2nd IA DSA IMP Questions

1. What is singly linked list. Write a C function for following operations on singly linked list

i) Insertion at beginning ii) Insertion at end iii) Deletion at the beginning

iv) Deletion at end

SINGLE LINKED LIST:

Singly linked lists are the basic type of linked lists where each node has exactly one pointer field.

A singly linked list is comprised of zero/ more number of nodes when the number of nodes is zero, the list is empty otherwise if the linked list is non-empty, the list is pictorially represented as 1st node links to 2nd node and 2nd node links to 3rd node and so on. the last node has zero link whose value of address is set to NULL.

REPRESENTING SLL IN C LANGUAGE / MEMORY ALLOCATION

The following features are used to represent SLL. Use the following 3 steps to create a SLL.

1. Define node's structure
   To define a node, self-referential structures are used
2. Create a new nodemalloc() or
   MALLOC macro is used to allocate memory for the defined structures nodes of the size needed for structure node considered.
3. Removal of nodes
   At any point if the allocated nodes are not in use, they are removed by free( )

i) Insertion at beginning ii) Insertion at end iii) Deletion at the beginning

iv) Deletion at end

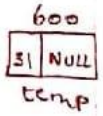
Implementation of SLL:

Structure definition:

```
struct node
{
    int info;
    struct node *next;
} *first = NULL, *temp = NULL, *last = NULL;
```

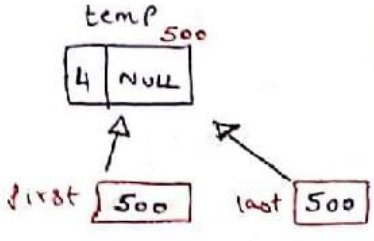
Create a Node:

```
void create()
{
   temp = (struct node *) malloc(sizeof(struct node));
   printf("Enter the information to be stored \n");
   scanf("%d", &temp->info);
   temp->next = NULL;
}
```

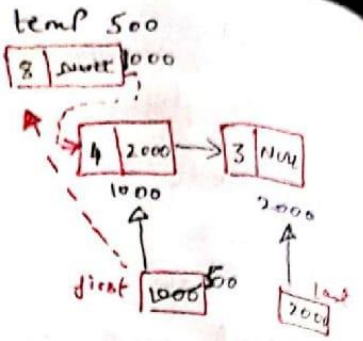


Insertion at the beginning (front)

```
void insertbeg()
{
    if (first == NULL)          //List is empty
    {
        create();
        first = temp;
        last = first;
    }
    else
```

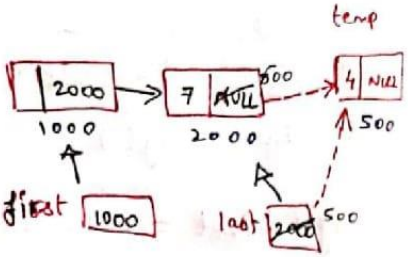


```
    {
        create();
        temp→next = first;
        first = temp;
    }
}
```



Insertion at the end (Rear)

```
Void insertEnd()
{
    if(first == NULL)    //List is empty.
    {
        create();
        first = temp;
        last = first;
    }
    else
    {
        create();
        last → next = temp;
        last = temp;
    }
}
```



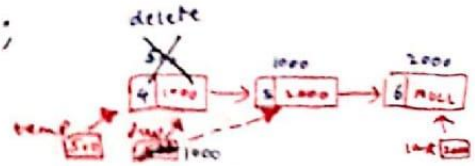
Display()

```
    void display()
    {   struct node *temp1;
```

```
templ = first;
if( templ == NULL)          // List is empty
{
      printf(" List is empty ");
      return;
}
Printf(" The contents of Linked List are: \n");
while(templ != NULL)        // Traverse through the list
  {
      printf(" %d \n", templ -> info);
       templ = templ -> next;
  }
}
```

Delete a node at the beginning (Front)

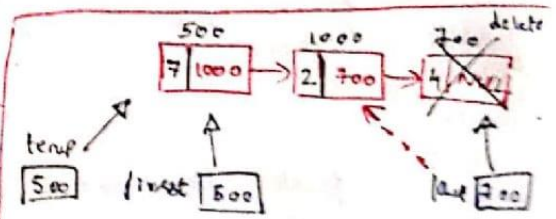
```
void deleteBeg()
{
   struct node *temp;
   temp = first;            // store address of 1st node in temp
   if( temp -> next == NULL) // only one node in list
   {
      printf(" The item deleted = %d \n", temp -> info);
       free(temp);
       first = last = NULL;
       return;
   }
   else
    {
       first = temp -> next; // make 2nd node as 1st node
       Pf(" The item deleted: %d \n", temp->info);
        free(temp);
    }
}
```



