NEW EDITION

(DATA STRUCTURES LABORATORY)

(III Semester of B.Tech)

As per the curricullam and syllabus
of
Bharath Institute of Higher Education & Research

(Data Structures Laboratory)

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)
ACCREDITED WITH 'A' GRADE BY NAAC

PREPARED BY
Mrs.S Jenita Christy

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Declared as Deemed-to-be University under section 3 of UGC Act, 1956)

(Vide Notification No. F.9-5/2000 - U.3, Ministry of Human Resource Development, Govt. of India, dated 4th July 2002)

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

SUBJECT NAME: DATA STRUCTURES

SUBJECT CODE: BCS3L1

Regualtion R 2015

(2015-2016)

BCS3L1	DATASTRUCTURES USING C LAB	L	T	P	C
	Total Contact Hours - 30	0	0	3	2
	Prerequisite –Fundamental of Computing and Programming, Data Structures, C Programming.				
	Lab Manual Prepared by – Dept. of Computer Science & Engineering				

OBJECTIVES

This course demonstrates familiarity with major algorithms and data structures and analyzes performance of algorithms. It is used to choose the appropriate data structure and algorithm design method for a specified application and determine which algorithm or data structure to use in different scenarios.

COURSE OUTCOMES (COs)

CO1	Implement various basic data structures and its operations.
CO2	Implement various sorting and searching algorithms.
CO3	Implement various tree operations.
CO4	Implement various graphs algorithms.
CO5	Develop simple applications using various data structures.
CO6	Develop algorithms using various searching and sorting techniques.

MAPPING BETWEEN COURSE OUTCOMES & PROGRAM OUTCOMES
(3/2/1 INDICATES STRENGTH OF CORRELATION) 3- High, 2- Medium, 1-Low

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	2	2	3	2		2		3	1	1	2	2	3	
CO2	3	2	2	3	3		2	2		1	1	2	2	3	
CO3				3	3				2				2	3	
CO4	3				2							1	2	3	
CO5		2	2	3	3		2	3		1	1	2	2	3	
CO6	3			3	3		2	2	3	1	1	2	2	3	
Category	Professional Core (PC)														
Approval	37th Meeting of Academic Council, May 2015														

LIST OF EXPERIMENTS:

1. Simple Cprograms -Control Structures -Functions - Aggregate data types-File handling
2. Implementation of-Lists, Stacks, Queues (Using Arrays, linked lists)-Trees - Searching andSorting algorithms

[BCS3L1]-[DATA STRUCTURES LABORATORY]

LIST OF EXPERIMENTS

Sl.No	Title
1.	Simple C programs ➢ Control Structures ➢ Functions ➢ Aggregate data types ➢ File handling
2.	Implementation of ➢ Lists, Stacks, Queues (Using Arrays, linked lists) ➢ Trees ➢ Searching and Sorting algorithms

CONTENT

Ex No :1 **CONVERT THE CELSIUS VALUE TO FAHRENHEIT**

AIM:

Write a Program in C to convert the Celsius value to Fahrenheit using Functions

ALGORITHM:

STEP 1: Start the Program

STEP 2: Declare the Input and Output Variables

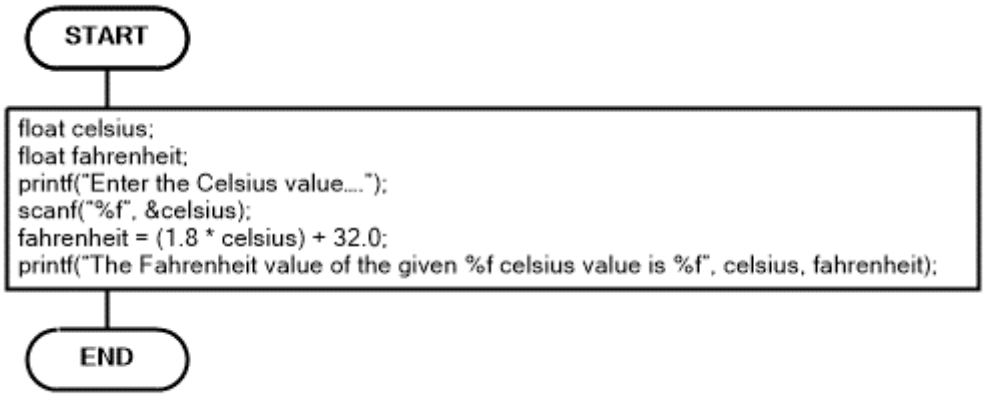
STEP 3: Enter the Initial Celsius Value

STEP 4: Calculate the Fahrenheit value by using the formula

Fahrenheit = (1.8 * Celsius) + 32.0;

