PRACTICAL NO 1

AIM: Write a program to implement the concept of Divide and conquer strategy.

THEORY:

Divide-and-conquer:

Both merge sort and quicksort employ a common algorithmic paradigm based on recursion. This paradigm, divide-and-conquer, breaks a problem into subproblems that are similar to the original problem, recursively solves the subproblems, and finally combines the solutions to the subproblems to solve the original problem. Because divide-and-conquer solves subproblems recursively, each subproblem must be smaller than the original problem, and there must be a base case for subproblems. You should think of a divide-and-conquer algorithm as having three parts:

1. Divide the problem into a number of subproblems that are smaller instances of the same problem.
2. Conquer the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. Combine the solutions to the subproblems into the solution for the original problem.

1.

PROGRAM:

```
#include <stdio.h>
# define size 100
int min1, max1;
void div_con(int *, int, int *, int *);
void main()
{
int i, list[size], number;
int *min, *max;
printf("\n Input the number of elements in the list :");
scanf("%d",&number);
printf("\n Input the list elements :\n");
for(i=0;i<number; i++)
```

```
scanf("%d",&list[i]);
printf("\n Min and Max are :");
div_con(list, number, min, max);
Created by- Ms. P. P. Wakodikarprintf("\n Minimum = %d", *min);
printf("\n Maximum = %d", *max);
}
void div_con(int list[], int n, int *min, int *max)
{
if(n==1)
*min = *max = list[0];
else
{
if(n==2)
{
if(list[0] < list[1])
{
*min= list[0];
*max=list[1];
}
else
{
*min= list[1];
*max=list[0];
}
}
else
{
div_con(list, n/2, min, max);
div_con(list+n/2, n-n/2, &min1, &max1);
if(min1<*min)
*min=min1;
if(max1 > *max)
*max = max1;
}
```

}

}

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. How the divide nad the conqure techniques works?
2. difference between static and dyanamic memory allocation?
3. What are the different operations perform on data structure?

PRACTICAL NO 2

AIM: Write a program to implement Stack using an array.

THEORY:

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.

PROGRAM:

```
#include<stdio.h>
# define SIZE 10
int top = -1;
int flag = 0;
void push(int *,int);
int pop(int *);
void display(int *);
Created by- Ms. P. P. Wakodikarvoid main()
{
int data;
int stack[SIZE];
char ch;
clrscr();
while(1)
{
printf("\n\n\t 1. Push \n\t 2. Pop \n\t 0. Quit");
printf("\n\n\t Enter your choice :");
fflush(stdin);
ch = getche();
switch(ch)
{
case '1' :
printf("\n\n Input the element to push :");
```

```
scanf("%d",&data);
push(stack,data);
if(flag)
{
printf("\n\n After inserting :");
display(stack);
Created by- Ms. P. P. Wakodikarif( top == (SIZE-1))
printf("\n\n Stack is Full");
}
else
printf("\n\n Stack overflow after pushing");
break;
case '2' :
data = pop(stack);
if(flag)
{
printf("\n\n Data popped is : %d",data);
printf("\n\n Rest data in stack is as follows :");
display(stack);
}
else
printf("\n\n Stack is underflow");
break;
case '0' :
exit(0);
break;
Created by- Ms. P. P. Wakodikardefault :
printf("\a\a\a");
break;
}
}
}
void push(int stack[],int d)
{
```

```
if(top == (SIZE-1))
flag = 0;
else
{
flag = 1;
++top;
stack[top] = d;
}
}
int pop(int stack[])
{
int poped_ele;
if(top == -1)
{
Created by- Ms. P. P. Wakodikarpoped_ele = 0;
flag = 0;
}
else
{
flag = 1;
poped_ele = stack[top];
--top;
}
return(poped_ele);
}
void display(int stack[])
{
int i;
if(top == -1)
{
printf("\n\n Stack is Empty");
}
else
{
```

```
for(i=top;i>=0;--i)
printf("\n%5d",stack[i]);
}
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What is stack?
2. What are the basic operations performed on stack?
3. How do you represent stack in C?
4. What is the significance of the top in a stack?
5. When do you say stack is full or empty?

PRACTICAL NO 3

AIM: Write a program to implement Queue using an array.

THEORY:

Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called REAR(also called tail), and the deletion of exisiting element takes place from the other end called as FRONT(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

The process to add an element into queue is called Enqueue and the process of removal of an element from queue is called Dequeue.

PROGRAM:

```
#include<stdio.h>
Created by- Ms. P. P. Wakodikar#include<conio.h>
# define SIZE 10
int front = 0;
int rear = 0;
void insert_q(int *,int);
void del_q(int *);
void display(int *);
void main()
{
int data;
int queue[SIZE];
char ch;
clrscr();
while(1)
{
printf("\n\n\t 1. Insert \n\t 2. Delete \n\t 0. Quit");
printf("\n\n\t Enter your choice :");
fflush(stdin);
ch = getche();
switch(ch)
{
case '1' :
```

```
Created by- Ms. P. P. Wakodikarprintf("\n\n Input the element to insert :");
scanf("%d",&data);
insert_q(queue,data);
printf("\n\n Queue after inserting :");
display(queue);
break;
case '2' :
del_q(queue);
printf("\n\n Rest data in queue is as follows :");
display(queue);
break;
case '0' :
exit(0);
break;
default :
printf("\a\a\a");
break;
}
}
Created by- Ms. P. P. Wakodikar}
void insert_q(int queue[],int d)
{
if(rear < SIZE-1)
{
++rear;
queue[rear] = d;
if(front == 0)
front = 1;
}
else
printf("\n\n Queue is overflow ");
}
void del_q(int queue[])
{
```

```
int data;
if(front == 0)
{
printf("\n\n Queue is Underflow");
return;
}
Created by- Ms. P. P. Wakodikarelse
{
data = queue[front];
printf("\n\n Element Deleted is %d",data);
}
if(front == rear)
{
front = 0;
rear = 0;
}
else
front ++;
}
void display(int queue[])
{
int i;
if(front == 0)
{
printf("\n\n Queue is Empty");
}
else
{
for(i=front;i<=rear;i++)
Created by- Ms. P. P. Wakodikarprintf("\n%5d",queue[i]);
}
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What is queue?
2. How is the queue different from the stack?
3. What are the operations performed on Queue?
4. How do you represent the queue in computer memory?
5. What are the disadvantages of a queue?

PRACTICAL NO 4 .

AIM: Write a program to implement Insertion and Deletion on Singly Link List.

THEORY:

There are three situation for inserting element in list.

1. Insertion at the front of list.
2. Insertion in the middle of the list.
3. Insertion at the end of the list.

1. Procedure For Inserting an element to linked list

Step-1: Get the value for NEW node to be added to the list and its position.

Step-2: Create a NEW, empty node by calling malloc(). If malloc() returns no error then go to step-3 or else say "Memory shortage".

Step-3: insert the data value inside the NEW node's data field.

Step-4: Add this NEW node at the desired position (pointed by the "location") in the LIST. Created by- Ms. P. P. WakodikarStep-5: Go to step-1 till you have more values to be added to the LIST. An element that is to be deleted from a linked list is to be searched first. For this, the traversing operation must be carried out thoroughly on th list. After finding the element there may be two cases for deletion:

1.Deletion at beginning

2.Deletion in between

Deletion at beginning:

We know that start pointer points to the first element of a linked list. If element to be deleted is the first element of linked list then we assign the value of start to temp as: temp=start ; So now temp points to first node which has to be deleted. Now we assign the next part of the deleted node to start as: start=start->next;

Since start points to the first element of the linked list, so start->next will point to the second element of the linked list. Now we should free the element to be deleted which is pointed by temp with the following statement: free(temp); Deletion in between:

If the element is other than the first element of the linked list then we assigh the next part of the deleted node to the next part of the previous node. This can be written as:

```
temp=q->next;
q->next=temp->next;
free(temp);
```

Here, the pointer q is pointing to the previous node of the node to be deleted. After the first

statement temp will point to the node to be deleted, after second statement next ofprevious node will point to next node of the node to be deleted.If the node to be deleted is the last node of the linked list then the second statement will be as:

q->next=NULL;

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
Created by- Ms. P. P. Wakodikar}*first=NULL;
void insert()
{
struct node *temp;
struct node *nn=(struct node*)malloc(sizeof(struct node));
printf("enter the data\n");
scanf("%d",&nn->data);
emp=first;
while(temp->next!=first)
temp=temp->next;
temp->next=nn;
nn->next=NULL;
}
void display()
{
struct node *temp;
temp=first;
if(temp==NULL)
{
printf("no elements\n");
return;
}
printf("elements in linked list are\n");
```

```
while(temp!=NULL)
{
printf("%d\n",temp->data);
temp=temp->next;
}
}
void deletion()
{
struct node *temp;
temp=first;
first=first->next;
temp->next=NULL;
free(temp);
}
int main()
{
int op;
do
{
printf("1.insertion\n2.deletion\n3.display\n4.exi\n");
printf("enter option\n");
scanf("%d",&op);
switch(op)
Created by- Ms. P. P. Wakodikar{
case 1:insert();
break;
case 2:deletion();
break;
case 3:display();
break;
}
}while(op!=6);
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What are the types of link list?
2. What are the advantages of link list?
3. What are the application of the link list?
4. What is node?
5. What do you mean by free pull?

PRACTICAL NO 5

AIM: Write a program to implement Circular Queue.

THEORY:

In a standard queue data structure re-buffering problem occurs for each dequeue operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue Circular queue is a linear data structure. It follows FIFO principle.

- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list fallow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue Both the front and the rear pointers points to the beginning of the array.
- It is also called as "Ring buffer".
- Items can inserted and deleted from a queue in O(1) time.

PROGRAM:

```
#include<stdio.h>
# define SIZE 5
int front = -1;
int rear = -1;
int data;
int queue[SIZE];
void insert_q();
void del_q();
void display_q();
void main()
{
char ch;
clrscr();
while(1)
{
printf("\n\n\t 1. Insert \n\t 2. Delete \n\t 0. Quit");
printf("\n\n\t Enter your choice :");
Created by- Ms. P. P. Wakodikarfflush(stdin);
ch = getche();
switch(ch)
```

```
{
case '1' :
insert_q();
printf("\n\n Queue after inserting :");
display_q();
break;
case '2' :
del_q();
printf("\n\n Rest data in queue is as follows :");
display_q();
break;
case '0' :
exit(0);
break;
default :
Created by- Ms. P. P. Wakodikarprintf("\a\a\a");
break;
}
}
}
void insert_q()
{
if ((front == 0) && (rear == SIZE-1))
{
printf("\n\n Queue is overflow ");
rear = -1;
return;
}
else
{
if(front < 0)
/* Insert First Element */
{
front = 0;
```

```
rear = 0;
printf("\n\n Input the element to insert :");
scanf("%d",&data);
queue[rear] = data;
}
else
Created by- Ms. P. P. Wakodikar{
if(rear == SIZE-1)
{
rear = 0;
printf("\n\n Input the element to insert :");
scanf("%d",&data);
queue[rear] = data;
}
else
{
rear++;
printf("\n\n Input the element to insert :");
scanf("%d",&data);
queue[rear] = data;
}
}
}
}
void del_q()
{
if(front < 0)
{
printf("\n\n Queue is Underflow");
Created by- Ms. P. P. Wakodikarreturn;
}
else
{
data = queue[front];
```

```
printf("\n\n Element Deleted is %d",data);
}
if(front == rear)
{
front = -1;
rear = -1;
}
else
{
if(front == SIZE-1)
{
front = 0;
}
else
{
front++;
}
}
}
Created by- Ms. P. P. Wakodikarvoid display_q()
{
int i;
if(front < 0)
{
printf("\n\n Queue is Empty");
}
else
{
if(rear >= front)
{
for(i=front;i<=rear;i++)
{
printf("\ni = %d",i);
printf("%5d",queue[i]);
```

```
}
}
else
{
for(i=front;i<SIZE;i++)
{
printf("\ni = %d",i);
printf("%5d",queue[i]);
}
Created by- Ms. P. P. Wakodikarfor(i=0;i<=rear;i++)
{
printf("\ni = %d",i);
printf("%5d",queue[i]);
}
}
}
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What are the advantages of circular queue as compared to simple queue?
2. What is circular queue?
3. How do you represent it?

PRACTICAL NO 6

AIM: Write a program to implement Infix to PostfixExpression Evaluation.

THEORY:

In order to convert infix to postfix expression, we need to understand the precedence of operators first. Precedence of Operators There are five binary operators, called addition, subtraction, multiplication, division and exponentiation. We are aware of some other binary operators. For example, all relational operators are binary ones. There are some unary operators as well. These require only one operand e.g. – and +. There are rules or order of execution of operators in Mathematics called precedence. Firstly, the exponentiation operation is executed, followed by multiplication/division and at the end addition/subtraction is done. The order of precedence is

(highest to lowest):Exponentiation -Multiplication/division *, /Addition/subtraction +,

-For operators of same precedence,the left-to-right rule applies: A+B+C means (A+B)+C.

For exponentiation, the right-to-left rule applies:

A B C means A (B C)

We want to understand these precedence of operators and infix and postfix forms of expressions. A programmer can solve a problem where the program will be aware of the precedence rules and convert the expression from infix to postfix based on the precedence rules.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define SIZE 64
int top = -1;
void push(char *,char);
char pop(char *);
void infix_post(char *,char *);
int priority(char);
Created by- Ms. P. P. Wakodikarvoid main()
{
char infix[SIZE],post[SIZE];
clrscr();
```

```
puts("Enter the infix expression :");
gets(infix);
infix_post(infix,post);
puts("The post fix expression is :");
puts(post);
getch();
}
void infix_post(char *i,char *p)
{
char t;
char stack[SIZE];
while(*i)
{
if(*i == ' ' || *i == '\t') /* ignore white spaces & tabs in infix expression */
{
i++;
continue;
}
if( isdigit(*i) || isalpha(*i))
Created by- Ms. P. P. Wakodikar{
*p = *i;
p++;
i++;
}
if( *i == '(')
/* push left parenthesis onto stack */
{
push(stack,*i);
i++;
}
if( *i == ')')
/* pop stack until matching left parenthesis */
{
t = pop(stack);
```

```
while( t != '(')
/* Write poped element into postfix array */
{
*p = t;
p++;
t = pop(stack);
}
i++;
}
if ( *i == '+' || *i == '-' || *i == '*' || *i == '/' || *i == '%')
{
if( top == -1)
push(stack,*i);
/* pop stack until first operator of higher priority & then push this
operator */
{
t = pop(stack);
while( priority(t) >= priority(*i) )
{
*p = t;
p++;
t = pop(stack);
}
push(stack,t);
push(stack,*i);
}
i++;
}
}
while( top != -1)
{
t = pop(stack);
*p = t;
p++;
```

```
}
*p = '\0';
}
Created by- Ms. P. P. Wakodikar
/* pop any remaining operators */void push(char *s,char t)
{
if( top == SIZE)
puts(" Stack is full");
else
{
++top;
s[top] = t;
}
}
char pop(char *s)
{
char t='\0';
if( top == -1)
{
puts("Stack is Empty");
return;
}
else
{
t = s[top];
top--;
}
return(t);
}
Created by- Ms. P. P. Wakodikarint priority(char t)
{
int pri;
if( t == '*' || t == '/' || t== '%')
pri = 2;
```

```
else
{
if( t == '+' || t == '-')
pri = 1;
else
pri = 0;
}
return(pri);
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What do you mean by notation?
2. What is reverse polish notation?
3. Which operator is having highest priority?

```
voi
main()
{
```

PRACTICAL NO 7

AIM: Write a program to implement the concept of binary tree.

THEORY:

We extend the concept of linked data structures to structure containing nodes with more than one self-referenced field. A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root. Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings. More tree terminology:

The depth of a node is the number of edges from the root to the node.

The height of a node is the number of edges from the node to the deepest leaf.

The height of a tree is a height of the root. A full binary tree.is a binary tree in which each node has exactly zero or two children. A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right. A complete binary tree is very special tree, it provides the best possible ratio between the number of rnodes and the height. The height h of a complete binary tree with N nodes is at most O(log N). We can easily prove this by counting nodes on each level, starting with the root, assuming that eachlevel has the maximum number of nodes:

PROGRAM:

```
#include<stdio.h>
#include<malloc.h>
struct tree
{
int val;
struct tree *left,*right;
};
#define SIZE 10
struct tree *create_bt(int *,int,int);
void display(struct tree *, int);
main()
{
```

```
struct tree *T = NULL;
int arr[SIZE],i = 0;
clrscr();
printf("\n\t Program to Create Binary Tree \n");
Created by- Ms. P. P. Wakodikarwhile(i<SIZE)
{
printf("\n Enter any Value : (-999 to Exit) : ");
scanf("%d",&arr[i]);
if(arr[i] == -999)
break;
i++;
}
T = create_bt(arr,0,--i);
printf("\n\t ****** TREE ******\n");
display(T,1);
}
struct tree *create_bt(int *arr,int lb,int ub)
{
struct tree *S;
int mid = (lb+ub)/2;
S = (struct tree *)malloc(sizeof(struct tree));
S->val = arr[mid];
if(lb > ub)
{
S->left = NULL;
Created by- Ms. P. P. WakodikarS->right = NULL;
return(S);
}
if(lb <= mid-1)
S->left = create_bt(arr,lb,mid-1);
else
S->left = NULL;
if(mid+1 <= ub)
S->right = create_bt(arr,mid+1,ub);
```

```
lse
S->right = NULL;
return(S);
}
void display(struct tree *S,int level)
{
int i;
if(S)
{
display(S->right,level+1);
printf("\n");
Created by- Ms. P. P. Wakodikarfor(i=0;i<level;i++)
printf(" ");
printf("%d",S->val);
display(S->left,level+1);
}
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

1. What do you mean by tree?
2. What is the binary tree?
3. What is the importance of binary tree?
4. What do you mean by strictly binary tree?
5. What are the properties of binary tree?