Deletion at the end (Rear)

```
void deleteEnd()
{
    struct node *temp;
    temp = first;
    if(temp->next == NULL)
    {
        printf("The item deleted = %d\n", temp->info);
        free(temp);
        first = last = NULL;
        return;
    }
    else
    {
        while(temp->next != last)
            temp = temp->next;
        printf("The Item deleted = %d\n", last->info);
        free(last);
        temp->next = NULL;
        last = temp;
    }
}
```

```
while(temp->next != last)
       1000  700  700
temp = temp->next;
       1000
```



2. Write a C function to

i. Concatenating singly linked lists.

ii.Finding the length of list

i.) **Function to concatenate two linked list pointed by list1 and list2**

```
NODE* concat(Struct NODE * list1,stuct NODE *list2)
{
struct NODE* temp;
if(list1==NULL)
return(list2)
if(list2==NULL)
return(list1);
temp=list1
while(temp->link!=NULL)
temp=temp->link;
temp->link=list2;
return(list1)
}
```

ii.Finding the length of list

Function to find the length of the the list

```
int Length(Node * first)
{
NODE *cur;
int count=0
if (first==NULL)
{
printf(“list is empty);
return(0)
}
cur=first ;
while(cur!=NULL)
{
count++;
 cur=cur->link;
}
return(count)
}
```

3. write C function to add two polynomials. show the linked list representation of below two polynomial and in addition

POLY1:5x2+4x+2, POLY2:3x2+2x+5

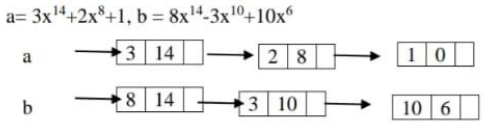


Figure 13: Representaion of a & b polynomials.

C Function for polynomial addition is given below:-

```
polyPointer padd(polyPointer a, polyPointer b)
{       /* return a polynomial which is the sum of a & b */
        polyPointer c, rear, temp;
        int sum;
        MALLOC(rear, sizeof(*rear));
        c=rear;
        while(a && b)
                switch(COMPARE (a->expon, b->expon))
                {
                        case -1:/* a->expon < b->expon */
                                attach(b->coef, b->expon, & rear);
                                b=b->link;
                                break;
                        case 0: /*a->expon == b->expon */
                                sum = a->coef + b->coef;
                                if(sum)
                                        attach(sum, a->expon, & rear);
                                a=a->link;
                                b=b->kink;
                                break;
                        case 1: /* a -> expon > b->expon */
                                attach(a->coef, a->expon, & rear);
                                a=a->link;
                }
                /* copy rest of the list a and then list b */
                for(;a;a->link)
                        attach(a->coef, a->expon, & rear);
                for(;b;b->link)
                        attach(b->coef, b->expon, & rear);
                rear->link = NULL;
```



```
                /* delete extra initial node */
                temp = c;
                c=c->link;
                free(temp);
                return c;
}
```

- The above function uses streaming process, that moves along the 2 polynomials, either copying terms directly / adding them to the result.
- Thus, while loop has 3 cases, depending on whether next pair of elements are =, < or >.
- To create a new node and append it to the end of c, the above addition function uses attach().

```
void attach( float coefficient, int exponent, polyPointer *ptr)
{       /* create a new node with coef = coefficient & expon = exponent, attach is to the node pointed to
by ptr. ptr is updated to point to this new nodes */
        polyPointer temp;
        MALLOC(temp, sizeof(*temp));
        temp->coef = coefficient;
        temp->expon = exponent;
        (*ptr)->link = temp;
        *ptr = temp;
}
```

4. discuss how to implement stacks using linked list. write c function and structure for it.exx

i. add to a linked list (push) , ii. delete from linked list (pop)

Implementing a stack using a linked list is a common and straightforward way to represent the stack data structure. Here is a step-by-step guide on how to implement a stack using a linked list in a general way, without focusing on a specific programming language:

1. Define the Node structure:

Start by defining a structure or class for the node in the linked list. Each node should have two components: data and a reference (or pointer) to the next node.

Node:

data

next

2. Initialize an empty stack:

Create a stack structure that keeps track of the top of the stack. Initially, the stack is empty, so the top is set to `null` (or equivalent).