STEP 5: Print the Output Fahrenheit value

STEP 6: Stop



SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>

void main()
{
        clrscr();
        float celsius;
        float fahrenheit;
        printf("Enter the Celsius value: ");
        scanf("%f", &celsius);
        fahrenheit = (1.8 * celsius) + 32.0;
        printf("The Fahrenheit value of the given %f celsius value is %f", celsius, fahrenheit);
```

```
    getch();
}
```

OUTPUT:

```
Enter the temperature in Celsius: 40
Temperature in Fahrenheit: 104.00 F
```

RESULT:

Thus the C program to convert the Celsius value to Fahrenheit using Functions was written,executed and the output was verified successfully.

Ex No :2 **FIND THE FACTORIAL OF A GIVEN NUMBER USING RECURSIVE FUNCTION**

AIM:

Write a Program in C to find the Factorial of a given number using Recursive function.

ALGORITHM:

STEP 1: Start the Program

STEP 2: Declare and Initialize the input variables

STEP 3: Enter the number

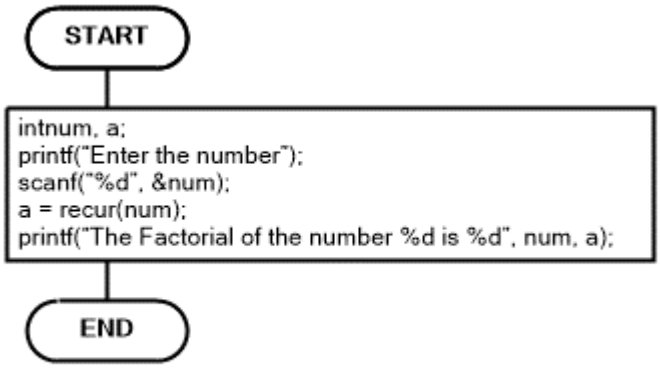
STEP 4: Call the Recursive Function passing the number to the recursive function as an Argument.

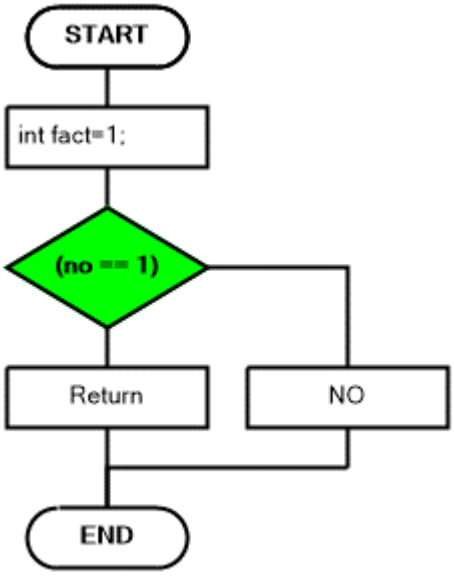
STEP 5: If the entered number is equal to one, then return one to Main function

STEP 6: If the number is less greater than one then call Recursive

STEP 7: Print the Factorial value of the number

STEP 8: Stop





SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
int recur(int b);
void main()
{
```

```
        clrscr();
        int     num, a;
        printf("Enter the Number");
        scanf("%d", &num);
        a = recur(num);
        printf("The Factorial of the number %d is %d", num, a);
        getch();
}
int recur(int no)
{
        int     fact;
        if(no == 1)
                return(1);
        else
                fact = no * recur(no - 1);
                return(fact);
}
```

OUTPUT:

Enter a Positive integer :6

Factorial of 6 =720

RESULT :

Thus the C program to find the Factorial of a given number using Recursive function was written,executed and the output was verified successfully.