PRACTICAL NO 8

AIM: Write a program to create and display threaded binary tree.

THEORY:

Inorder traversal of a Binary tree is either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists). There are two types of threaded binary trees. Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)Double Threaded: Where both left and right NULL pointers are made to point to inorderpredecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal. The threads are also useful for fast accessing ancestors of a node. Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads. C representation of a Threaded NodeFollowing is C representation of a single threaded node.

```
struct Node
{
int data;
Node *left, *right;
bool rightThread;
}
```

Since right pointer is used for two purposes, the boolean variable rightThread is used to indicate whether right pointer points to right child or inorder successor. Similarly, we can add leftThread for a double threaded binary tree.

PROGRAM:

```
#include<stdio.h>
#include<malloc.h>
struct tree
{
Created by- Ms. P. P. Wakodikarint val;
int tleft,tright;
struct tree *left,*right;
};
struct tree * create_t(struct tree *,int);
```

```
void in_order_trav(struct tree *);
struct tree *Q;
void main()
{
int data;
struct tree *T = NULL;
clrscr();
printf("\n Program for Tree Creation :\n");
while(1)
{
printf("\n\n\t Enter any value : ( -999 to Exit ) : ");
scanf("%d",&data);
if(data == -999)
break;
Created by- Ms. P. P. WakodikarT = create_t(T,data);
}
printf("\n The in-order Traversal of a Threaded Binary Tree :\n\n");
in_order_trav(T);
getch();
}
struct tree * create_t(struct tree *S,int data)
{
int flag = 0;
struct tree *temp;
if( S == NULL )
{
S = (struct tree *) malloc(sizeof(struct tree ));
S->val = data;
S->left = NULL;
S->right = NULL;
S->tleft = 1;
S->tright = 1;
Created by- Ms. P. P. WakodikarQ=S;
return(Q);
```

```
}
while(flag != 1)
/* Using flag to indicate inserted */
{
if(data < S->val && S->tleft == 0)
S = S->left;
else
{
if(data > S->val && S->tright == 0)
S = S->right;
else
{
if(data < S->val && S->tleft == 1)
flag = 1;
else
{
if(data > S->val && S->tright == 1)
flag = 1;
}
}
}
}
Created by- Ms. P. P. Wakodikarif( data < S->val && S->tleft == 1) /* Insertion on Left /
{
temp = (struct tree*)(malloc(sizeof(struct tree)));
temp ->val = data;
temp->left = S->left;
temp->right = S;
temp->tleft = 1;
temp->tright = 1;
S->left = temp;
S->tleft = 0;
}
else
```

```
{
if( data > S->val && S->tright == 1)
/* Insert on Right */
{
temp = (struct tree*)(malloc(sizeof(struct tree)));
temp ->val = data;
temp->left = S;
temp->right = S->right;
temp->tleft = 1;
temp->tright = 1;
Created by- Ms. P. P. WakodikarS->right = temp;
S->tright = 0;
}
}
return(Q);
}
void in_order_trav(struct tree *S)
{
int flag = 0;
while(S != NULL)
{
while(S->tleft == 0 && flag == 0)
S = S->left;
printf("%5d",S->val);
if(S->tright == 0)
{
S = S->right;
flag = 0;
}
else
{
Created by- Ms. P. P. WakodikarS = S->right;
flag = 1;
}
```

```
}
}
```

OUTPUT:

RESULT: we have executed the program successfully

FLOWCHART:

VIVA QUESTIONS:

1. What is threaded binary tree?
2. What are its advantages and disadvantages?
3. Why we are implement the concept of threaded binary tree?

PRACTICAL NO 9

AIM: Write a program to implement the concept of depth first search and breadth first search.

THEORY:

Traversal of graphs and digraphs To traverse means to visit the vertices in some systematic order. You should be familiar with various traversal methods for trees:

preorder: visit each node before its children.postorder: visit each node after its children.

(for binary trees only): visit left subtree, node, right subtree.We also saw another kind of traversal, topological ordering, when I talked about shortest paths. Today, we'll see two other traversals: breadth first search (BFS) and depth first search (DFS). Both of these construct spanning trees with certain properties useful in other graph algorithms. We'll start by describing them in undirected graphs, but they are both also very useful for directed graphs. Breadth First Search This can be throught of as being like Dijkstra's algorithm for shortest paths, but with every edge having the same length. However it is a lot simpler and doesn't need any data structures. We just keep a tree (the breadth first search tree), a list of nodes to be added to the tree, and markings (Boolean variables) on the vertices to tell whether they are in the tree or list.breadth first search:unmark all verticeschoose some starting vertex x

mark x

list L = x

tree T = x

while L nonempty

choose some vertex v from front of list

visit v

for each unmarked neighbor w

mark w

add it to end of list

add edge vw to T

Depth first search

Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It ismost easy to program as a recursive routine:

```
preorder(node v)
{
visit(v);
for each child w of v
preorder(w);
}
```

To turn this into a graph traversal algorithm, we basically replace "child" by "neighbor". But toprevent infinite loops, we only want to visit each vertex once. Just like in BFS we can use marksto keep track of the vertices that have already been visited, and not visit them again. Also, justlike in BFS, we can use this search to build a spanning tree with certain useful properties.

```
dfs(vertex v)
{
visit(v);
for each neighbor w of v
```



```
{
dfs(w);
add edge vw to tree T
}
}
```

PROGRAM:

```
#include<stdio.h>
#include<malloc.h>
struct tree
{
int val;
struct tree *left,*right;
};
#define SIZE 10
int depth_bt = 0;
struct tree *create_bt(int *,int,int);
void display(struct tree *, int);
int depth(struct tree *,int);
```

```
main()
{
struct tree *T = NULL;
int arr[SIZE],i = 0;
clrscr();
Created by- Ms. P. P. Wakodikarprintf("\n\t Program to Create Binary Tree \n");
while(i<SIZE)
{
printf("\n Enter any Value : (-999 to Exit) : ");
scanf("%d",&arr[i]);
if(arr[i] == -999)
break;
i++;
}
T = create_bt(arr,0,--i);
printf("\n\t ****** TREE ******\n");
display(T,1);
depth_bt = depth(T,0);
printf("\n Depth of Binary Tree is : %d",depth_bt);
getch();
}
struct tree *create_bt(int *arr,int lb,int ub)
{
struct tree *S;
int mid = (lb+ub)/2;
Created by- Ms. P. P. WakodikarS = (struct tree *)malloc(sizeof(struct tree));
S->val = arr[mid];
if(lb > ub)
{
S->left = NULL;
S->right = NULL;
return(S);
}
if(lb <= mid-1)
```

```
S->left = create_bt(arr,lb,mid-1);
else
S->left = NULL;
if(mid+1 <= ub)
S->right = create_bt(arr,mid+1,ub);
else
S->right = NULL;
return(S);
}
void display(struct tree *S,int level)
Created by- Ms. P. P. Wakodikar{
int i;
if(S)
{
display(S->right,level+1);
printf("\n");
for(i=0;i<level;i++)
printf(" ");
printf("%d",S->val);
display(S->left,level+1);
}
}
int depth(struct tree *S,int level)
{
if(S)
{
if(level > depth_bt)
depth_bt = level;
depth(S->left,level+1);
depth(S->right,level+1);
Created by- Ms. P. P. Wakodikar}
return(depth_bt);
}
```

OUTPUT:

RESULT: we have executed the program successfully

FLOWCHART:

VIVA QUESTIONS:

1. What is graph?
2. Differntiate between DFS and BFS?
3. What are the different shortest path algorithm?
4. At the time of implementation of DFS which data structure is used?

PRACTICAL NO 10

AIM: Write a program to implement Quick Sort.

THEORY:

Quick Sort Algorithm

Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less aditional space. It is based on the rule of Divide and Conquer(also called partition-exchange sort). This algorithm divides the list into three main parts :

1. Elements less than the Pivot element
2. Pivot element
3. Elements greater than the pivot element

In the list of elements, mentioned in below example, we have taken 25 as pivot. So after the first pass, the list will be changed like this.6 8 17 14 25 63 37 52 Hnece after the first pass, pivot will be set at its position, with all the elements smaller to it on its left and all the elements larger than it on the right. Now 6 8 17 14 and 63 37 52 areconsidered as twoseparate lists, and same logic is applied on them, and we keep doing this until the complete list is sorted.

PROGRAM:

```
void main()
{
int arr[]={5,8,17,9,10,21,14,1};
clrscr();
printf("\n The array before sorting :\n");
Created by- Ms. P. P. Wakodikardisplay(arr,8);
printf("\n The array after Bubble Sort :\n\n");
quick_sort(arr,0,7);
/*display(arr,8);*/
getch();
}
display(int *arr,int size)
{
int i;
for(i=0;i<size;i++)
```

```
printf("%5d",arr[i]);
}
quick_sort(int *arr,int first,int last)
{
int low,high,mid,temp,pivot;
low = first;
high = last;
mid = (first+last)/2;
pivot = arr[mid];
printf("\n\n First= %d\t Last= %d",first,last);
while(low<=high)
{
Created by- Ms. P. P. Wakodikarwhile(arr[low] < pivot)
low++;
while(arr[high] > pivot)
high--;
if(low <= high)
{
temp = arr[low];
arr[low] = arr[high];
arr[high] = temp;
low++;
high--;
}
}
printf("\n\n Low = %d\t High = %d",low,high);
printf("\n\n");
display(arr,8);
if(first <= high)
quick_sort(arr,first,high);
if(low <= last)
quick_sort(arr,low,last);
}
```

OUTPUT:

FLOWCHART:

RESULT: we have executed the program successfully

VIVA QUESTIONS:

What do you mean by interna;l and external sorting?

2. What is sorting?

3. What are the different types of sorting techniques?

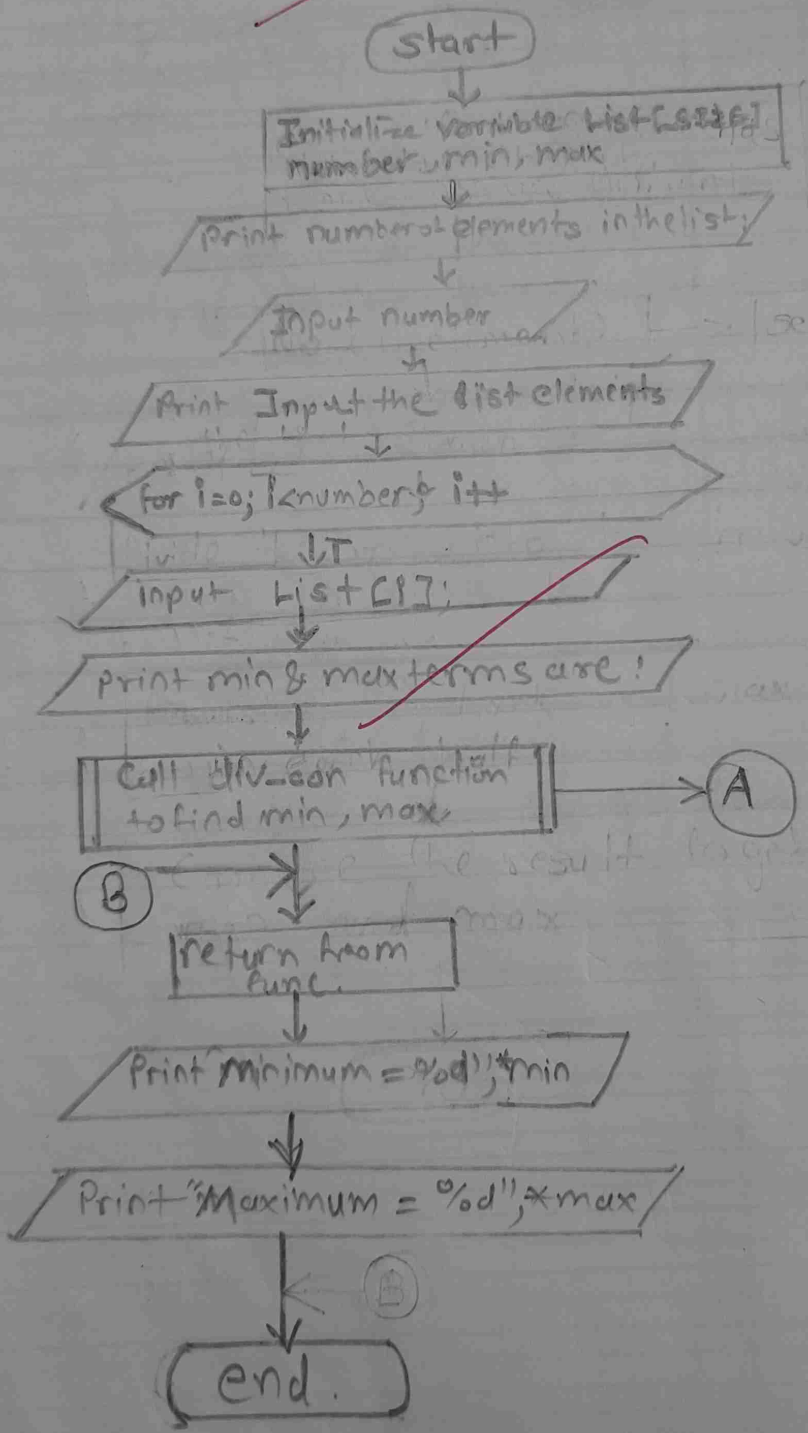
4. What is a quick sort? How it is fiffere from the bubble sort?

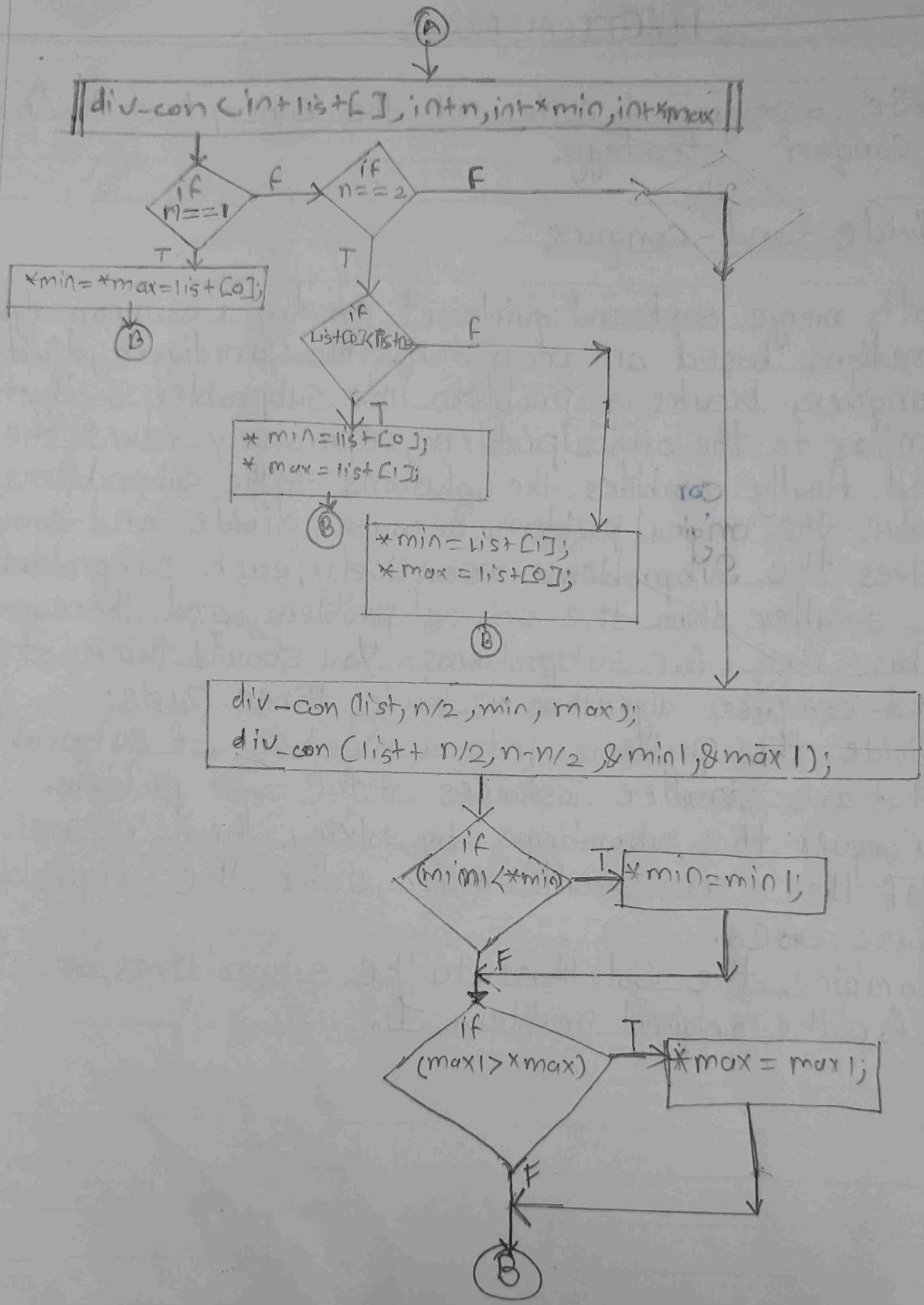
5. What are the factors to be concider during the selection of a sorting techniques?