Stack:

top = null

i. Push operation:

To push an element onto the stack:

- Create a new node with the given data.
- Set the `next` pointer of the new node to the current top of the stack.
- Update the top of the stack to the new node.

```
void push(int i, element item)
{/* add item to the ith stack */
   stackPointer temp;
   MALLOC(temp, sizeof(*temp));
   temp→data = item;
   temp→link = top[i];
   top[i] = temp;
}
```

ii. Pop operation:

To pop an element from the stack:

- If the stack is not empty:
  - Save the data of the top node.
  - Move the top pointer to the next node.

```
element pop(int i)
{/* remove top element from the ith stack */
   stackPointer temp = top[i];
   element item;
   if (!temp)
     return stackEmpty();
   item = temp→data;
   top[i] = temp→link;
   free(temp);
   return item;
}
```

5. Explain concept of sparse matrix representation using linked list. Represent the following sparse matrix in linked list format

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Sparse matrix representation using a linked list involves creating a linked list where each node stores information about a non-zero element in the matrix. Each node typically includes the row index, column index, and the value of the element. This representation is efficient for matrices with a large number of zero elements.

6. Define Binary tree with an example. write c recursive routine to traverse the given tree using inorder, preorder and post order

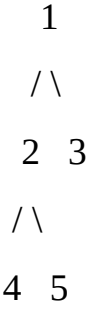
A binary tree is a hierarchical data structure where each node has at most two children, referred to as the left child and the right child. The topmost node in a binary tree is called the root, and nodes with no children are called leaves. Each node in a binary tree has a value and may or may not have left and right children.

Binary Tree:

A binary tree is a recursive data structure in which each node has at most two children, referred to as the left child and the right child.

Now, let's illustrate a binary tree with a simple example:

Consider the following binary tree:



In this example:

- The root node has the value 1.
- The left child of the root has the value 2, and its left child has the value 4.
- The left child of the root also has a right child with the value 5.
- The right child of the root has the value 3.

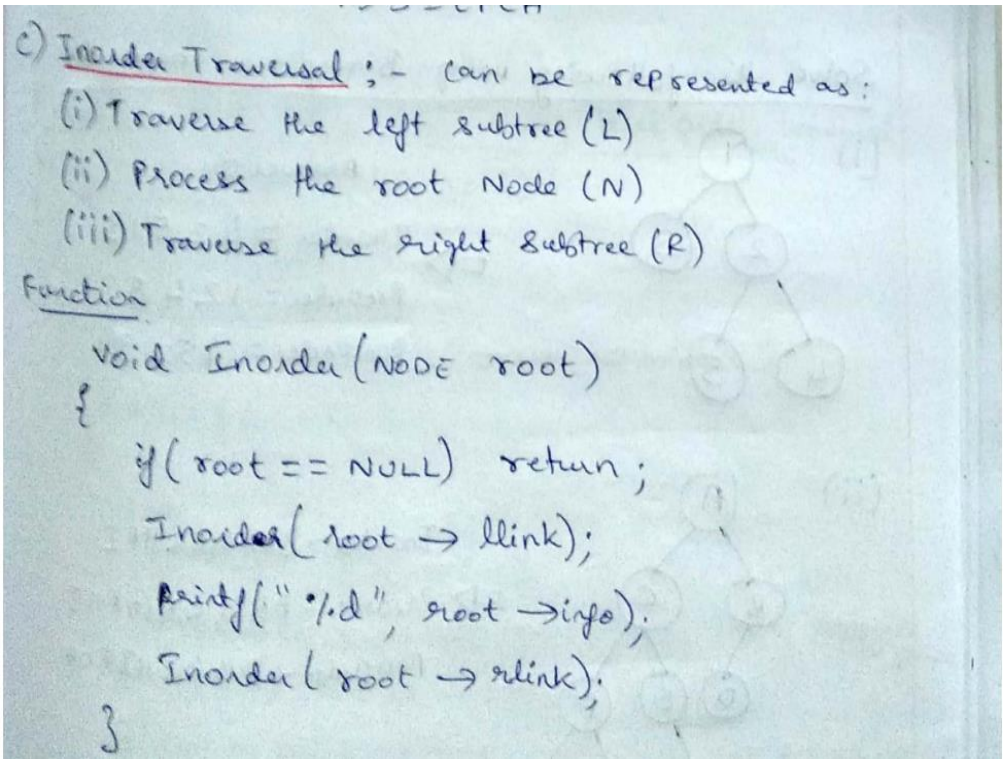
This binary tree follows the binary tree properties:

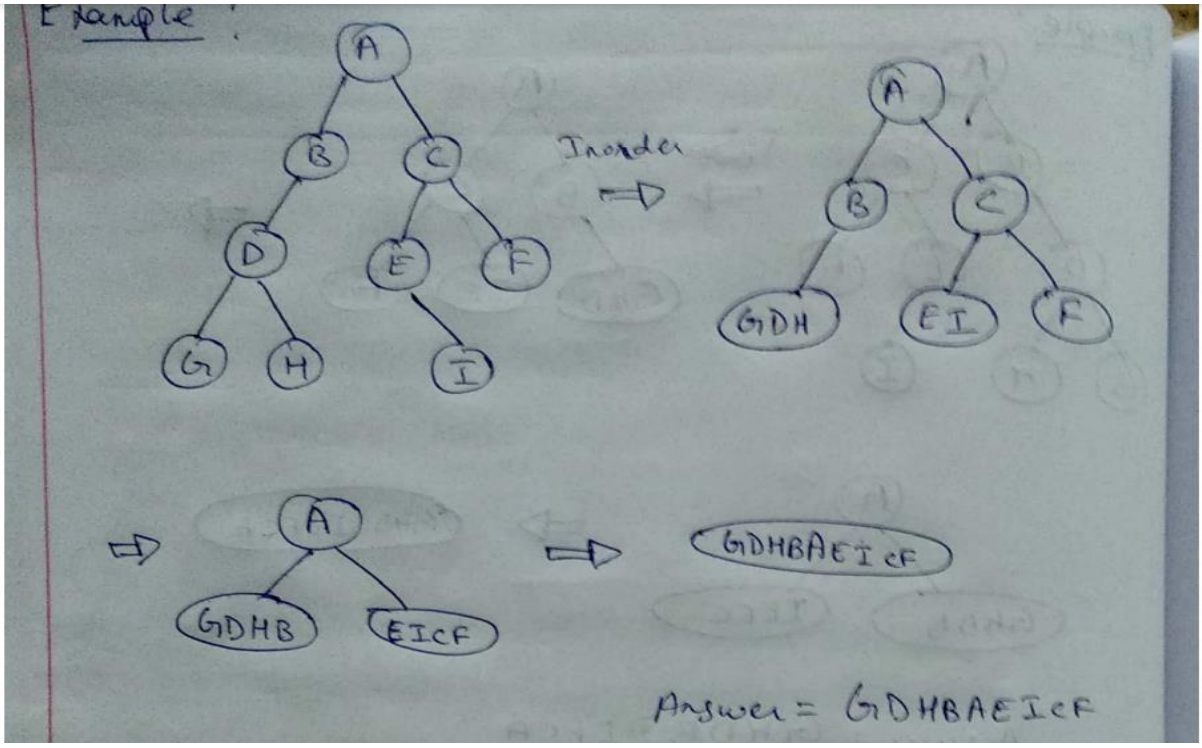
1. Each node has at most two children.
2. Each node has a value.
3. The left child of a node contains values less than or equal to the node's value.
4. The right child of a node contains values greater than the node's value.

In the example above, the left child of the root (2) contains values (4) less than the root's value (1), and the right child of the root (3) contains a value (5) greater than the root's value. This structure allows for efficient searching, insertion, and deletion operations in a binary tree.