Ex No :3

EAD AND WRITE THE CONTENTS OF A FILE

AIM:

Write a Program in C to Read and Write the Contents of a File.

ALGORITHM:

STEP 1: Start the Program

STEP 2: Initialize the File Pointer

STEP 3: Open the File in the Write Mode Using File Pointer

STEP 4: Enter the Data

STEP 5: Store the input data in the file using the putc() statement

STEP 6: Close the File

STEP 7: Open the File in the Read Mode using the File Pointer

STEP 8: Print the data in the file

SOURCE CODE :

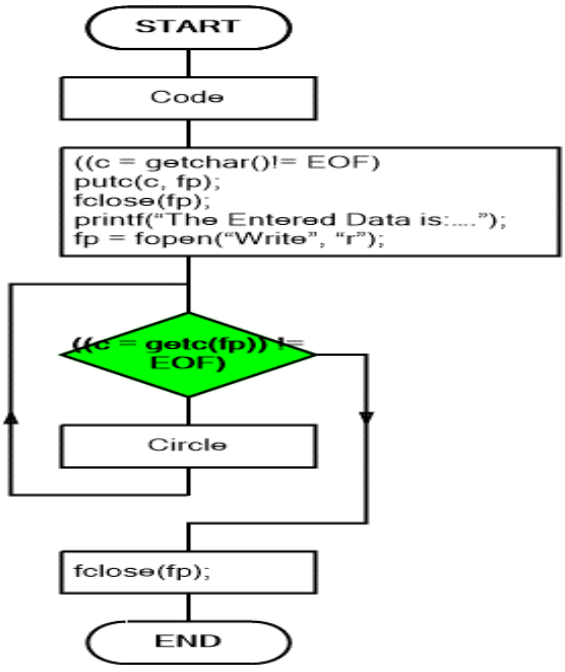
```
#include <stdio.h>
#include <stdlib.h>
  struct person
{
    int id;
    char fname[20];
    char lname[20];
};

int main ()
{
    FILE *infile;
    struct person input;

    // Open person.dat for reading
    infile = fopen ("person.dat", "r");
    if (infile == NULL)
    {
        fprintf(stderr, "\nError opening file\n");
        exit (1);
    }
```

```
    // read file contents till end of file
    while(fread(&input, sizeof(struct person), 1, infile))
       printf ("id = %d name = %s %s\n", input.id,
       input.fname, input.lname);

    // close file
    fclose (infile);

    return 0;
}
```

OUTPUT:

```
id = 1   name = rohan sharma
id = 2   name = mahendra dhoni
```

RESULT:



Result: Thus the C program to Read and Write the Contents of a File was written,executed and the output was verified successfully.

Ex No :4

STANDARD I/O LIBRARY FUNCTION WITH ARGUMENTS AND RETURN VALUE

AIM :

To write a Program in C Using Standard I/O Library Function with Arguments and Return Value.

ALGORITHM:

STEP 1: Start the Program

STEP 2: Declare and Initialize the Variables

STEP 3: Enter the two input numbers

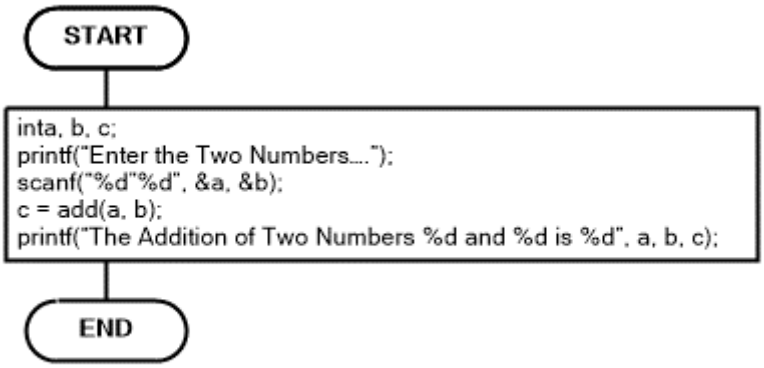
STEP 4: Call the function with two arguments passed to it