Practical No-1

Aim :- Write a program to implement concept of Divide and conquer strategy.

Flow chart :-




A
div-con (int list[], int n, int *min, int *max)
if n==1
F
T
*min = *max = list[0];
B
if n==2
F
T
if list[0] < list[1]
F
T
*min = list[0];
*max = list[1];
B
*min = list[1];
*max = list[0];
B
div-con (list, n/2, min, max);
div-con (list + n/2, n-n/2, &min1, &max1);
if (min1 < *min)
T
*min = min1;
F
if (max1 > *max)
T
*max = max1;
F
B

Output:- Input the number of elements in List : 5
Input the list elements:

3
1
4
1
5

Min and Max are :

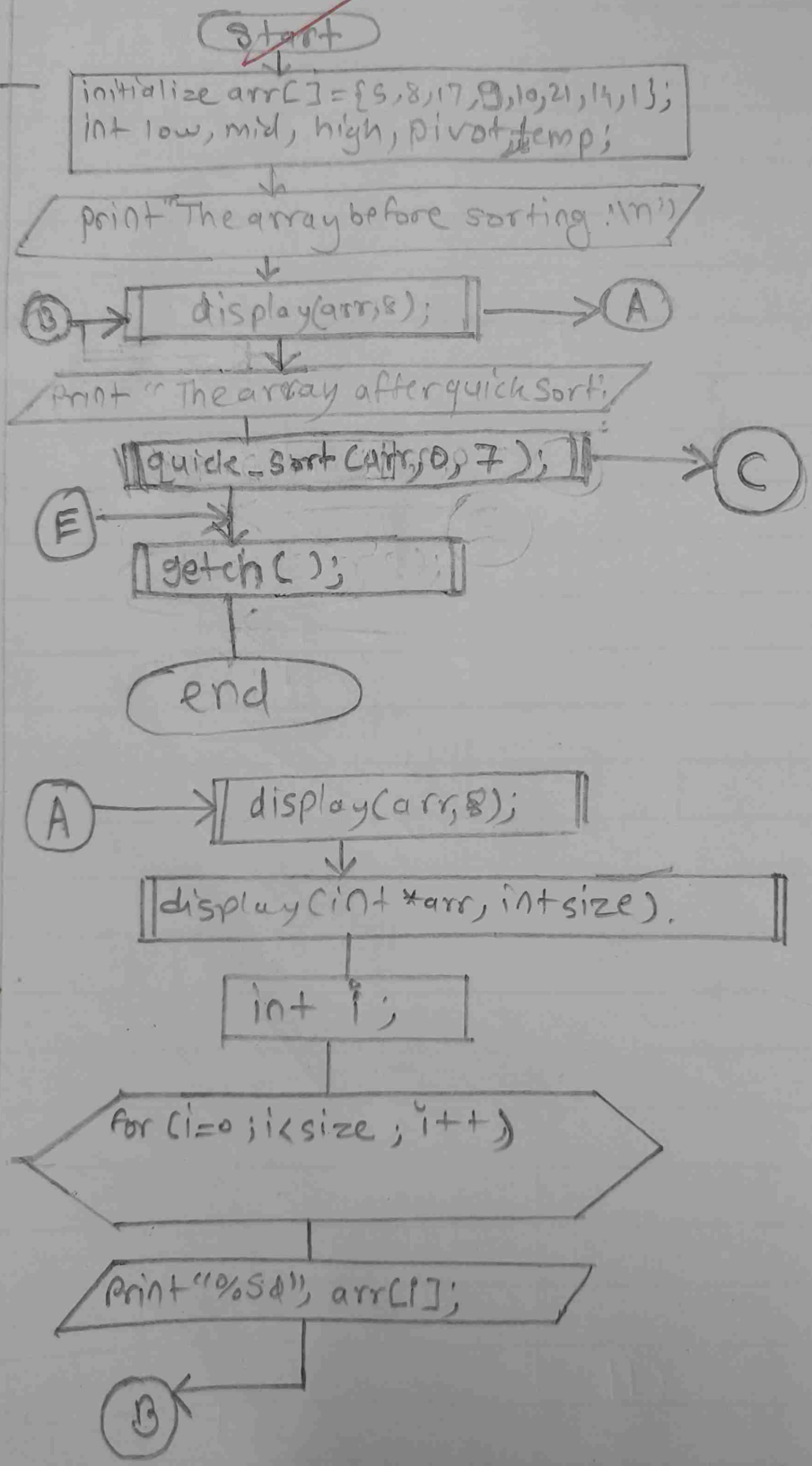
Minimum : 1

Maximum : 5

Practical No- 2

Aim :- Write a program to implement Quick Sort.

Flow-Chart :-



Output:- The array before sorting :

5
8
17
9
10
21
14
1

First = 5 Last = 3

Low = 5 High = 3

5 8 17 9 10 21 14 17

Low = 2 High = 1

5 1 8 9 10 21 14 17

Low = 1 High = 0

1 5 8 9 10 21 14 17

Low = 7 High = 5

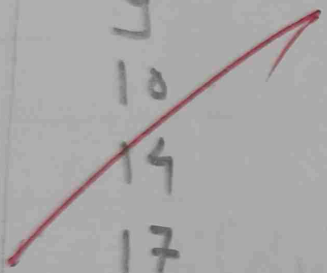
1 5 8 9 10 14 17 21

Low = 6 High = 5

1 5 8 9 10 14 17 21

The array after Quick Sort :

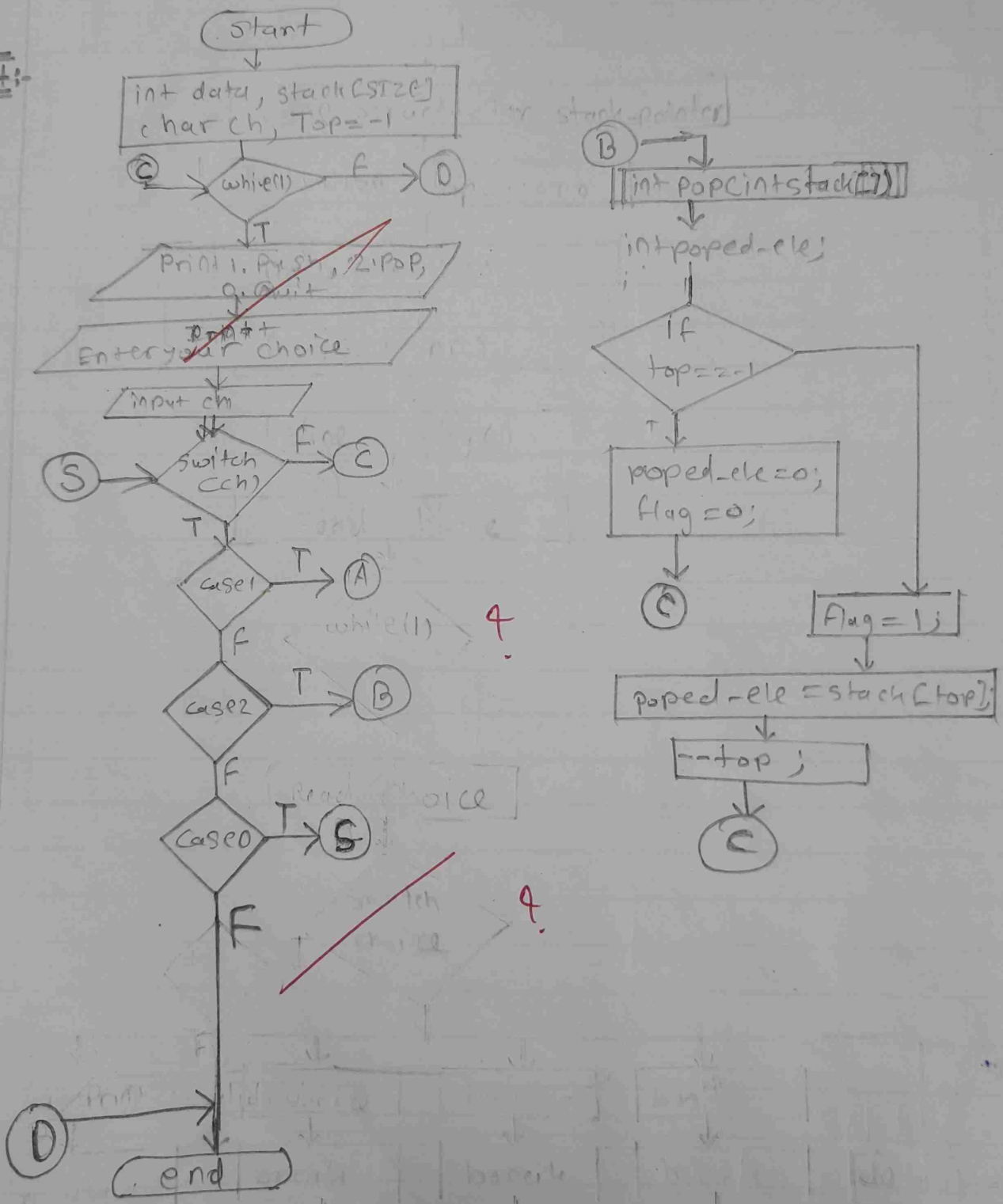
1
5
8
9
10
14
17
21

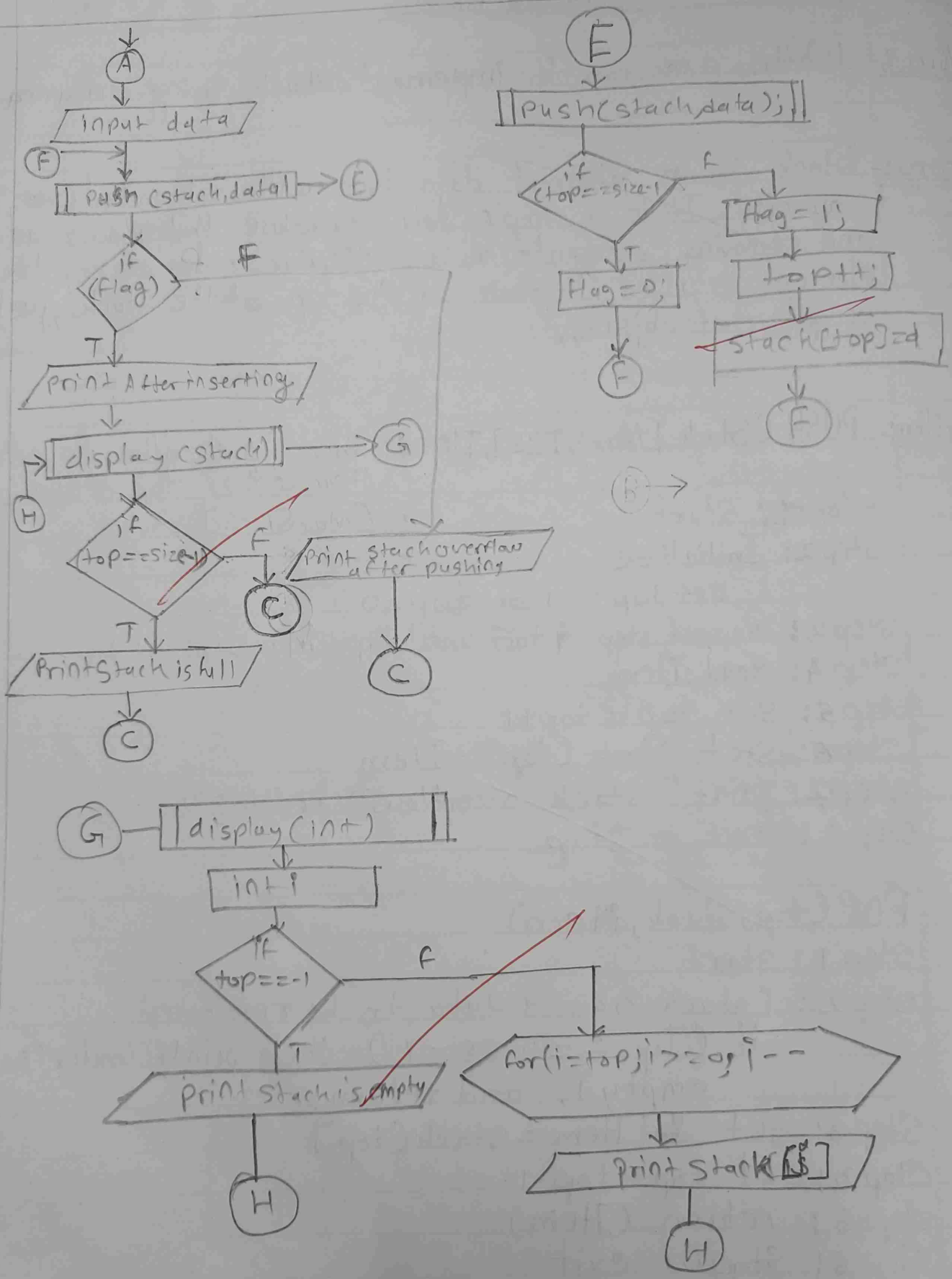


Practical No-3

Aim:- Write a program to implement Stack using an an array

Flow Chart:-




A
input data
F
push(stack,data) → E
if (flag)
F
T
print After inserting
display(stack) → G
H
if top==size-1
F
print stack overflow after pushing
C
T
Print Stack is full
C
C
E
push(stack,data);
if (top==size-1
F
T
flag=0;
flag=1;
top++;
stack[top]=d
F
F
R →
G
display(int)
int i
if top==-1
F
T
print stack is empty
for(i=top;i>=0;i--
print stack[i]
H
H

Output:

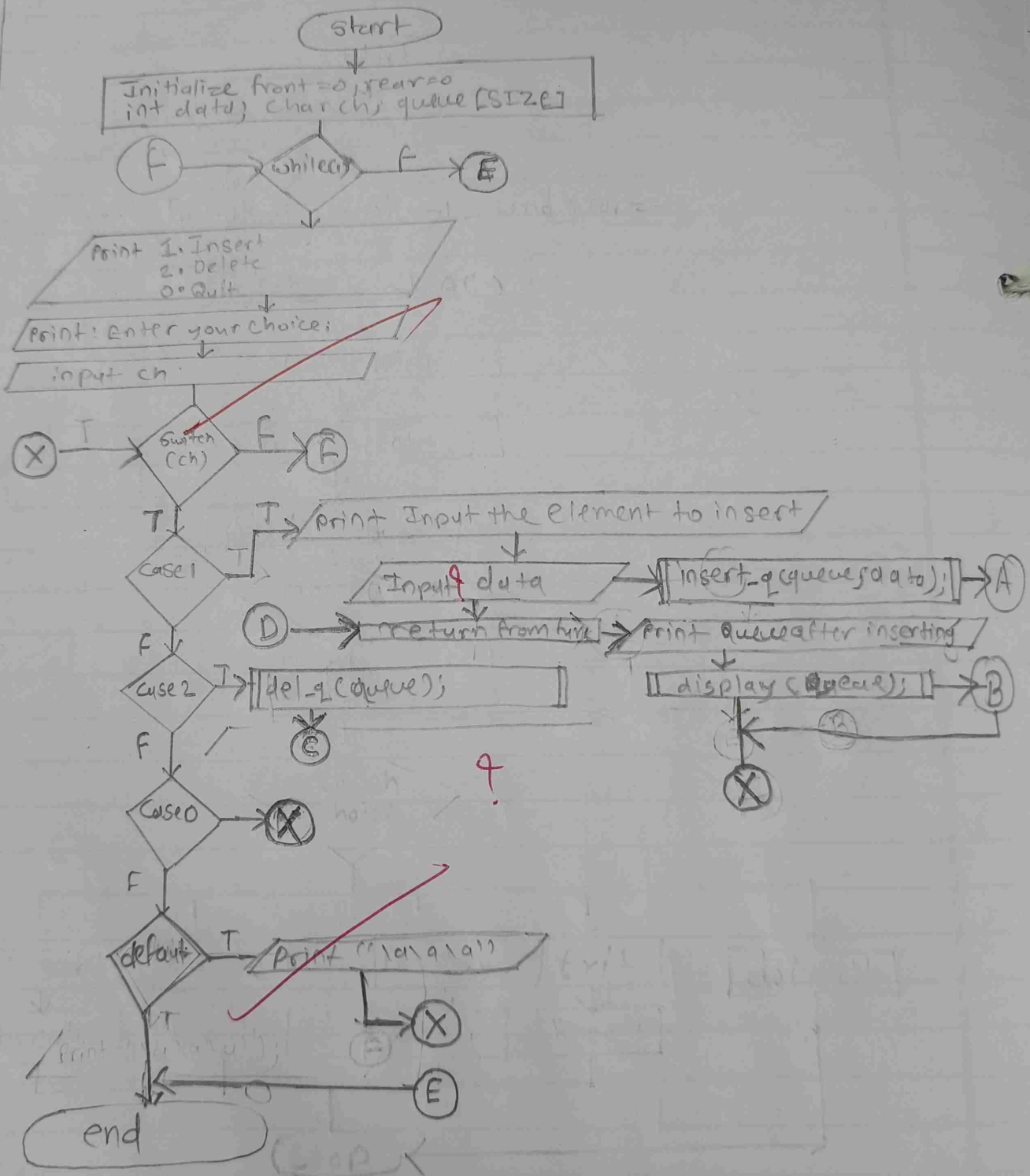
```
1. Push
2. POp
0. Quit

Enter your choice: 1
Input the element to push: 10

After inserting:
Stack elements are:
10

Enter your choice: 1
Input the element to push: 20

After inserting:
Stack elements are:
20
10

Enter your choice: 1
Input the element to push: 30

After inserting:
Stack elements are:
30
20
10

Enter your choice: 2
Data popped is: 30

Reset data in stack is a follows:
stack elements are:
10
```

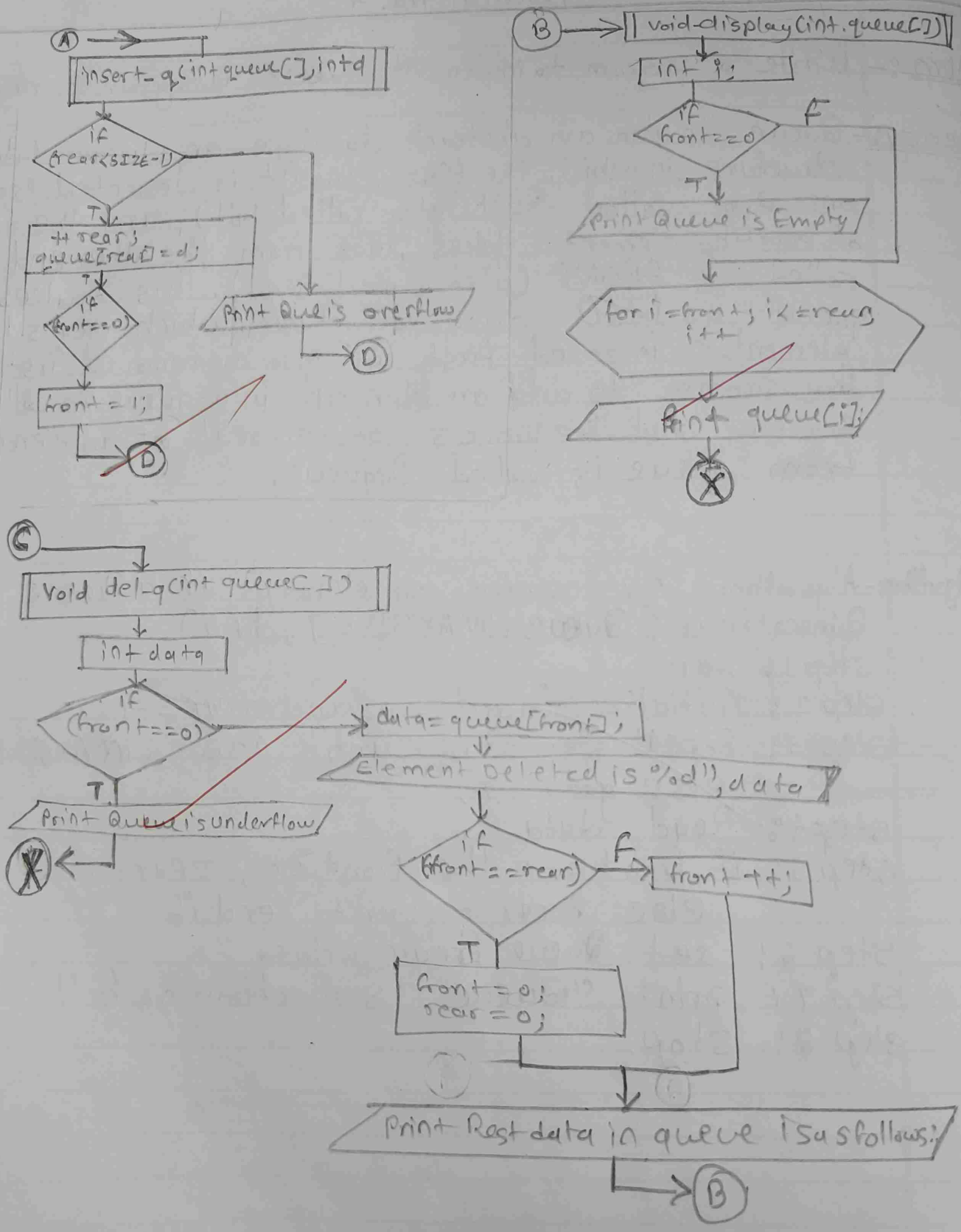
Enter your choice : 0

Practical No -4

Aim :- Write a program to implement Queue using an Array.

Flow chart :-




A
insert_q(int queue[], int d)
if (rear<SIZE-1)
T
++rear;
queue[rear] = d;
if (front==0)
front = 1;
Print Que is overflow
D
B
void display(int queue[])
int i;
if front==0
F
T
Print Queue is Empty
for i=front; i<=rear; i++
Print queue[i];
C
Void del_q(int queue[])
int data
if (front==0)
T
Print Queue is underflow
data = queue[front];
Element Deleted is %d", data
if (front==rear)
F
front ++;
front = 0;
rear = 0;
Print Rest data in queue is as follows:

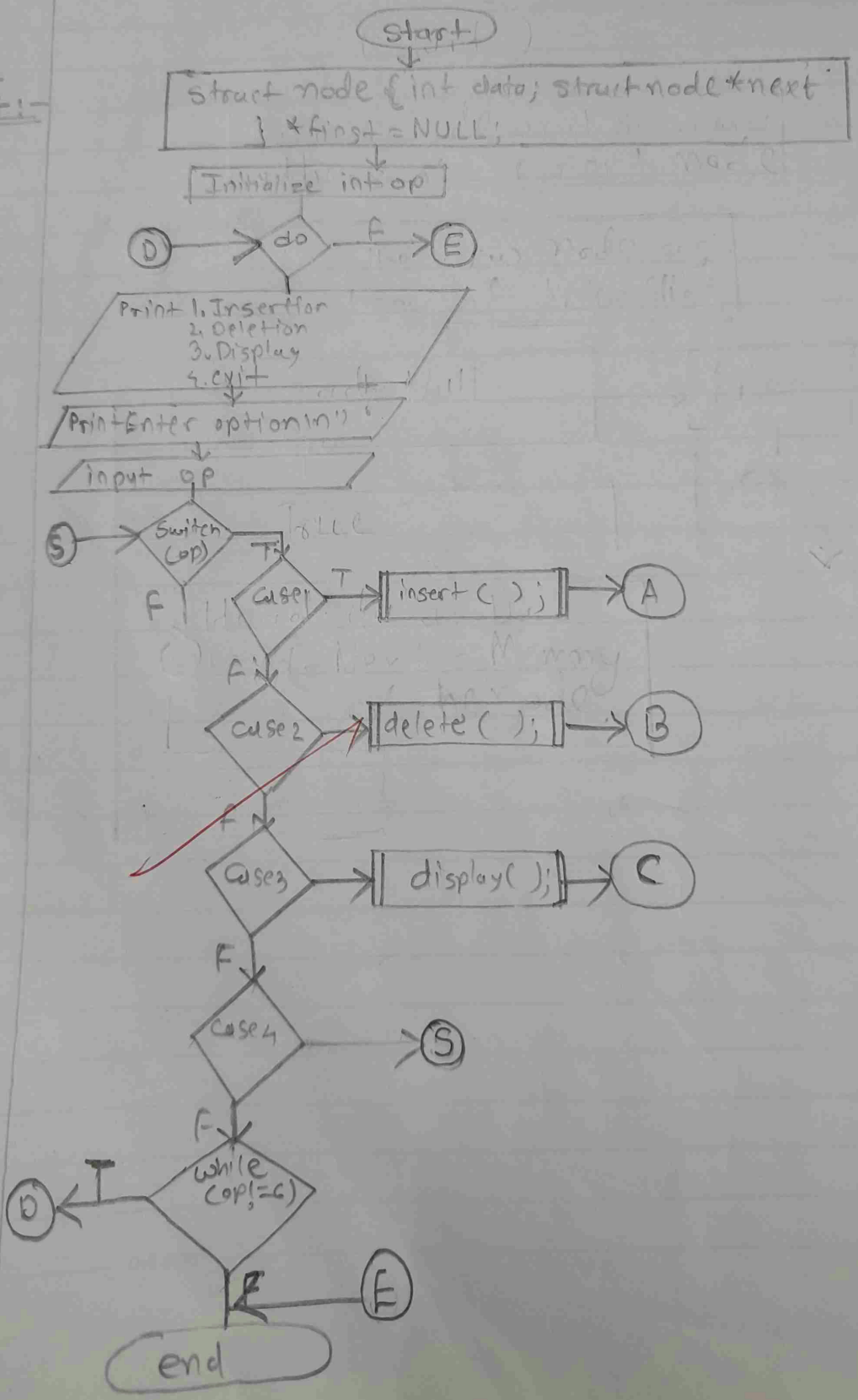
Output

```
1. Insert
2. Delete
0. Exit
Enter your choice : 1
Input the element to insert : 10
Que after inserting :
10
1. Insert
2. Delete
0. Exit
Enter your choice : 1
Input the element to insert : 20
Queue after inserting :
10 20
1. Insert
2. Delete
0. Exit
Enter your choice : 2
Deleted item : 10
Rest data in queue is as follows :
20
1. Insert
2. Delete
0. Exit
Enter your choice : 0
```

Practical No = 5

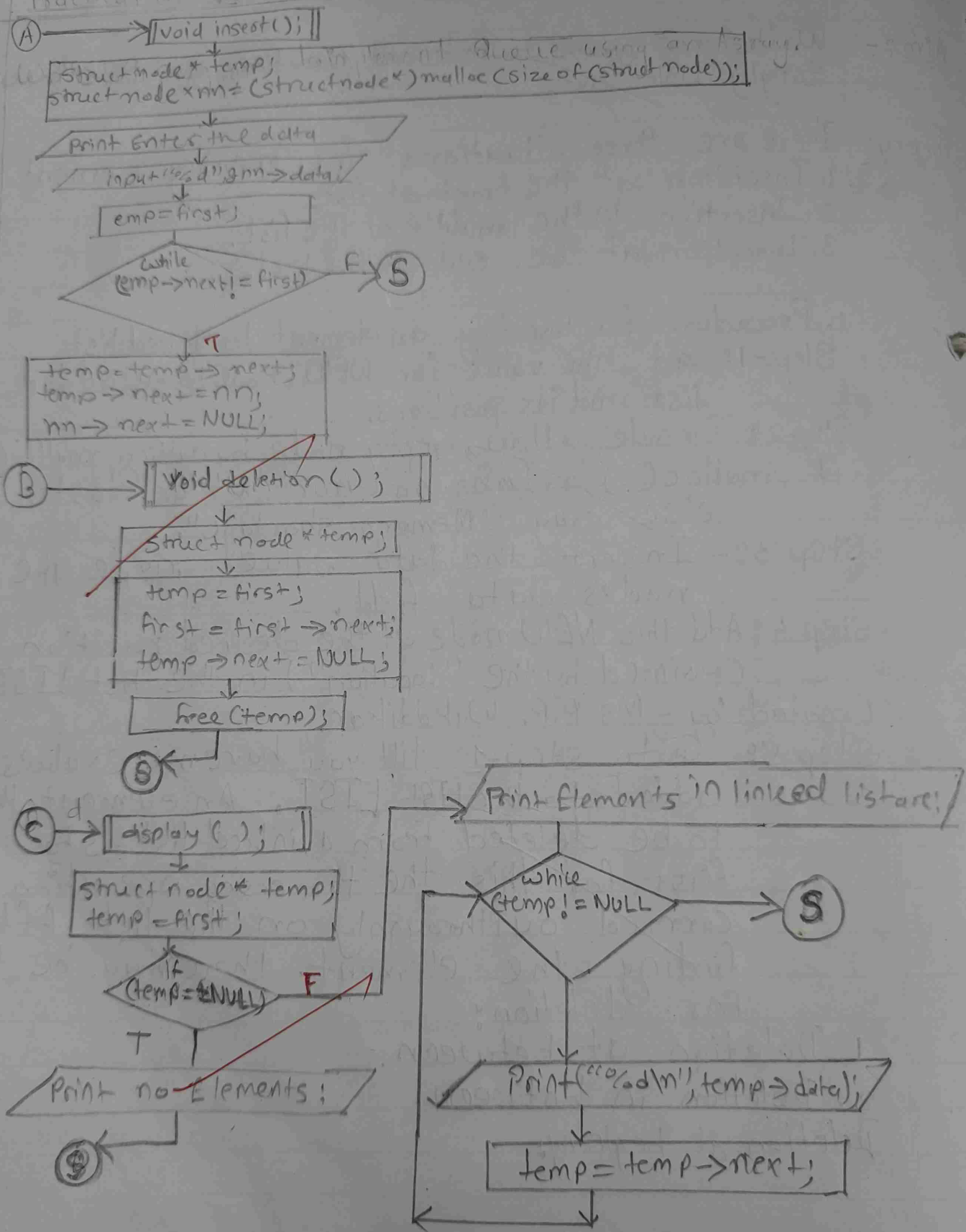
Aim :- Write a program to implement Insertion and Deletion on Singly linked list.