C recursive routine to traverse the given tree

i. Inorder

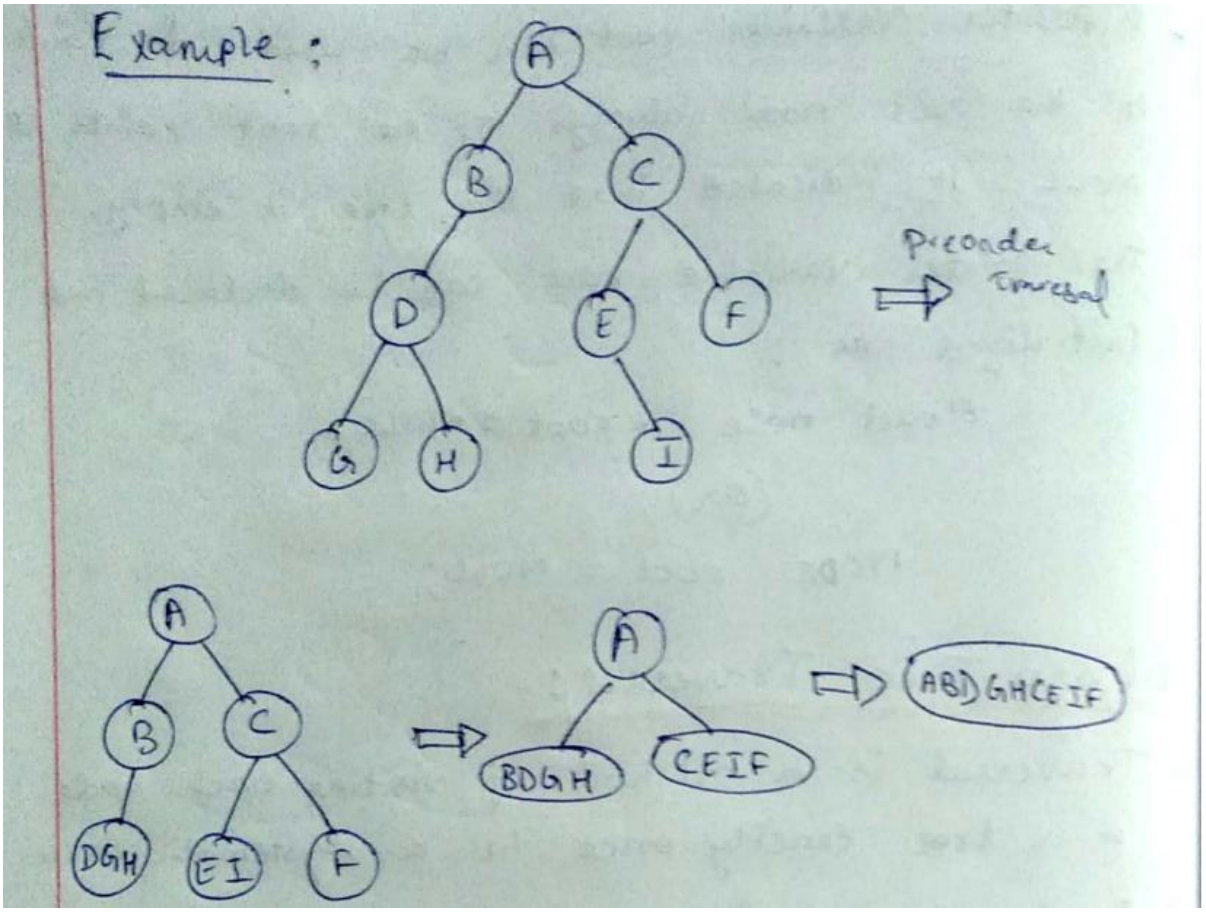




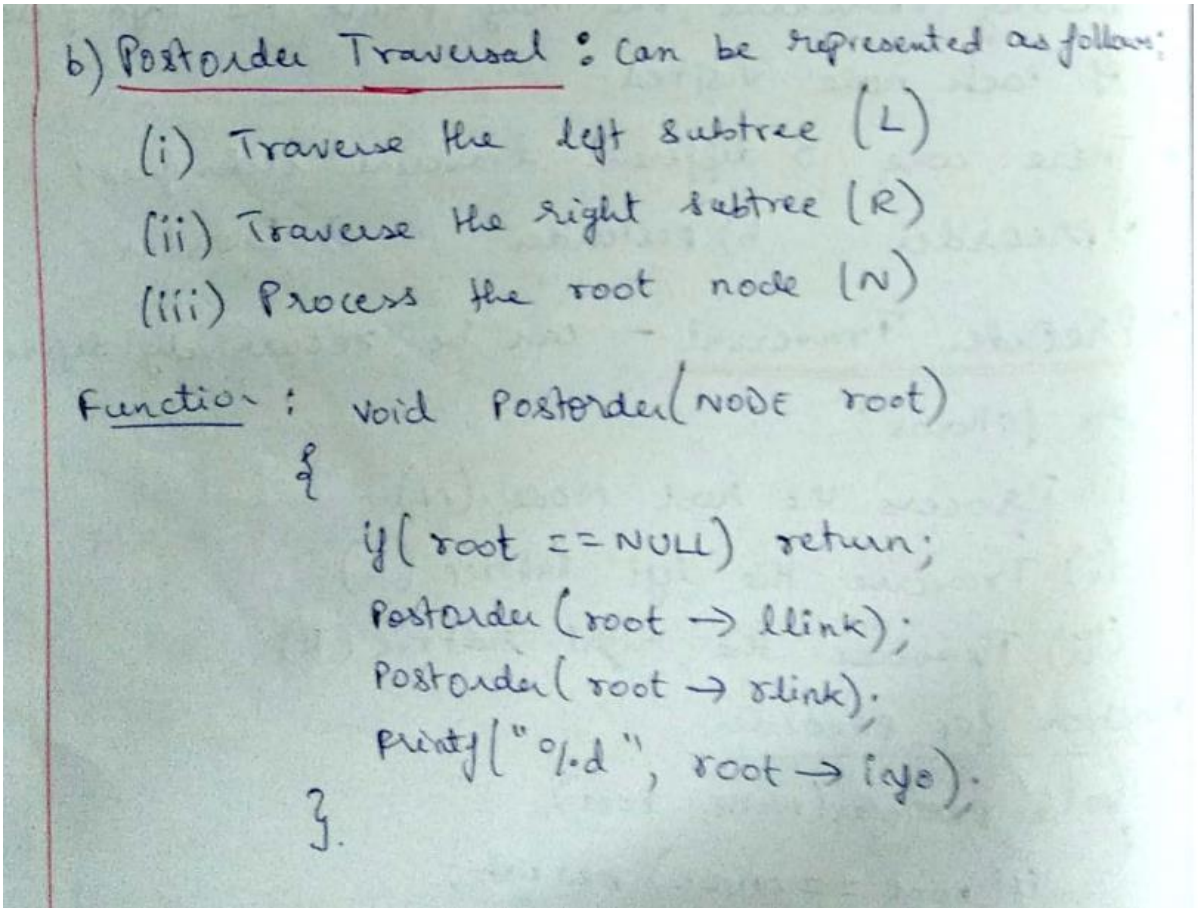
ii. Preorder

Function for preorder

```
Void preorder(NODE root)
{
    if(root == NULL) return;
    Printf("%d", root -> info); // visit the node