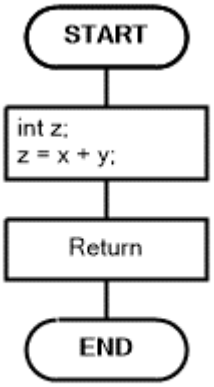
STEP 5: Add the two numbers in the calling function

STEP 6: Return the addition value to the called function from the calling function

STEP 7: Print the addition value in the main function

STEP 8: Stop






```
#include <stdio.h>
#include <conio.h>
int add(int x, int y);
void main()
{
        clrscr();
        int     a, b, c;
```

```
        printf("Enter the Two Numbers:");
        scanf("%d %d", &a, &b);
        c = add(a, b);
        printf("The Addition of Two Numbers %d and %d is %d", a, b, c);
        getch();
}

int add(int x, int y)
{
        int     z;
        z = x + y;
        return(z);
}
```

OUTPUT :

id = 1 name = rohan sharma

id = 2 name = mahendra dhoni

RESULT:

Thus the C program using Standard I/O Library Function with Arguments and Return Value.was written,executed and the output was verified successfully.

Ex No :5

PROGRAM TO FIND THE SIZE OF DATA TYPES

AIM:

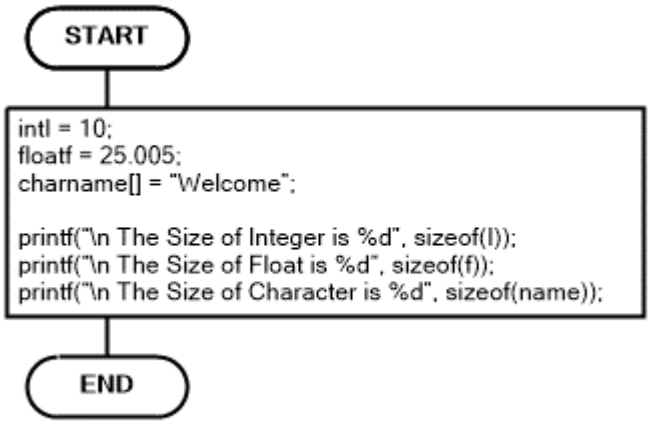
Write a Program in C to find the Size of Data Types.

ALGORITHM:

STEP 1: Start the Program

STEP 2: Initialize the Data Types

STEP 3: Print the size of the data types using the statement sizeof()



STEP 4: Stop

SOURCE CODE

```
#include <stdio.h>
#include <conio.h>
void main()
{
        Clrscr();
        int     i = 10;
        float   f = 25.005;
        char    name[] = "Welcome";
        printf("\n The Size of Integer is %d", sizeof(i));
        printf("\n The Size of Float is %d", sizeof(f));
        printf("\n The Size of Character is %d", sizeof(name));
        getch();
}
```

OUTPUT

```
Size of int: 4 bytes
Size of float: 4 bytes
Size of char: 1 byte
```

RESULT:

Thus the C program to find the Size of Data Types was written,executed and the output was verified successfully.

Ex No :6

IMPLEMENTATION OF STACKS USING LINKED LISTS

AIM:

To write a Program in C for Implementation of Stacks Using Linked Lists

ALGORITHM (For Push):

STEP 1: Start

STEP 2: new <-CreateNode (Node)

STEP 3: new Data<-Item

STEP 4: new Link<-Top

STEP 5: Top<-New

STEP 6: Stack_Head.Link<-Top

STEP 7: Stop

ALGORITHM (For Pop):