Flow chart :-



Practical No-4

Aim:-



Output :-

```
1. Insertion
2. Deletion
3. Display
4. Exit
1
Enter the data : 10
No elements in the linked list
```

(This message is not shown here because the list will now contain the element 10)

```
1. Insertion
2. Deletion
3. Display
4. Exit
1
Enter the data : 20
Elements in linked list are :
10
20
1. Insertion
2. Deletion
3. Display
4. Exit
1
Enter the data : 30
Elements in linked list are :
10
20
30
1. Insertion
2. Deletion
3. Display
4. Exit
2
Deleted first element
1. Insertion
2. Deletion
3. Display
4. Exit
```

3.
Elements in the list are:

20
30

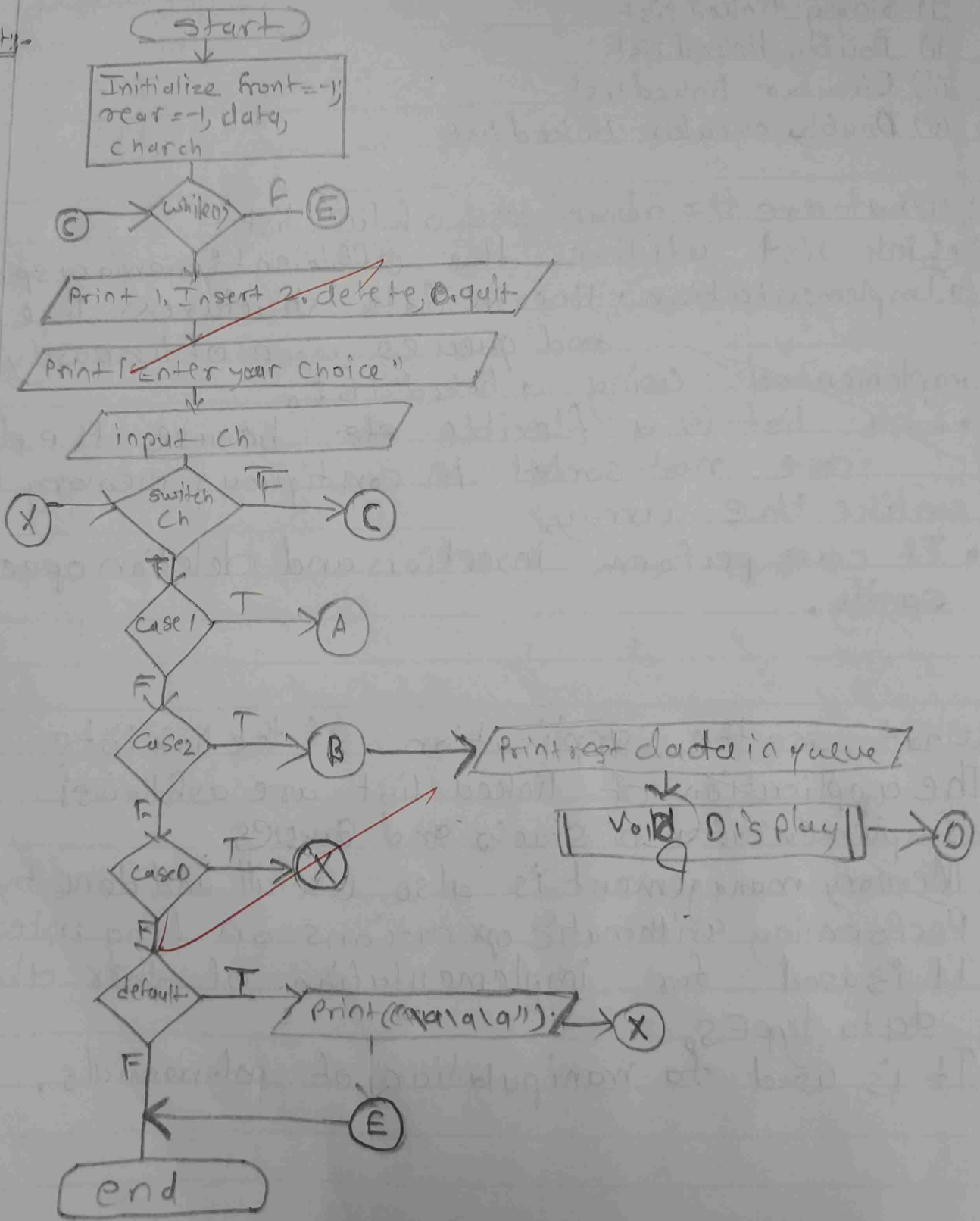
1. Insertion
2. Deletion
3. Display
4. Exit

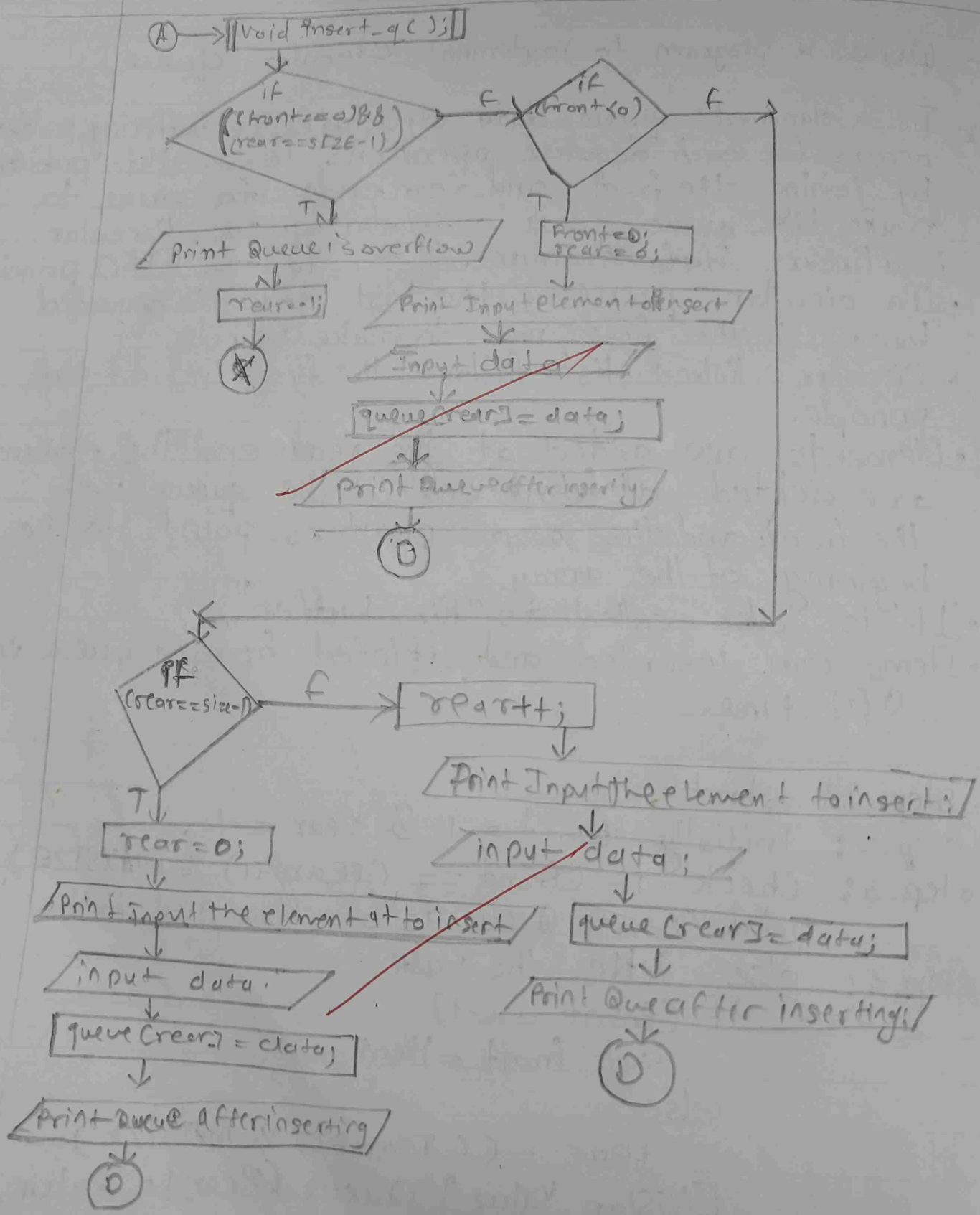
4

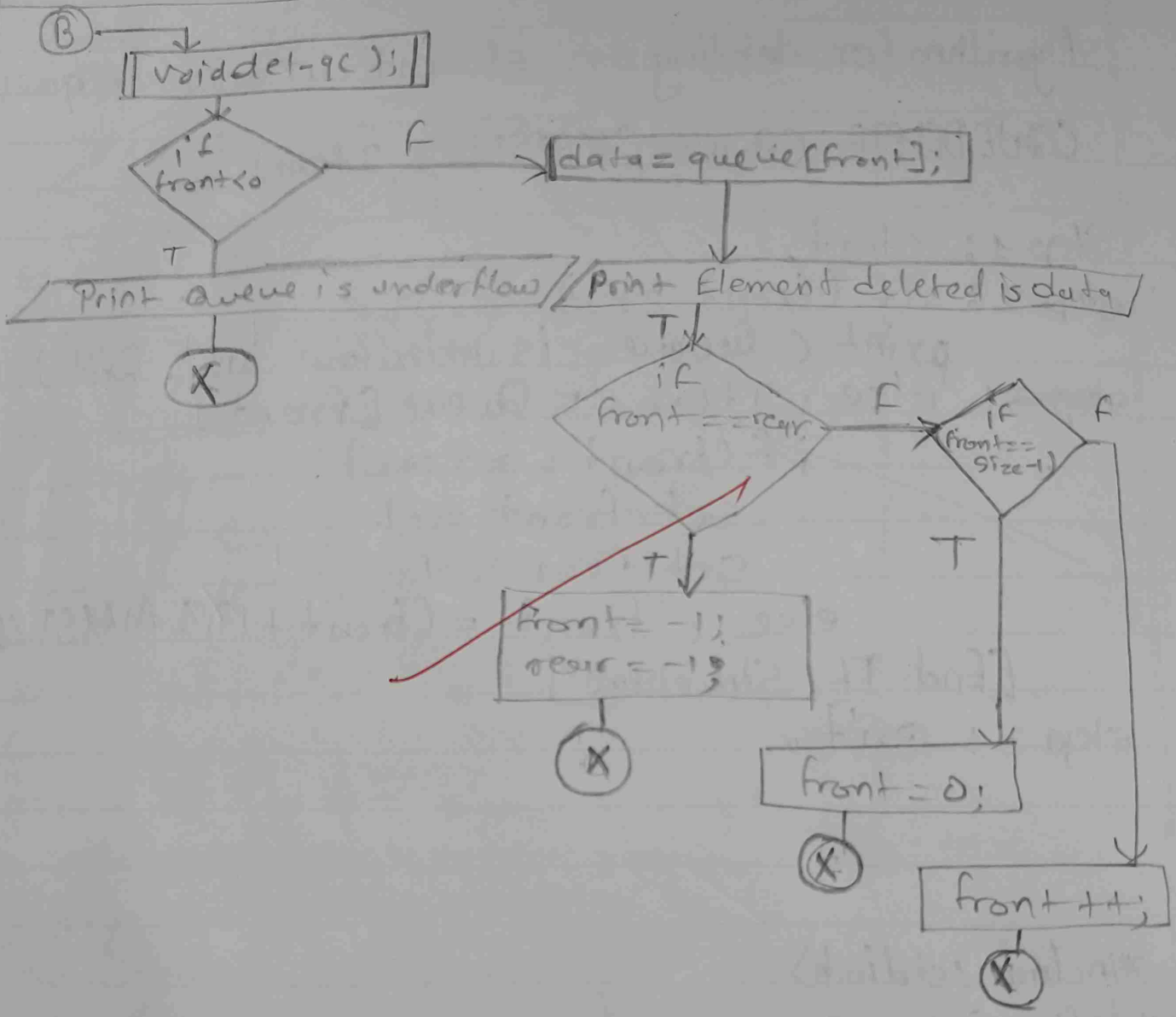
Practical No. 6

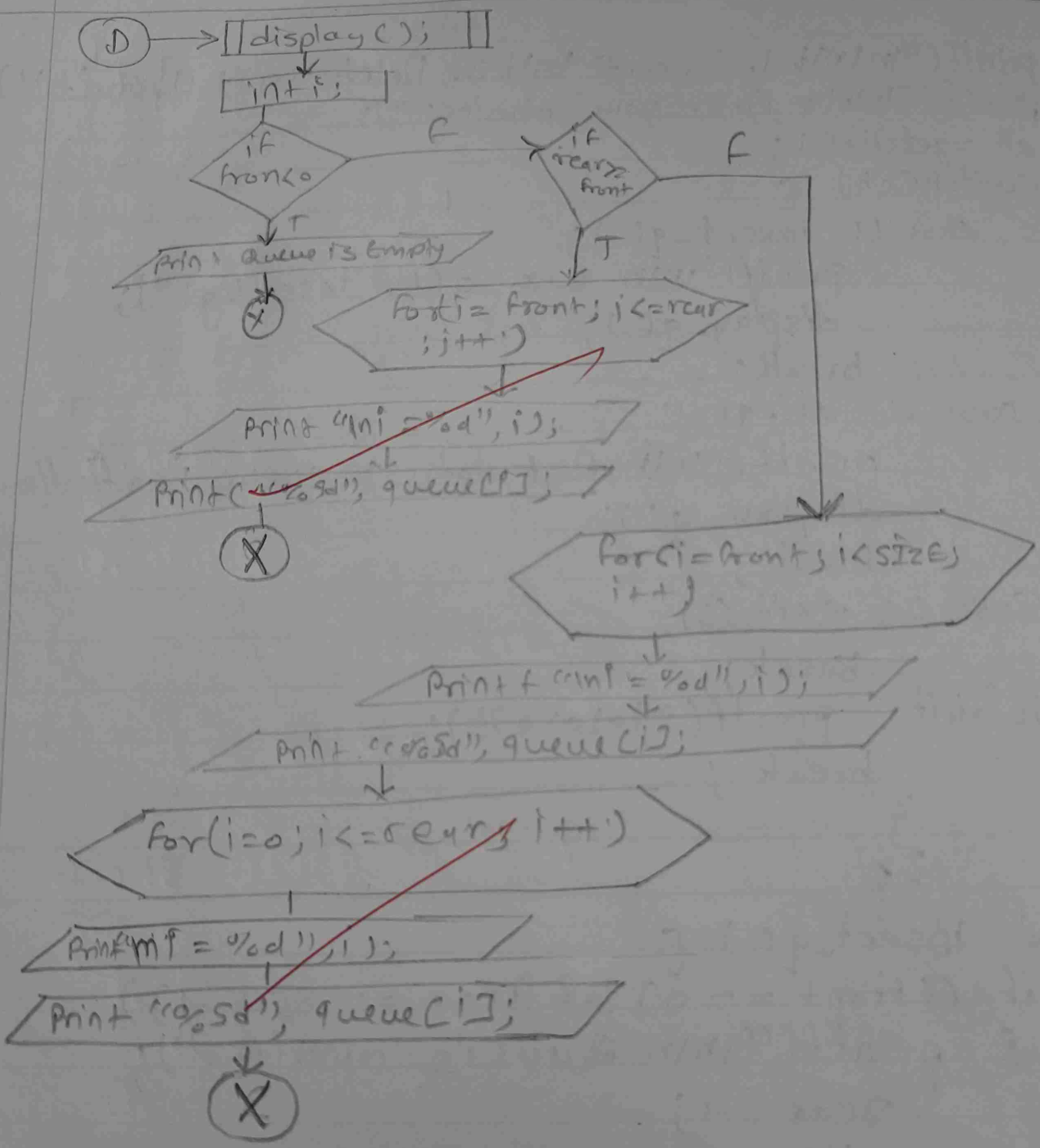
Aim:- Write a program to implement Circular Queue;

Flowchart:-




A
Void Insert_q( );
if ((front==0)&& (rear==SIZE-1))
F
if (front<0)
F
T
Print Queue is overflow
rear--;
T
front=0; rear=0;
Print Input element to insert
Input data
queue[rear]=data;
Print Queue after inserting
D
F
if (rear==size-1)
F
rear++;
Print Input the element to insert;
T
rear=0;
input data;
Print Input the element to insert
input data.
queue[rear]=data;
queue[rear]=data;
Print Que after inserting;
Print Queue after inserting
D
D


B
void del-qc();
if front<0
F
T
data = queue[front];
Print queue is underflow
Print Element deleted is data
X
T
if front == rear
F
if front == size-1
F
T
T
front = -1;
rear = -1;
X
front = 0;
X
front++;
X


D
display();
int i;
if front<0
F
T
Print "queue is Empty"
X
if rear< front
F
T
for(i= front; i<=rear ;i++)
Printf("\ni =%d", i);
Print("%5d", queue[i];
X
for(i=front; i<SIZE; i++)
Printf("\ni = %d", i);
Printf("%5d", queue[i];
for(i=0; i<=rear; i++)
Printf("i = %d", i);
Printf("%5d", queue[i];
X

Output:-

1. Insert
2. Delete
3. Quit

Enter your choice: 1

Enter the data: 5

Queue after inserting:

5

1. Insert
2. Delete
0. Quit

Enter your choice: 1

Enter the data = 8:

Queue after inserting:

58

1. Insert
2. Delete
0. Quit

Enter your choice: 2

Element deleted is: 5

Rest data in Queue is as follows:

8

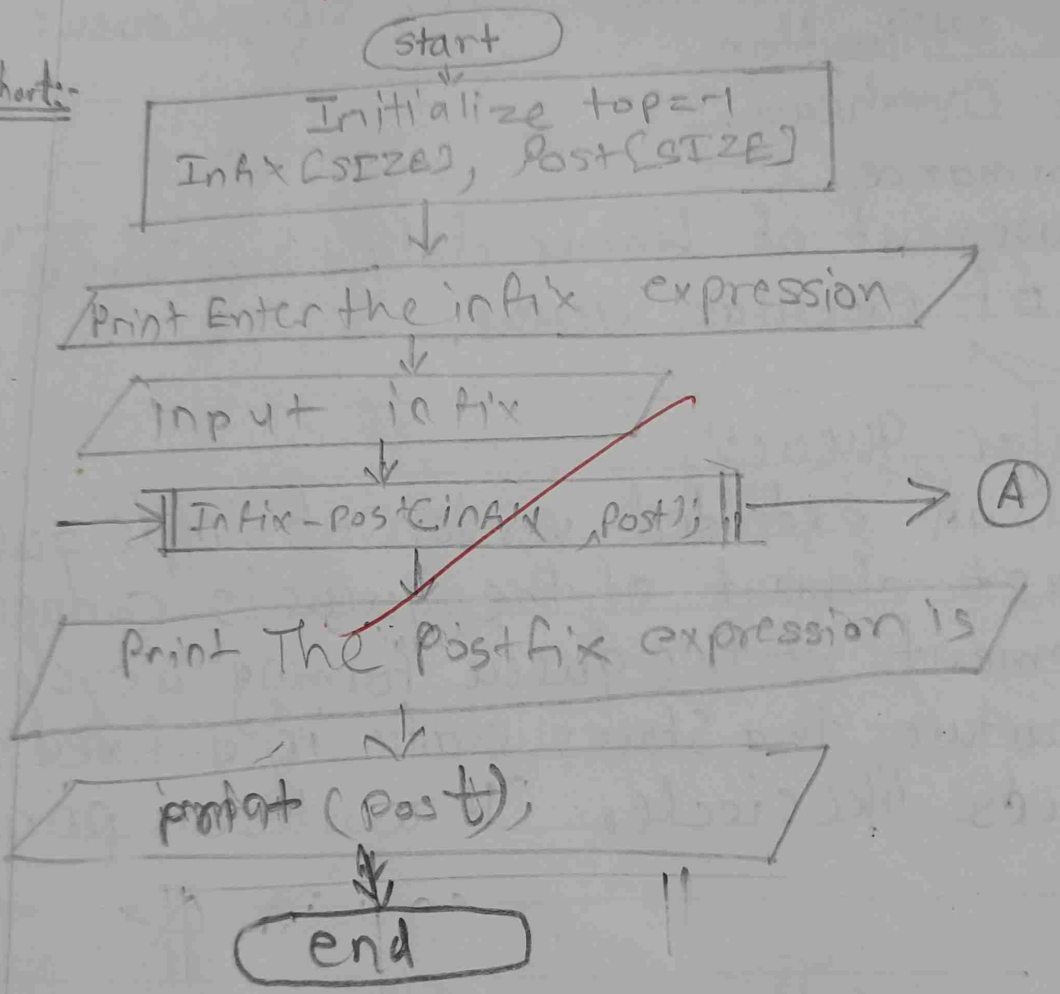
1. Insert
2. Delete
0. Quit

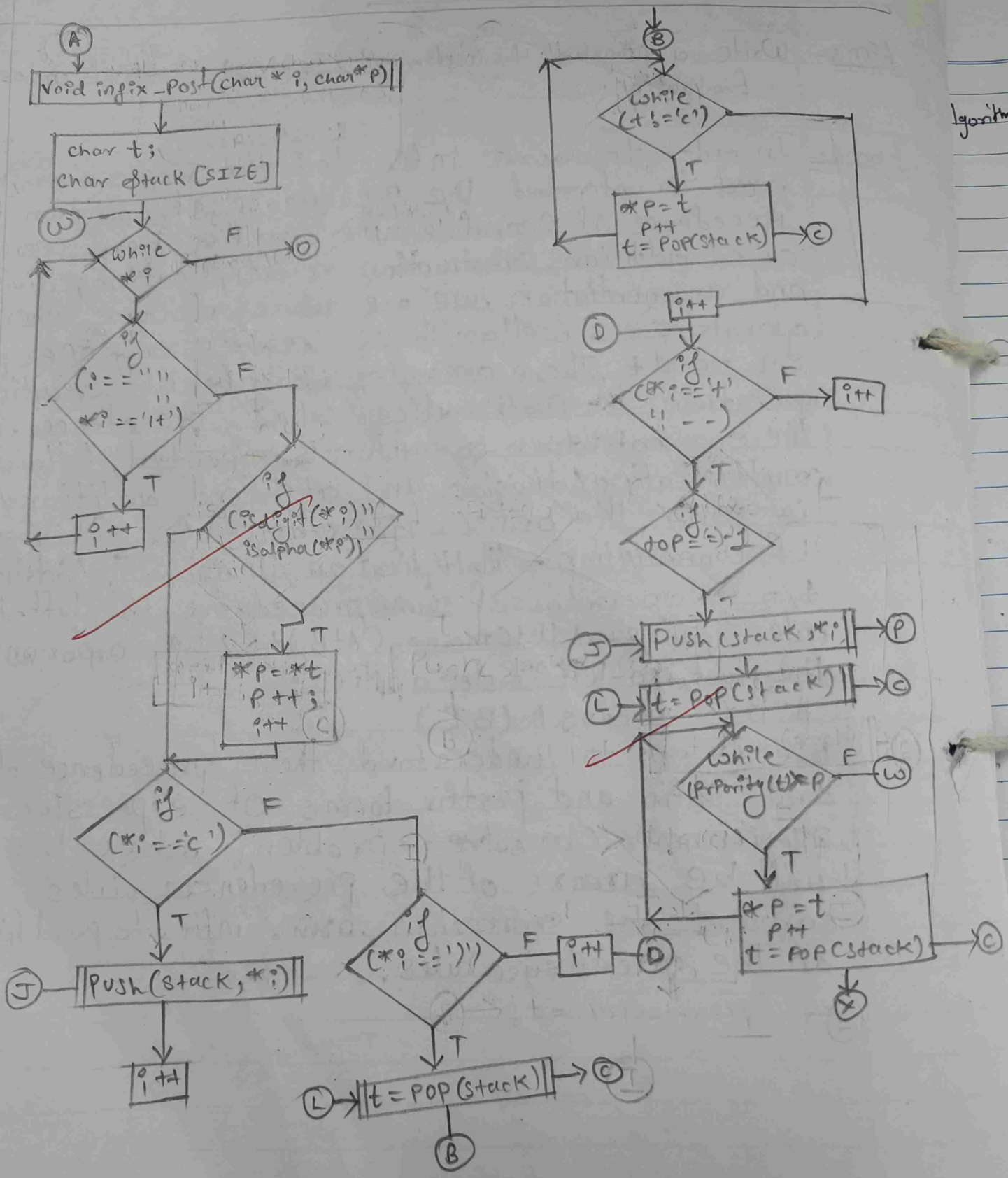
Enter your choice: 0

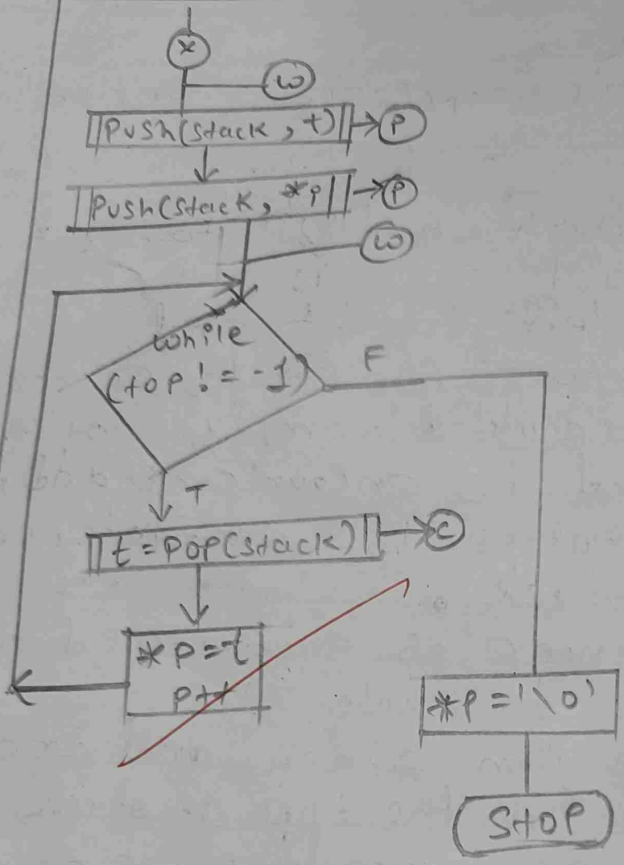
Practical No. 7 :

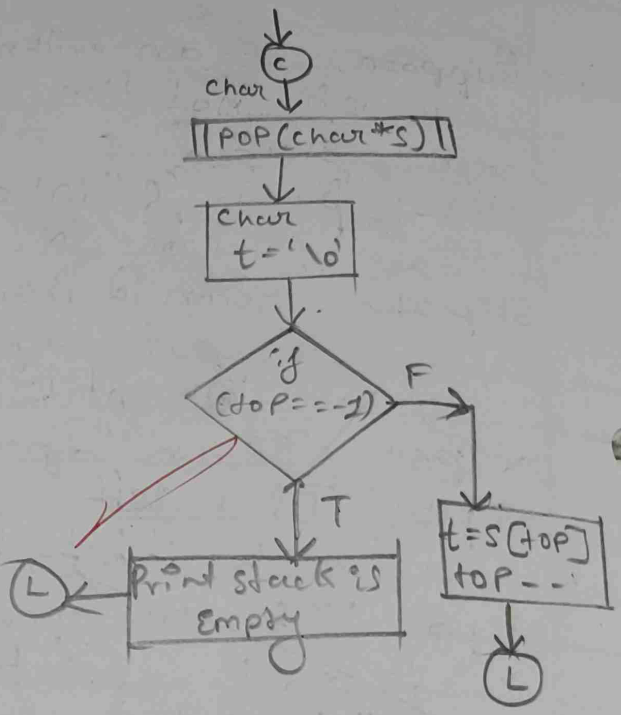
Aim :- Write a program to implement Infix to Postfix Expression Evaluation.

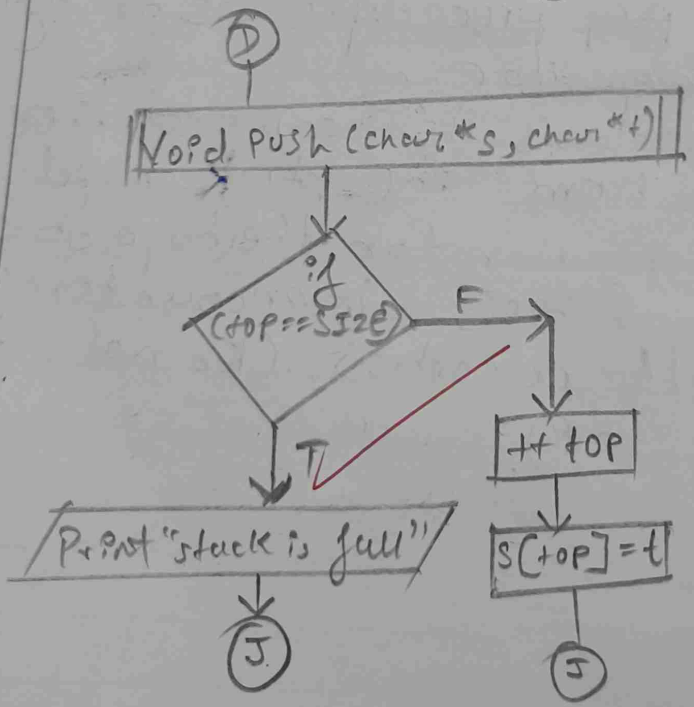
Flowchart :-




A
Void infix_Post(char *i, char*p)
char t;
char stack[SIZE]
W
while
*i
F
O
T
if
(i=='('
||
||
*i=='{')
i++
if
(isdigit(*i)||
isalpha(*p))
*p=*t
p++;
i++
C
if
(*i=='(')
T
J
Push(stack,*i)
i++
if
(*i==')')
F
i++
D
T
L
t=POP(stack)
B
while
(t!='(')
*p=t
p++
t=POP(stack)
C
i++
if
(*i=='+'
||
--)
F
i++
T
if
top==-1
T
J
Push(stack,*i)
P
L
t=POP(stack)
C
while
(Pr(t)>=Pr(*p)
F
W
T
*p=t
p++
t=POP(stack)
C
X


x
w
Push(stack, t)
P
Push(stack, *p)
P
w
while (top != -1)
F
T
t = Pop(stack)
c
*p = t
p++
*p = '\0'
STOP


c
char
Pop(char *s)
char t = '\0'
if (top == -1)
F
T
t = s[top]
top--
L
Print stack is Empty
L


D
Void push (char *s, char t)
if (top == SIZE)
F
T
++top
Print "stack is full"
s[top] = t
J
J

Output:

Enter the infix expression:

A+B*(C-D)

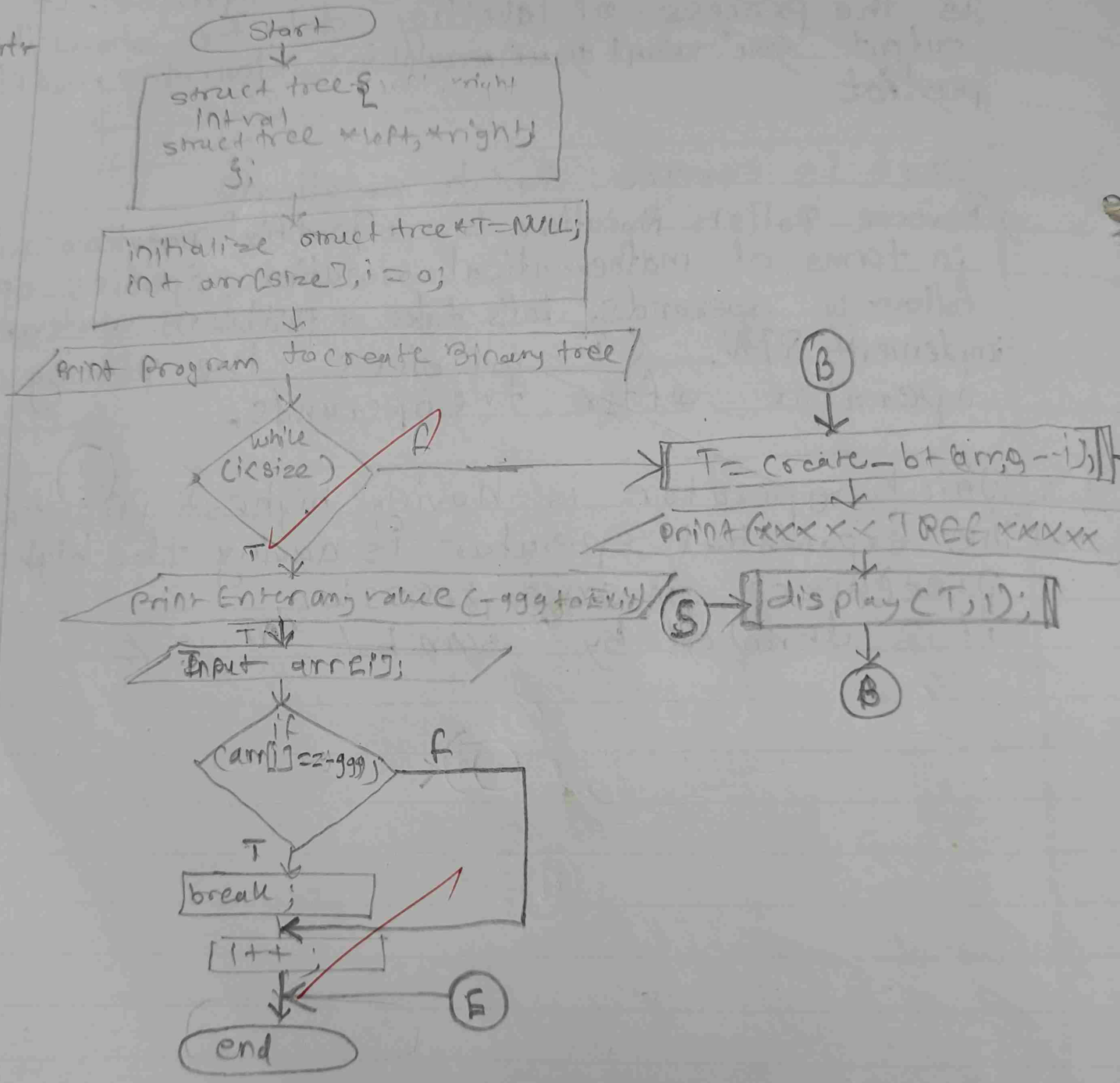
The Postfix expression is:

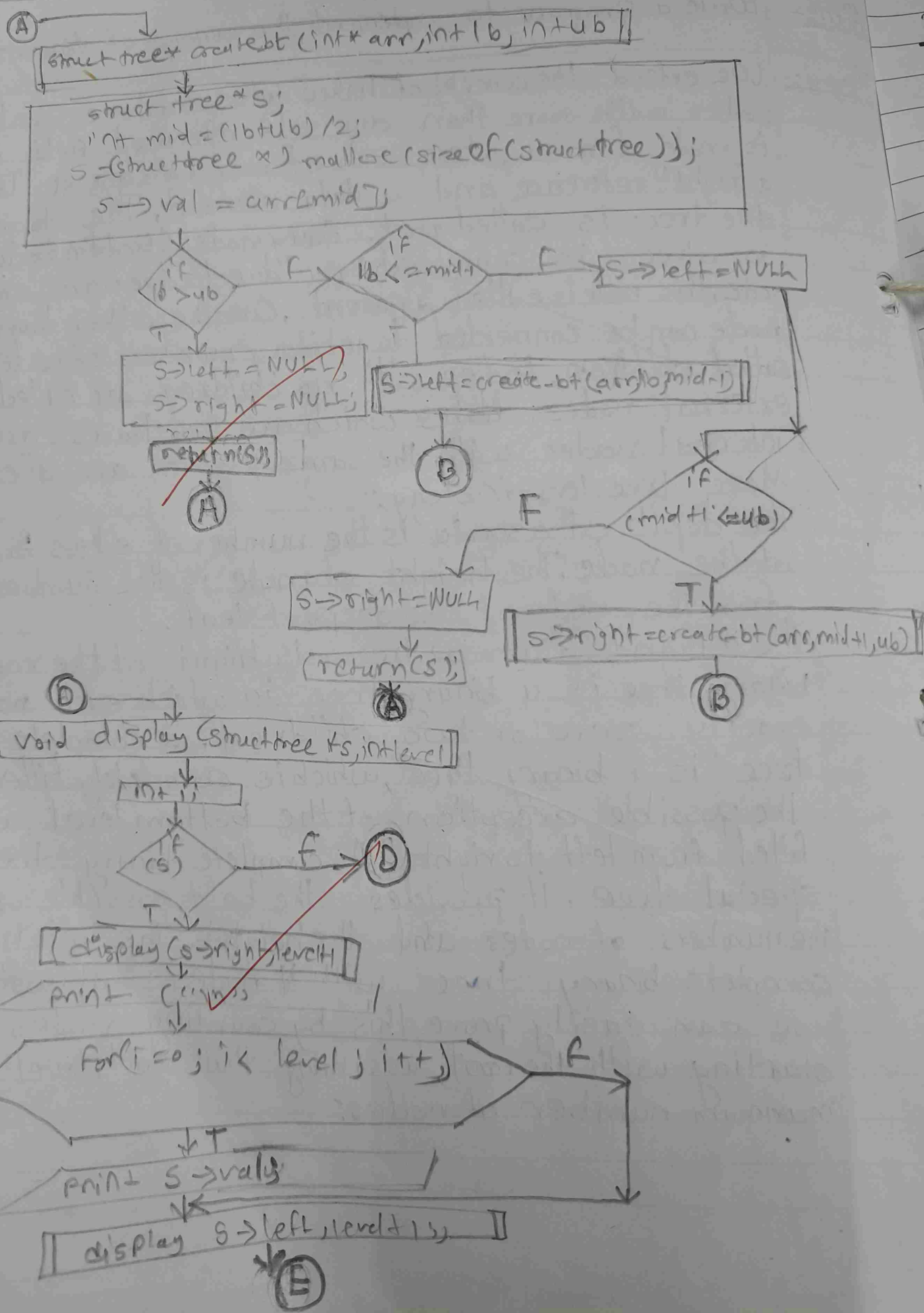
ABCD-*+

Practical No. 8

Aim:- Write a program to implement the concept of binary tree.

flowchart:-

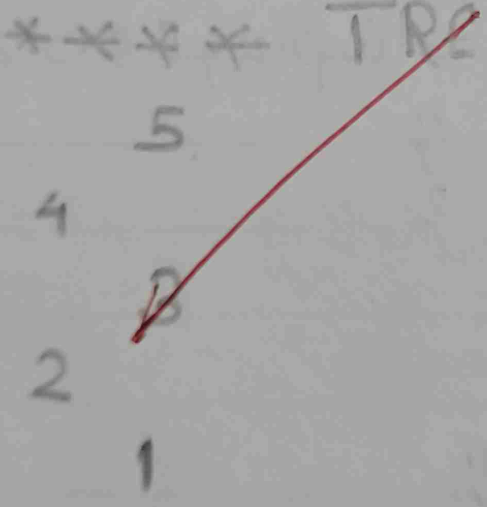



A
struct tree* creatbt(int* arr, int lb, int ub
struct tree *S;
int mid = (lb+ub)/2;
S=(struct tree *) malloc(sizeof(struct tree));
S->val = arr[mid];
if lb>ub
T
F
S->left=NULL;
S->right=NULL;
return(S);
A
IF lb<=mid-1
T
F
S->left=creatc_bt(arr,lb,mid-1)
B
S->left=NULL
IF (mid+1<=ub)
T
F
S->right=creatc_bt(arr,mid+1,ub)
B
S->right=NULL
return(S);
A
D
void display(struct tree *s, int level)
int i;
if (s)
T
F
D
display(s->right, level+1)
print("\n");
for(i=0; i< level; i++)
T
F
print s->val;
display s->left, level+1);
E

Output: Program to create binary Tree.

Enter any value : (-999 to Exit): 1
Enter any value : (-999 to Exit): 2
Enter any value : (-999 to Exit): 3
Enter any value : (-999 to Exit): 4
Enter any value : (-999 to Exit): 5
Enter any value : (-999 to Exit): -999

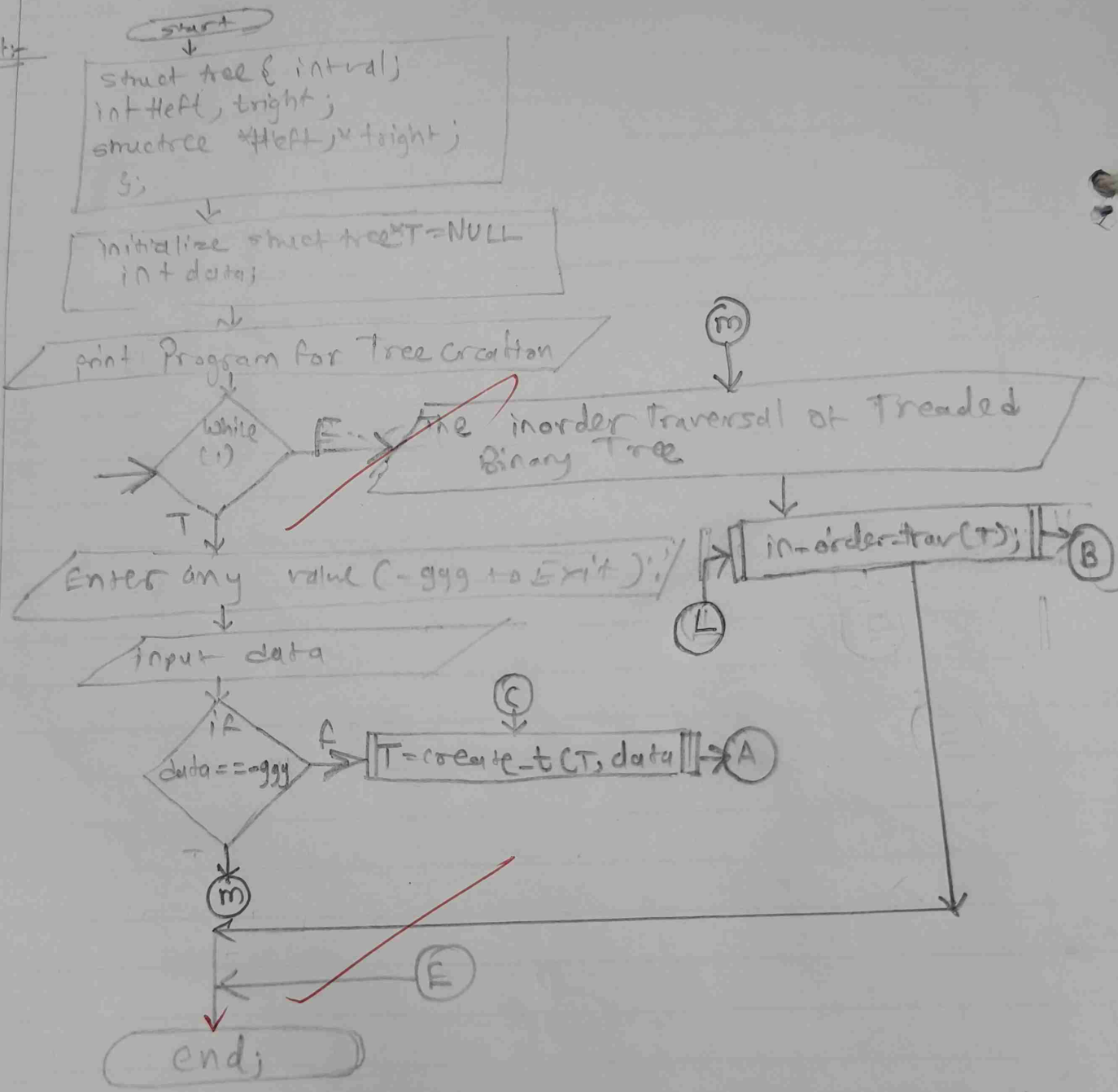
**** TREE ****

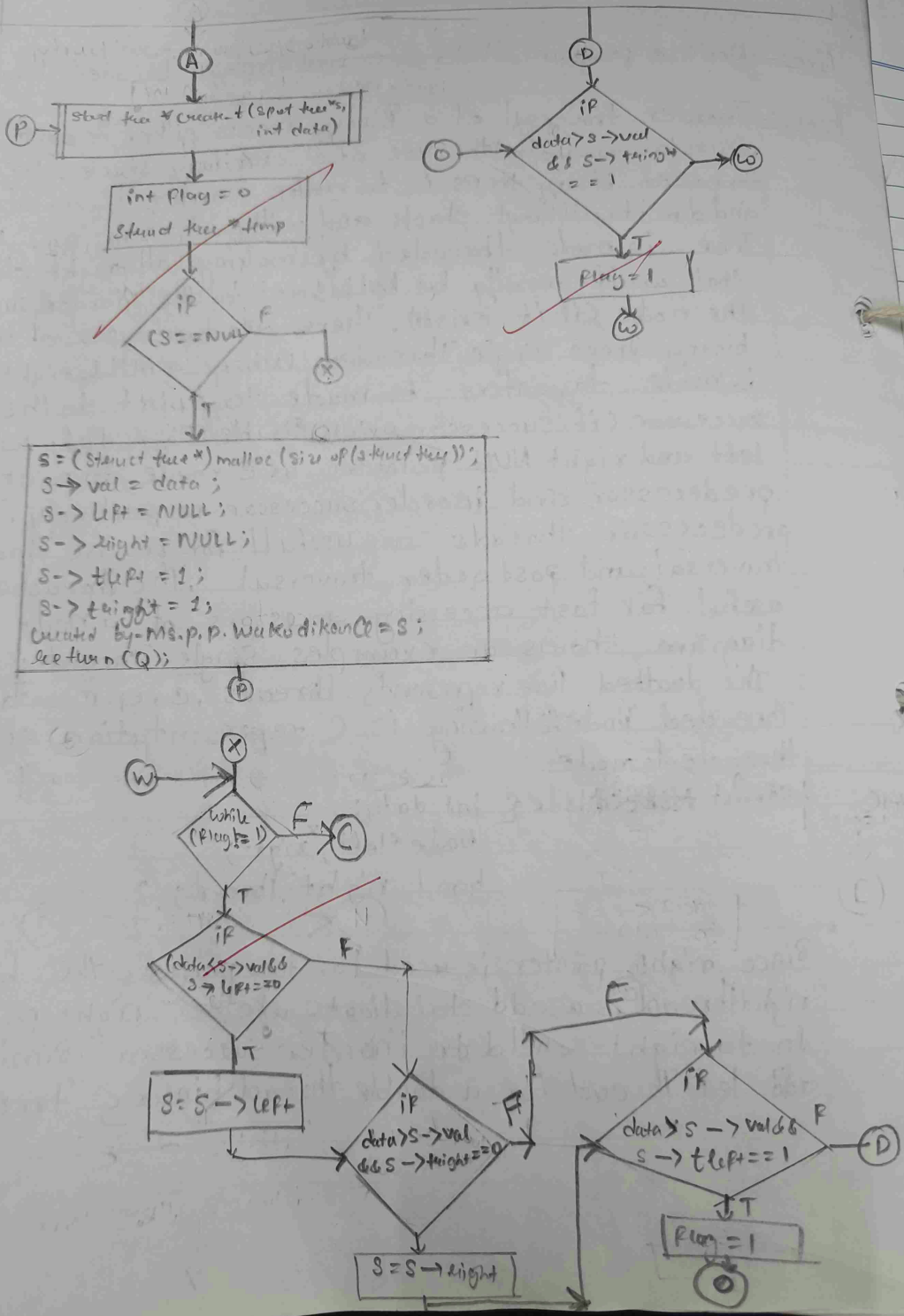


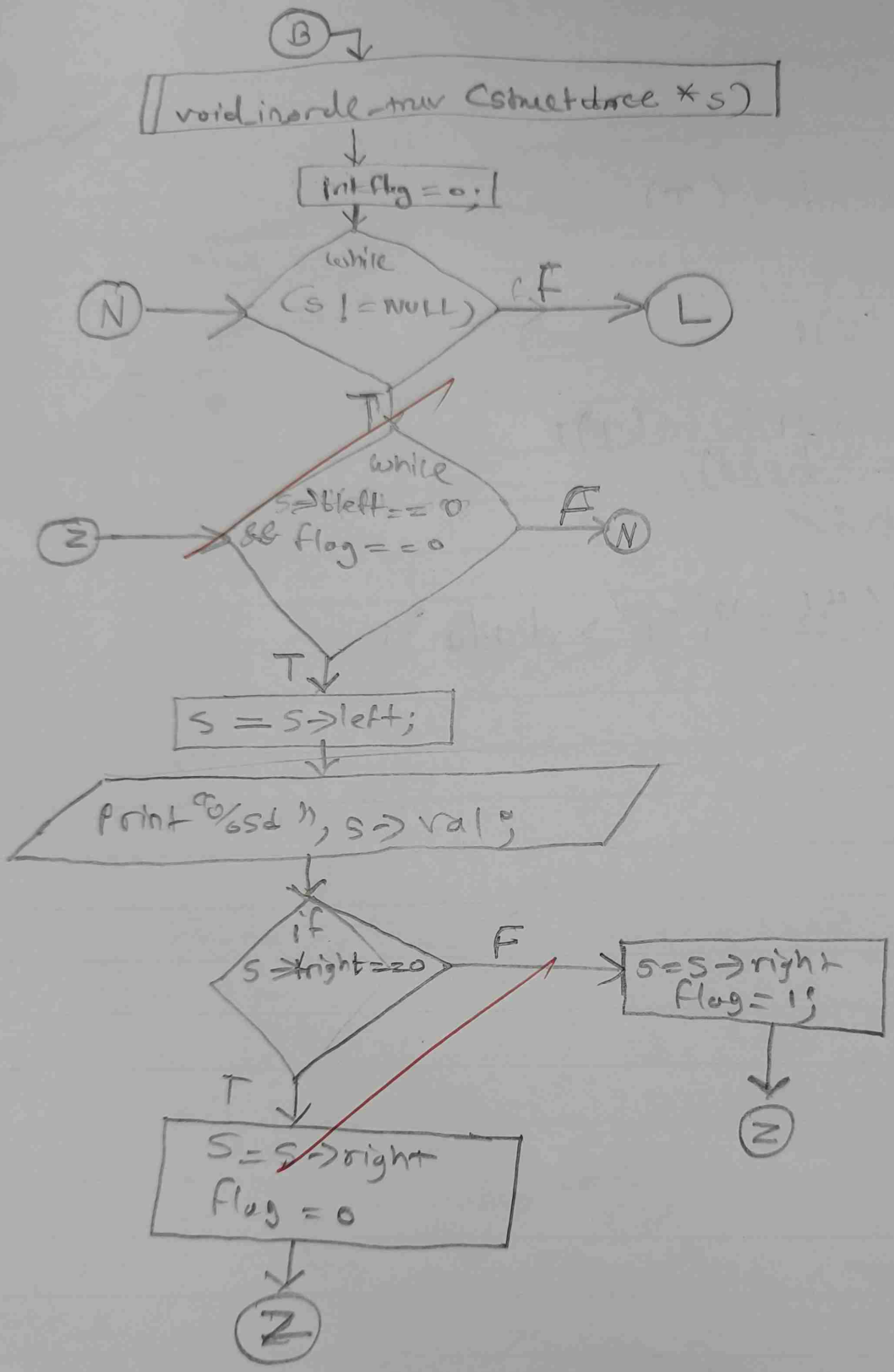
Practical No. 9

Aim:- Write a program to create and display threaded binary tree.