    Preorder (root -> llink); // visit left S.T recursively
    Preorder (root -> rlink); // visit right S.T recursively
}
```



iii. post order



b) Postorder Traversal : Can be represented as follows:

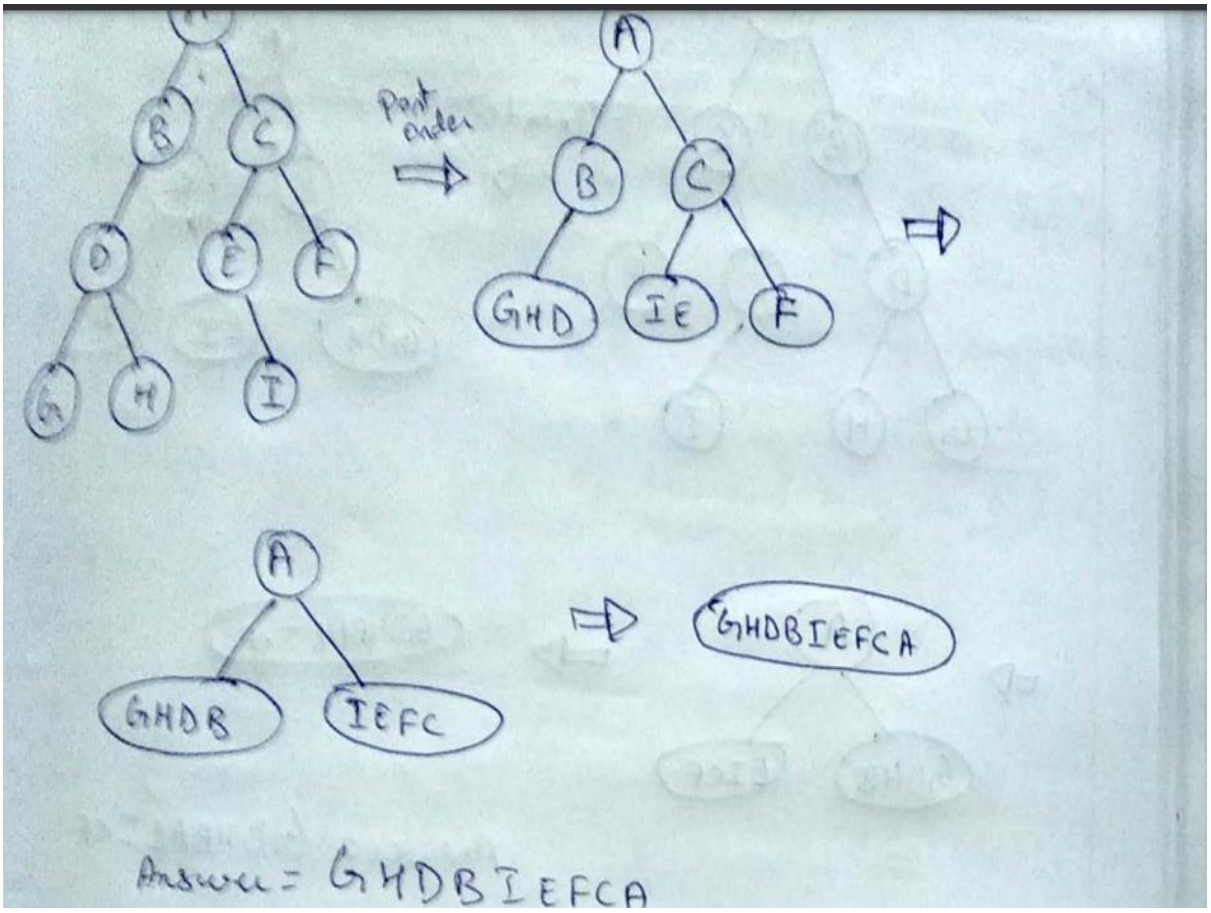
(i) Traverse the left subtree (L)