STEP 1: Start

STEP 2: If Top=NULL then print “Stack Empty or Underflow” Exit

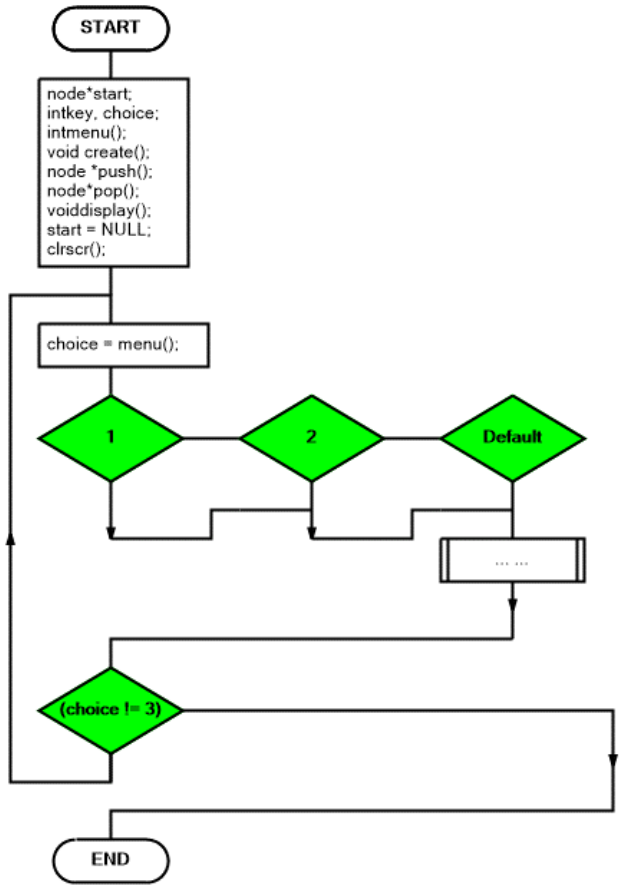
STEP 3: Else

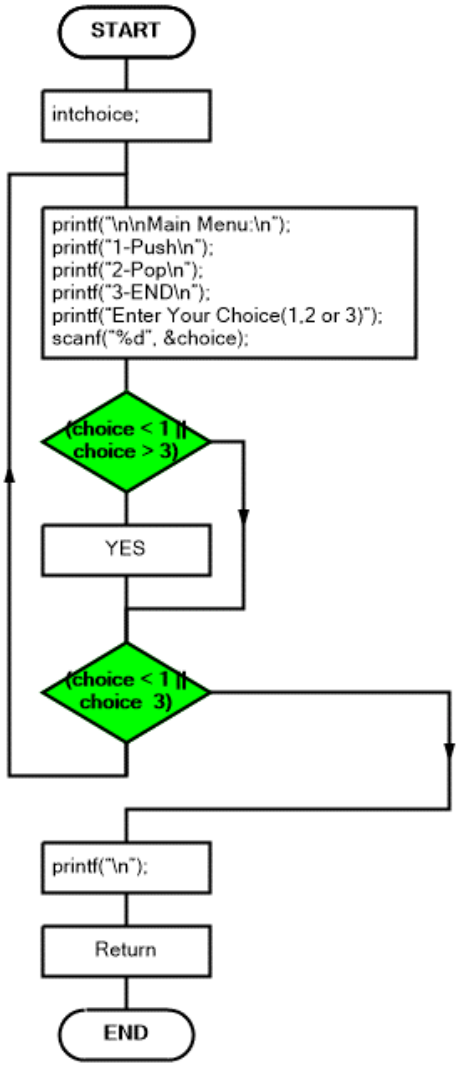
Ptr <- Top.Link

Item <- Top.Data

Stack <- Head.Link <- Ptr

Top <- Ptr

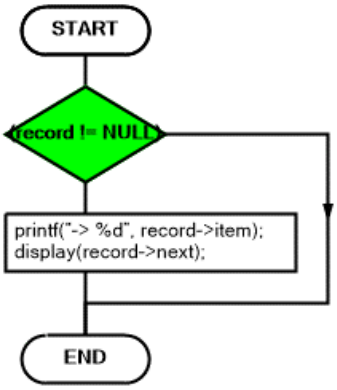


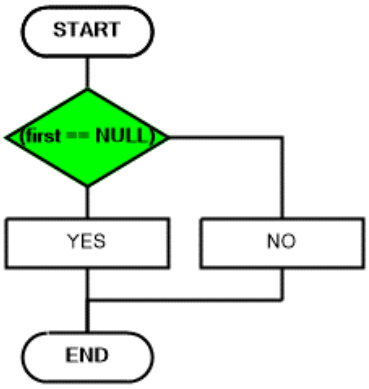


STEP 4: Stop

SOURCE CODE :