Flowchart:-




A
P
struct tree *create_t(struct tree *s, int data)
int flag = 0
struct tree *temp
if (S == NULL)
F
X
T
s = (struct tree *) malloc (size of (struct tree));
S -> val = data;
S -> left = NULL;
S -> right = NULL;
S -> tleft = 1;
S -> tright = 1;
Created by-MS.P.P. Wakodikar
CC = S;
return (Q);
P
D
if
data > s -> val && S -> tright == 1
O
Lo
T
flag = 1
W
W
X
while (flag != 1)
F
C
T
if
(data < S -> val && S -> left == 0
F
T
S = S -> left
if
data > S -> val && S -> tright == 0
F
F
if
data > S -> val && S -> tleft == 1
F
D
T
flag = 1
O
S = S -> right


B
void inorder_trav (struct tree *s)
int flag = 0;
while
(s != NULL)
F
N
L
T
while
s->left == 0
&& flag == 0
F
Z
N
T
s = s->left;
print "%d", s->val;
if
s->right == 0
F
s = s->right
flag = 1;
Z
T
s = s->right
flag = 0
Z

Output:

```
Program for Tree creation:-
Enter any value:(-999 to Exit): 50
Enter any value:(-999 to Exit): 30
Enter any value:(-999 to Exit): 70
Enter any value:(-999 to Exit): 20
Enter any value:(-999 to Exit): 40
Enter any value:(-999 to Exit): 60
Enter any value:(-999 to Exit): 80
Enter any value:(-999 to Exit)-999
```

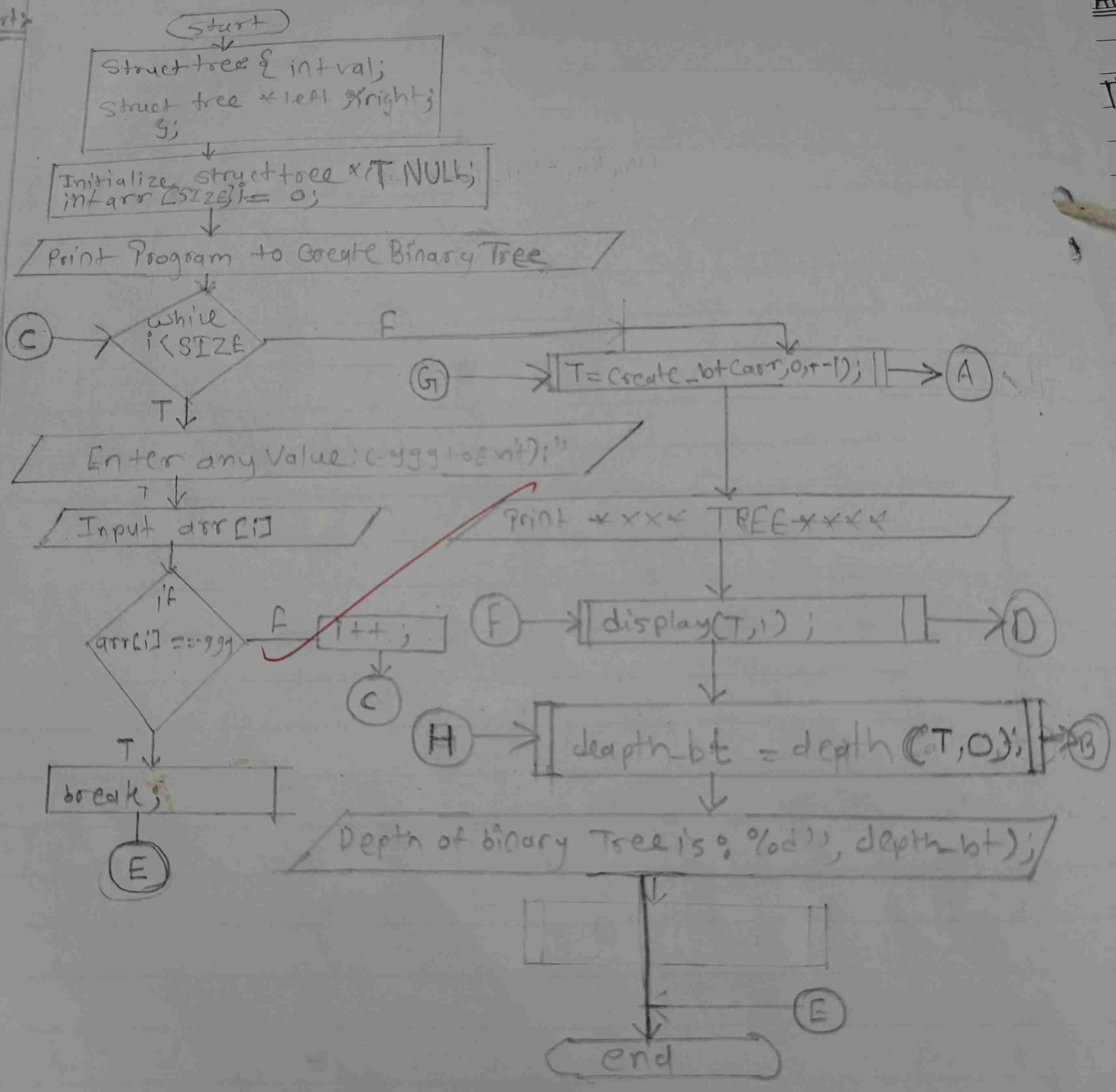
The in-order Traversal of a Threaded Binary Tree.

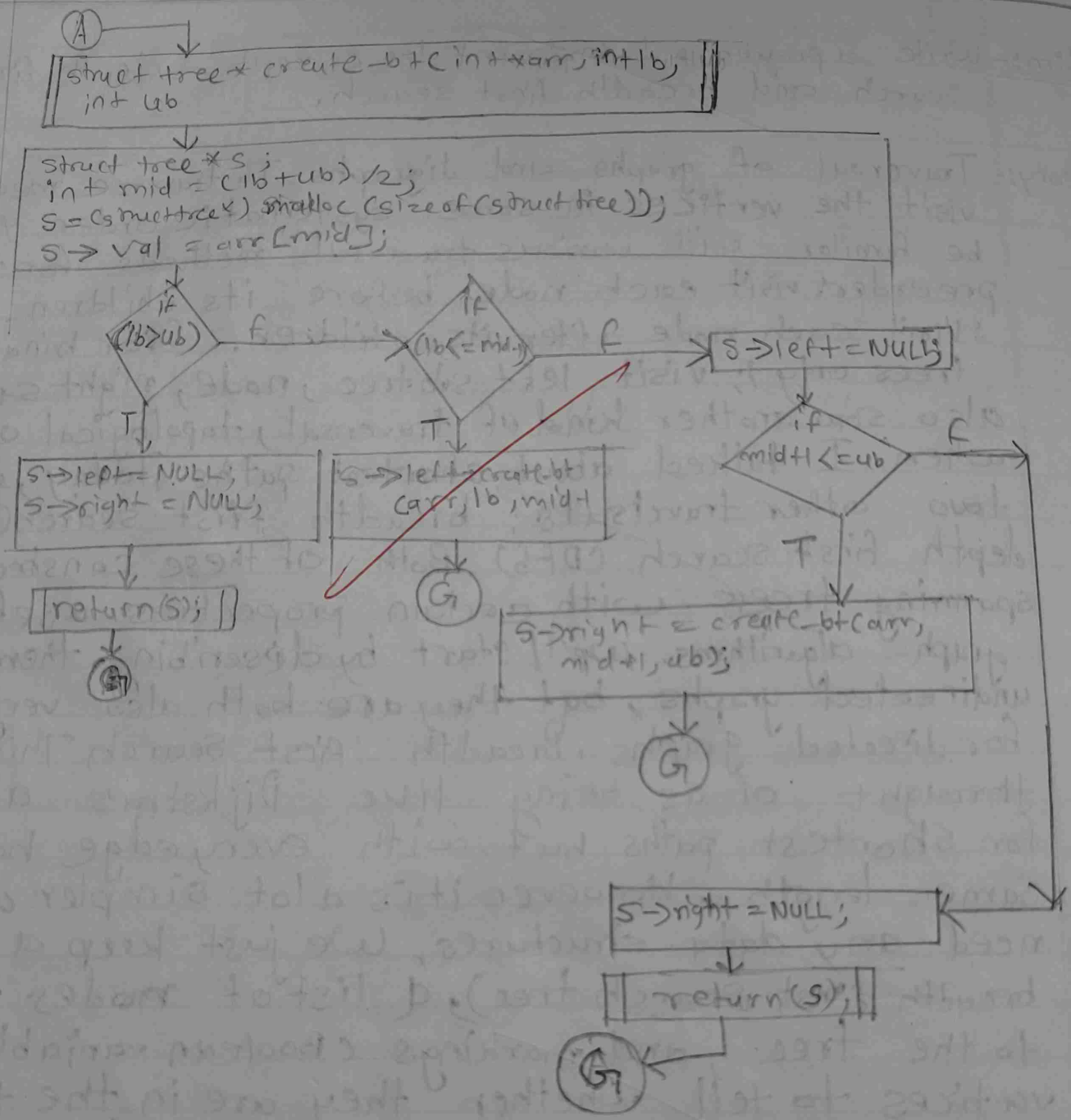
```
20
30
40
50
60
70
80
```

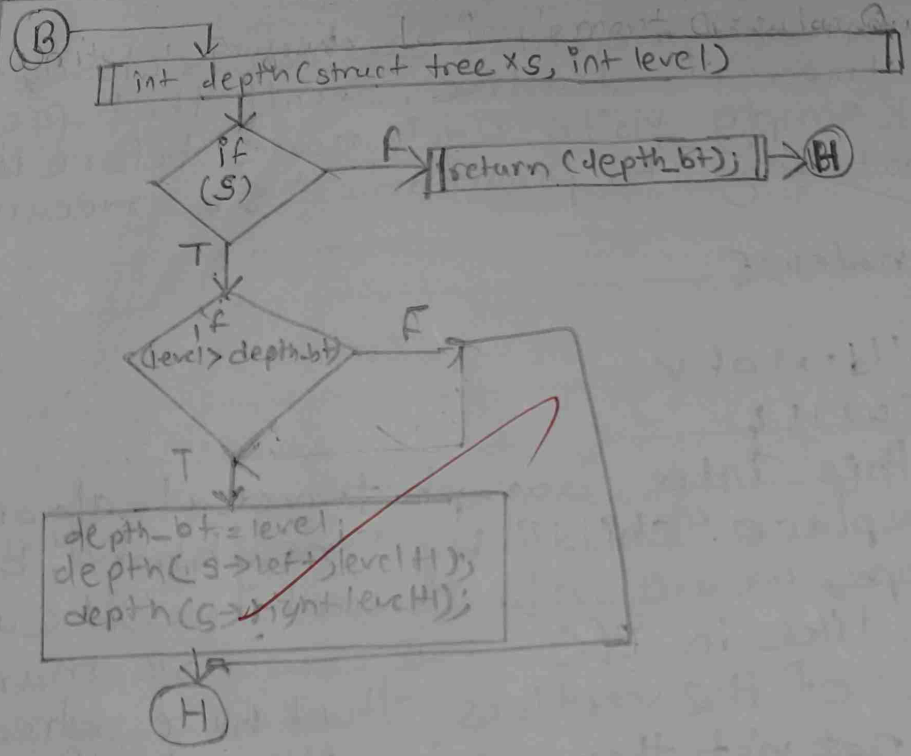
Practical No: 10

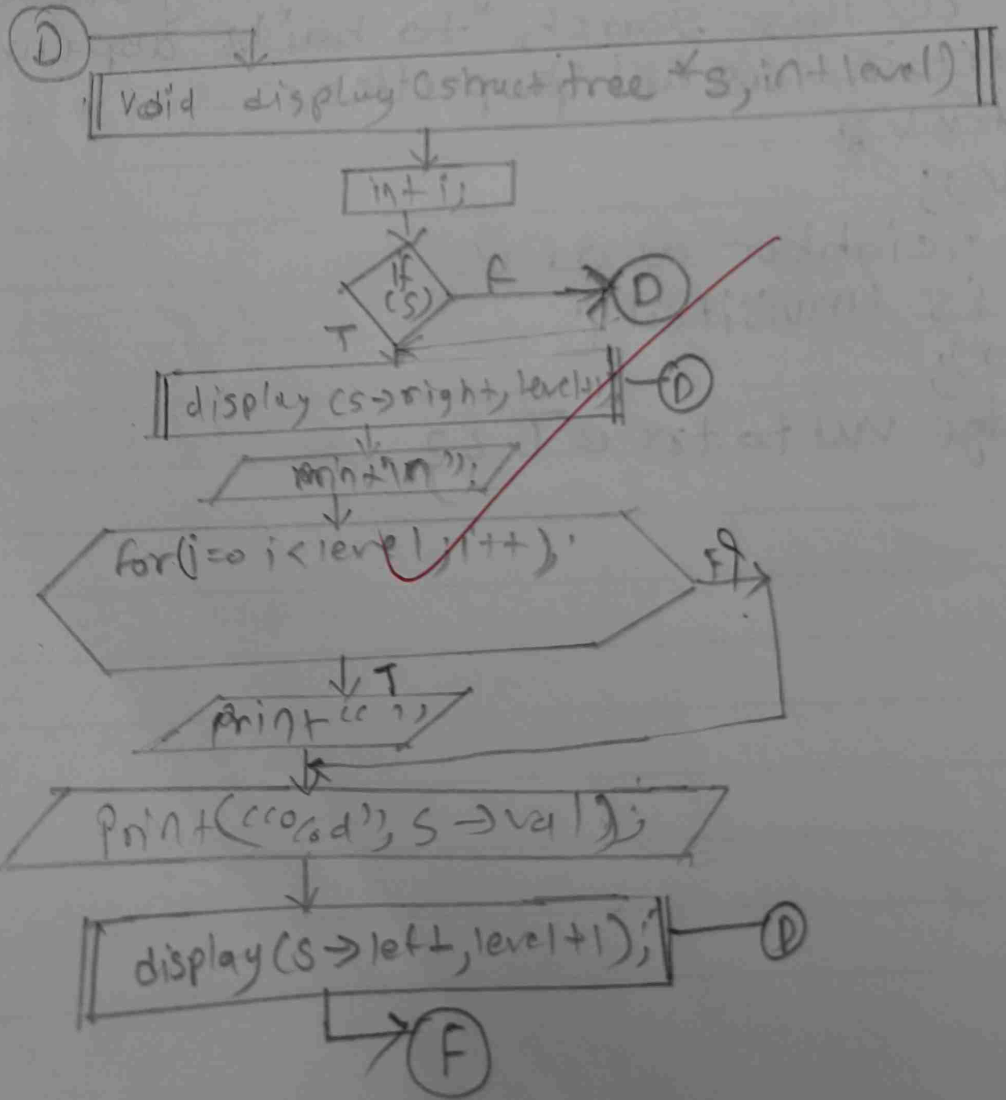
Aim:- Write a program to implement the concept of depth first search and breadth first search.

Flowchart:-




A
struct tree* create_bt(int *arr, int lb,
int ub
struct tree *s;
int mid = (lb+ub)/2;
s=(struct tree*) malloc (sizeof(struct tree));
s->val = arr[mid];
if (lb>ub)
F
T
if (lb<=mid)
F
T
s->left=NULL;
if mid+1<=ub
F
T
s->left=NULL;
s->right = NULL;
s->left=create_bt (arr, lb, mid-1
G
s->right = create_bt(arr, mid+1, ub);
G
return(s);
G
s->right=NULL;
return(s);
G

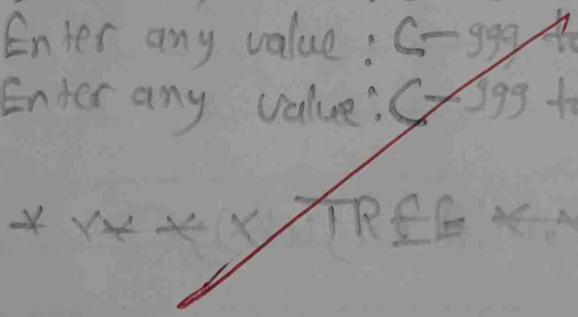

B
int depth(struct tree *s, int level)
if (s)
F
return (depth_bt);
H
T
if (level > depth_bt)
F
T
depth_bt = level;
depth(s->left, level+1);
depth(s->right, level+1);
H


D
void display(struct tree *s, int level)
int i;
if (s)
F
D
T
display(s->right, level+1);
D
printf("\n");
for(i=0 i<level; i++);
F
T
printf(" ");
printf("%d", s->val);
display(s->left, level+1);
D
F

Program to create Binary Tree.

Output:

```
Enter any value: (-999 to Exit): 10
Enter any value: (-999 to Exit): 20
Enter any value: (-999 to Exit): 30
Enter any value: (-999 to Exit): 40
Enter any value: (-999 to Exit): 50
Enter any value: (-999 to Exit): 60
Enter any value: (-999 to Exit): 70
Enter any value: (-999 to Exit): 80
Enter any value: (-999 to Exit): 90
Enter any value: (-999 to Exit): -999
```



```
        90
      80
    70
  60
50
  40
    30
      20
        10
```

Depth of Binary Tree is: 4