(ii) Traverse the right subtree (R)

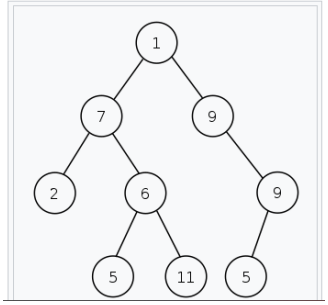
(iii) Process the root node (N)

Function :

```
void Postorder(NODE root)
{
    if(root == NULL) return;
    Postorder(root -> llink);
    Postorder(root -> rlink);
    Printf("%d", root -> info);
}
```


A
B
C
D
E
F
G
H
I
Post order
A
B
C
GHD
IE
F
A
GHDB
IEFC
GHDBIEFCA
Answer = GHDBIEFCA

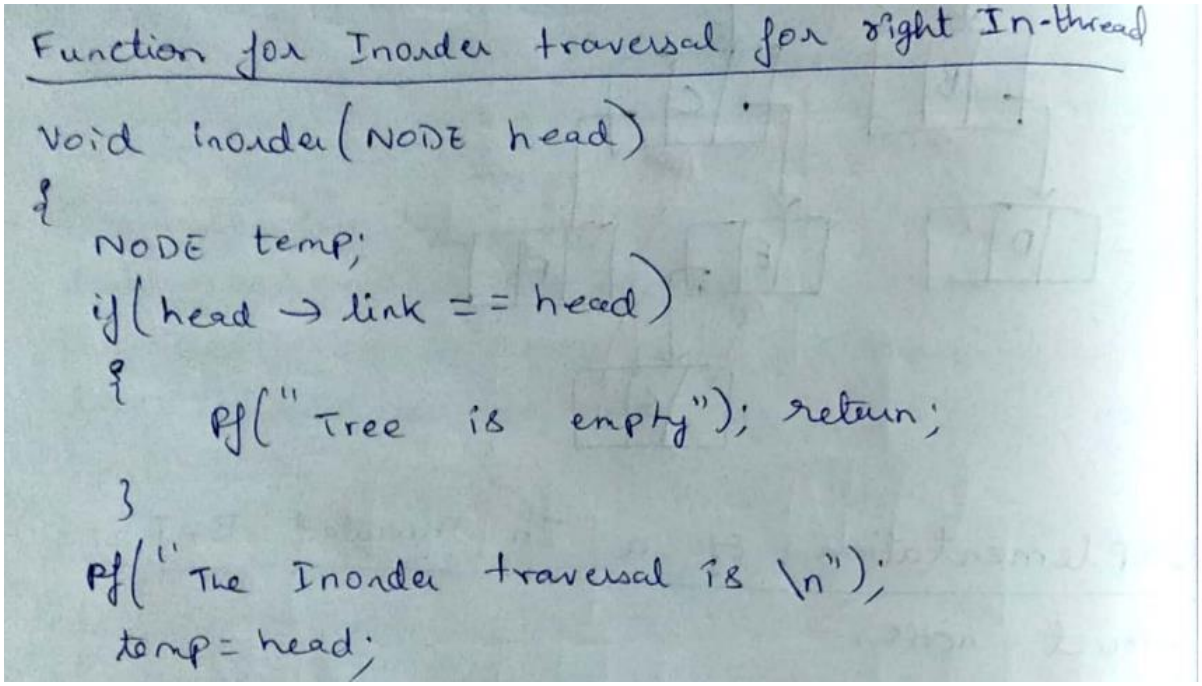
7. What is the advantage of the threaded binary tree over binary tree? Explain construction of threaded binary tree for following, explain c function for inorder traversal of threaded Binary tree



Advantages and disadvantages of Threaded B.T's:

Advantages: ① Traversing is faster, and easy to implement.
② Computations of predecessor and successor of given nodes is very easy and efficient.
③ Wastage of memory is avoided by storing NULL values with address of some other nodes.
④ Can move in upward and also downward directions.