#include<stdio.h>





```
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
}*top = NULL; // Initially the list is empty
void push(int);
void pop();
void display();
int main()
{
int choice, value;
printf("\nIMPLEMENTING STACKS USING LINKED LISTS\n");
while(1){
printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("\nEnter the value to insert: ");
scanf("%d", &value);
push(value);
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
break;
default: printf("\nInvalid Choice\n");
}}}
void push(int value)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value; // get value for the node
if(top == NULL)
newNode->next = NULL;
else
newNode->next = top; // Make the node as TOP
top = newNode;
printf("Node is Inserted\n\n");
}
void pop()
{
if(top == NULL)
printf("\nEMPTY STACK\n");
else{
struct Node *temp = top;
```

```
printf("\nPopped Element : %d", temp->data);
printf("\n");
top = temp->next; // After popping, make the next node as TOP
free(temp);
}}
void display()
{
// Print the stack
if(top == NULL)
printf("\nEMPTY STACK\n");
else
{
printf("The stack is \n");
struct Node *temp = top;
while(temp->next != NULL){
printf("%d--->",temp->data);
temp = temp -> next;
}
printf("%d--->NULL\n\n",temp->data);
}}
```

OUTPUT:

```
IMPLEMENTING STACKS USING LINKED LISTS
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 15
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 30
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
The stack is
30--->15--->NULL
```

```
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2

Popped Element : 30
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2

Popped Element : 15
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2

STACK UNDERFLOW
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 4

Process returned 0 (0x0)   execution time : 22.311 s
Press any key to continue.
```

RESULT:

Thus the C program for Implementation of Stacks Using Linked Lists was written,executed and the output was verified successfully.

Ex No :7

IMPLEMENT STACKS USING ARRAYS

AIM:

Write a Program in C to Implement Stacks using Arrays

ALGORITHM:

INSERTION:

PUSH(item)

STEP 1. if(item=max of stack)
Print “Overflow”
Return

STEP 2. Top = Top + 1

STEP 3. Stack[Top]=item

STEP 4. Return

DELETION

POP(item)

STEP 1. If(top = -1)
Print “Underflow”
Return

STEP 2. Item = Stack[Top]

STEP 3. Top = Top – 1

STEP 4. Return1

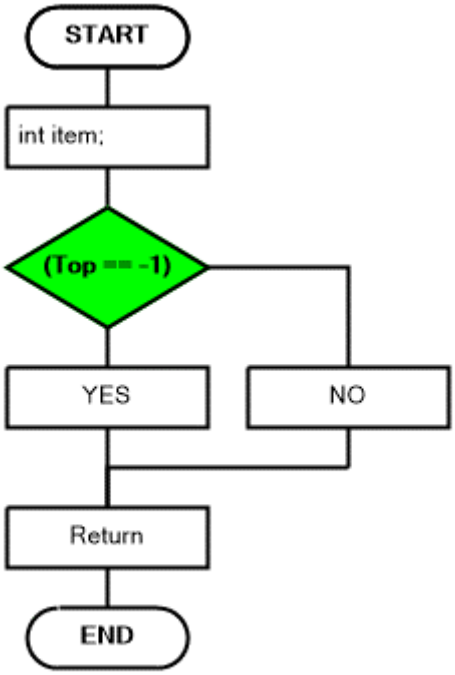
DISPLAY

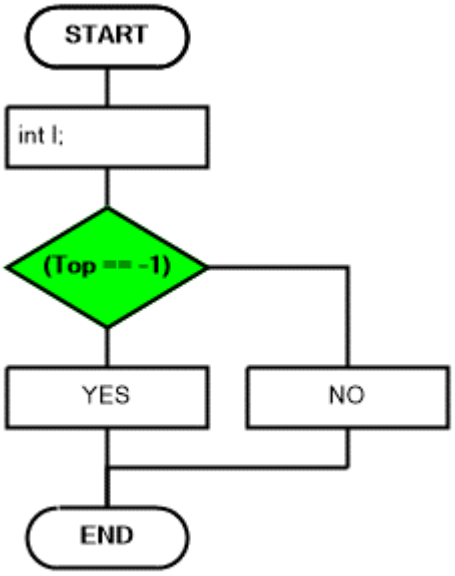
STEP 1. If Top = - 1
Print “Underflow”

