 20, 42, 35, 39, 30
(c) 15, 20, 10, 23, 25, 42, 35, 39, 30
✓(d) 15, 10, 23, 25, 20, 35, 42, 39, 30

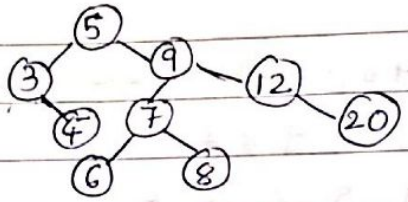
(5) The pre-order and In-order traversal of binary tree is A B D G C E H I F and D G B A H E I C F respectively then the post-order traversal will be.

Answers:

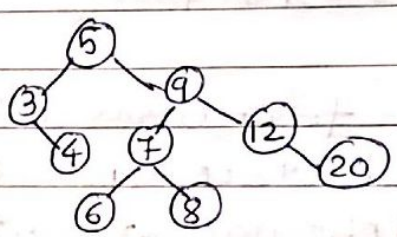
(1) (a)



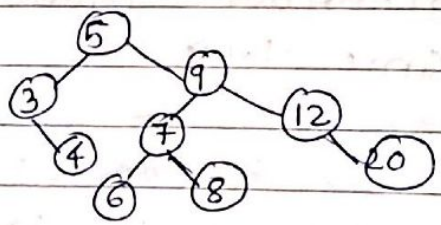
(b)



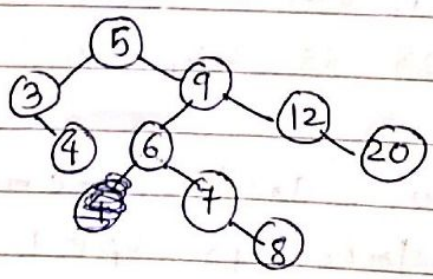
(c)



(d)

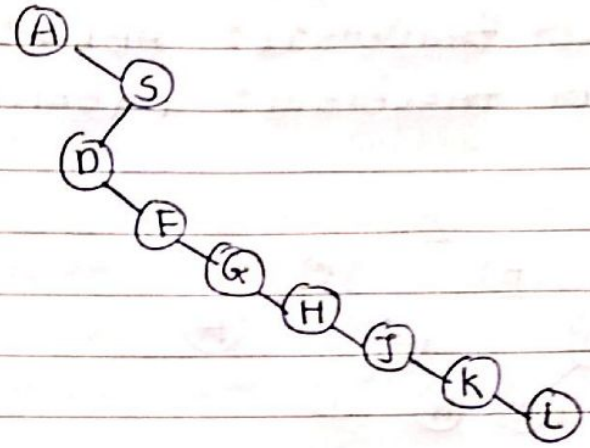


(e)

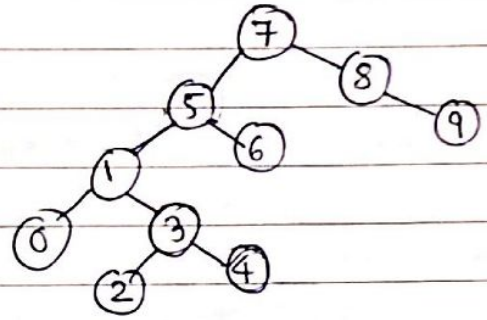


So, Ans. = (e)

(2)

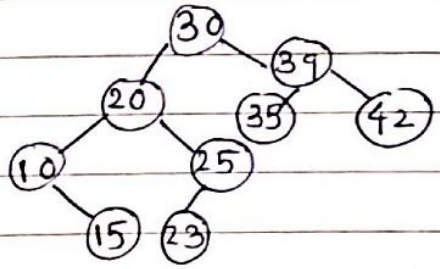


(3) Input: 7, 5, 1, 8, 3, 6, 0, 9, 4, 2



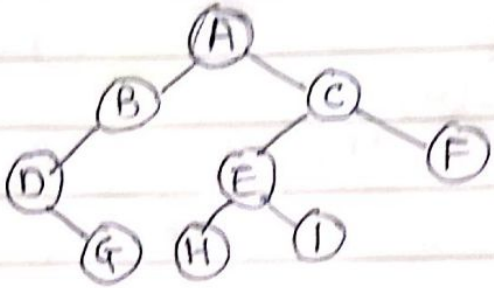
Inorder-traversal = 0 1 2 3 4 5 6 7 8 9.

(4) The Pre-order traversal : 30, 20, 10, 15, 25, 23, 39, 35, 42



Post-order traversal : 15 10 23 25 20 35 42 39 30

(5) The Pre-order traversal: ABDGCEHIF
The In-order traversal: DGBAHEICF



The postorder traversal: GDBHIEFCA