Disadvantages: ① Here extra fields are required to check whether a link is a thread or not. Hence occupy more memory.
② Insertion & deletion of a node consumes more time than other type of trees.



```
Function for Inorder traversal for right In-thread

void inorder(NODE head)
{
  NODE temp;
  if(head -> link == head)
  {
     pf("Tree is empty"); return;
  }
  pf("The Inorder traversal is \n");
  temp = head;
```

8. List out the difference between SLL and DLL.

Singly Linked List (SLL) and Doubly Linked List (DLL) are two types of linked list data structures, and they differ in terms of their node structure, traversal capabilities, and memory usage. Here are some key differences between SLL and DLL:

1. Node Structure:
   - SLL: Each node in a singly linked list contains data and a reference (link or pointer) to the next node in the sequence.
   - DLL: Each node in a doubly linked list contains data and two references: one to the next node and another to the previous node.

2. Traversal:
   - SLL: In a singly linked list, traversal is only possible in one direction, starting from the head and moving towards the tail.
   - DLL: A doubly linked list allows traversal in both directions. You can traverse forward starting from the head or tail, and you can also traverse backward.

3. Memory Usage:
   - SLL: Singly linked lists generally consume less memory compared to doubly linked lists because they only store a reference to the next node.
   - DLL: Doubly linked lists use more memory since each node has references to both the next and the previous nodes.

4. Insertion and Deletion:

- SLL: Insertion and deletion operations in a singly linked list are relatively straightforward, but deleting a node requires traversing the list to update the links.
- DLL: Doubly linked lists provide easier insertion and deletion at any position in the list because each node has references to both its previous and next nodes.

5. Complexity:
- SLL: Singly linked lists are simpler to implement and use less memory, but they are less flexible when it comes to certain operations.
- DLL:Doubly linked lists are more complex but offer more flexibility, especially for operations involving both forward and backward traversal.

6. Extra Storage:
- SLL: Requires less memory for storage because it has only one link (next).
- DLL: Requires more memory for storage due to the presence of two links (next and previous).

7. Implementation Complexity:
- SLL: Implementation of singly linked lists is generally simpler compared to doubly linked lists.
- DLL: Implementing doubly linked lists involves managing both forward and backward links, adding some complexity to the implementation.

Choose between a singly linked list and a doubly linked list based on the specific requirements of your application and the operations you need to perform efficiently.

Write a C function for following operations on doubly linked list

i) Insert node at rear end ii) Delete a node at rear end

9. Discuss how to implement stacks using linked list.

10. Write a node structure for linked representation of a polynomial. Explain the function to add two polynomials represented using linked list.
11. For the given sparse matrix, give the linked list representation with necessary C declarations.

$$A = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 \\ 6 & 5 & 0 & 0 & 0 \\ 0 & 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

12. What is the advantage of doubly linked list over singly linked list? Illustrate with an example.

13. Given the following traversal, Draw a binary tree:

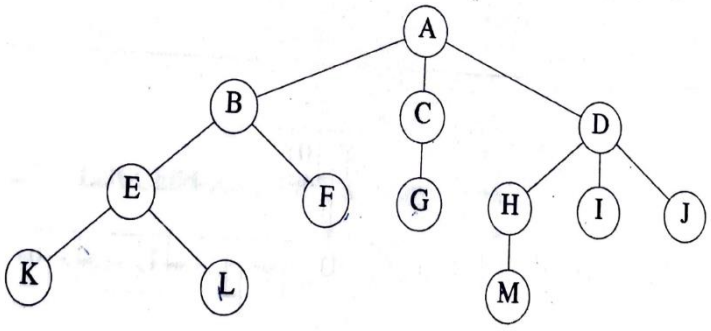
i) Inorder sequence - 8,4,10,9,11,2,5,1,6,3,7 and
Preorder sequence - 1,2,4,8,9,10,11,5,3,6,7.

ii) Post order : 9, 1, 2, 12, 7, 5, 3, 11, 4, 8 and
Inorder: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

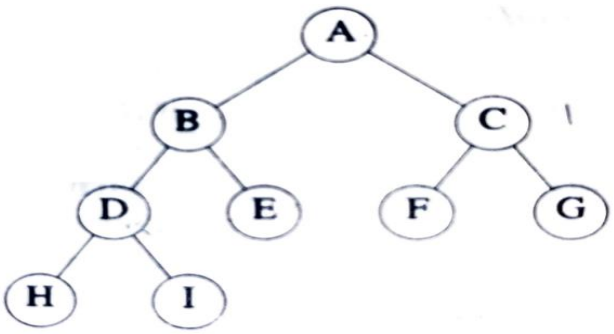
14. Represent the below given tree, using

i) Linked list representation

ii) Left child right sibling representation.



15. What is the advantages of the threaded Binary tee over binary tree? Explain the construction of threaded binary tree (In order Traversal) for below tree.



16. Define binary tree. List and discuss any two properties of binary tree.