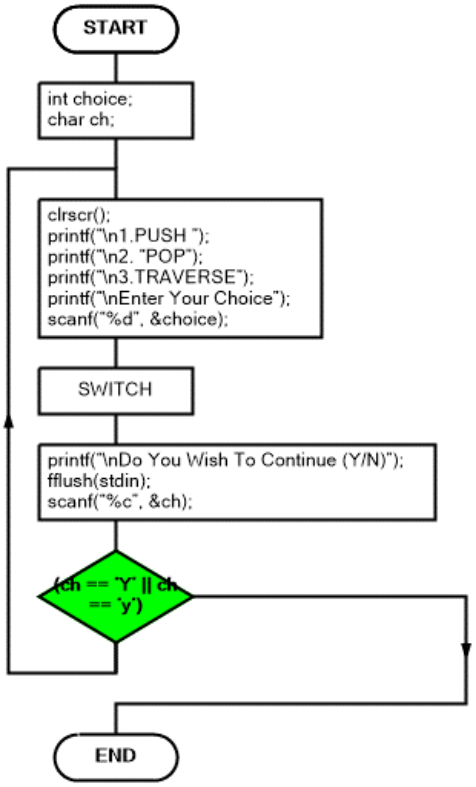
STEP 2. Repeat Step 3 for I = top to I >= 0

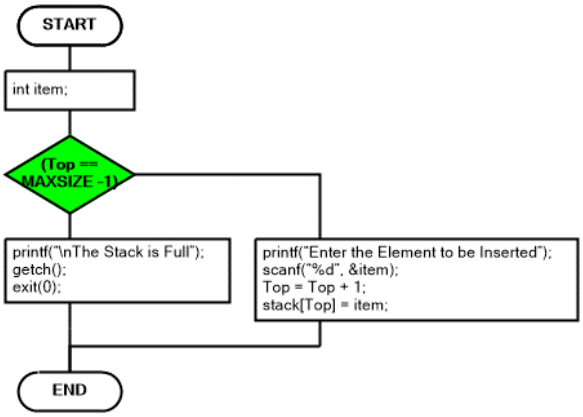
STEP 3. Print Stack[I]

STEP 4. Return


START
int item;
(Top == -1)
YES
NO
Return
END


START
int I;
(Top == -1)
YES
NO
END


START
int choice;
char ch;
clrscr();
printf("\n1.PUSH ");
printf("\n2. "POP");
printf("\n3.TRAVERSE");
printf("\nEnter Your Choice");
scanf("%d", &choice);
SWITCH
printf("\nDo You Wish To Continue (Y/N)");
fflush(stdin);
scanf("%c", &ch);
(ch == 'Y' || ch == 'y')
END


START
int item;
(Top == MAXSIZE -1)
printf("\nThe Stack is Full");
getch();
exit(0);
printf("Enter the Element to be Inserted");
scanf("%d", &item);
Top = Top + 1;
stack[Top] = item;
END

SOURCE CODE :

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
   //clrscr();
   top=-1;
   printf("\n Enter the size of STACK[MAX=100]:");
   scanf("%d",&n);
   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t--------------------------------");
   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
   do
   {
      printf("\n Enter the Choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
            break;
         }
         case 3:
         {
            display();
            break;
         }
         case 4:
         {
            printf("\n\t EXIT POINT ");
            break;
         }
         default:
         {
            printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
         }

      }
```

```
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

OUTPUT:

Enter the size of STACK[MAX=100]:10

```
        STACK OPERATIONS USING ARRAY
       ------------------------------
        1.PUSH
        2.POP
        3.DISPLAY
        4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:12

 Enter the Choice:1
 Enter a value to be pushed:24

 Enter the Choice:1
 Enter a value to be pushed:98

 Enter the Choice:3

 The elements in STACK

98
24
12
 Press Next Choice
 Enter the Choice:2

        The popped elements is 98
 Enter the Choice:3

 The elements in STACK

24
12
 Press Next Choice
 Enter the Choice:4

        EXIT POINT
```

RESULT:

Thus the C program for Implementation of Stacks Using Arrays was written,executed and the output was verified successfully.

Ex No :8

IMPLEMENT QUEUES USING ARRAYS

AIM:

Write a Program in C to implement Queues using Arrays.

ALGORITHM:

STEP 1: start the program

STEP 2: initialize the array variable a() for storing elements and declare the required variable

STEP 3: Define a function to insert , delete display the data items for data

STEP 4: For insert() function

*get the new elements

*create the rear is greater than array size display queue is"overflow".

Else

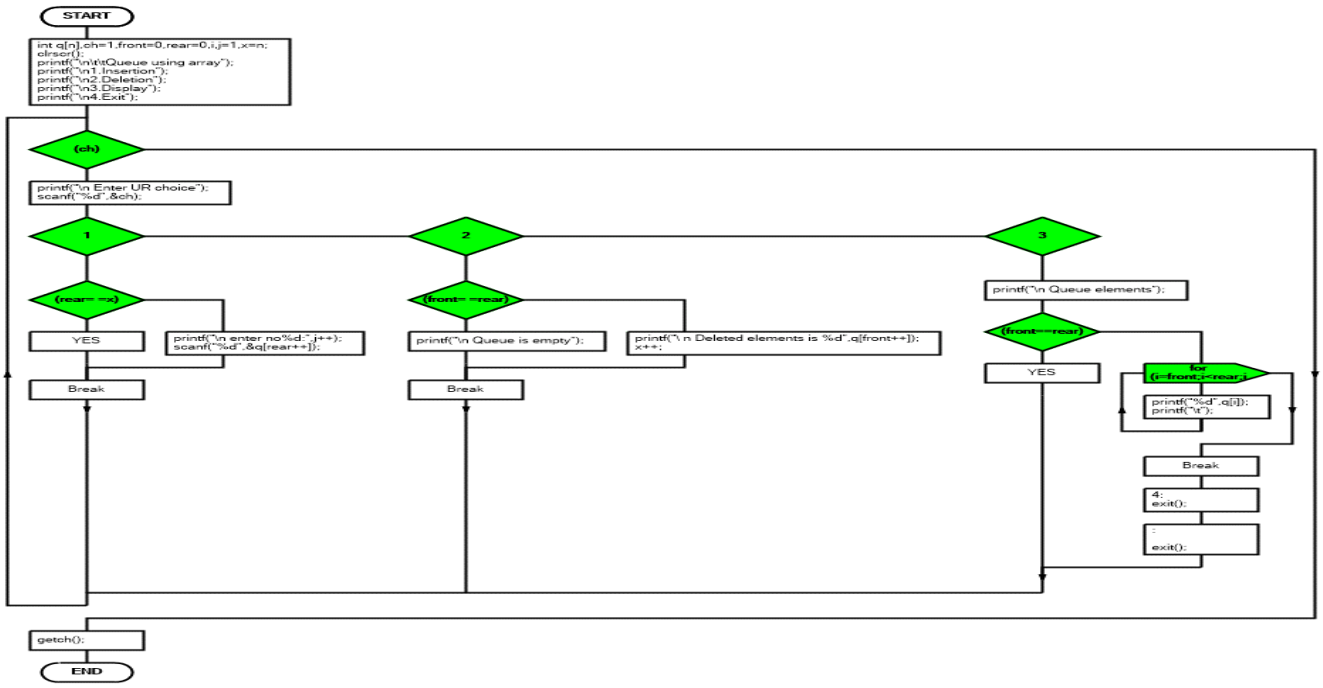
Insert the new element & increment the top pointer

STEP 5: For delete() function If the queue is empty than display "queue is over flow"

Else

Decrement the top pointer

STEP 6: Stop



SOURCE CODE:

```
#include<stdio.h>
#define n 5
int main()
{
   int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
   printf("Queue using Array");
   printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
   while(ch)
   {
      printf("\nEnter the Choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         if(rear==x)
            printf("\n Queue is Full");
         else
         {
            printf("\n Enter no %d:",j++);
            scanf("%d",&queue[rear++]);
         }
         break;
      case 2:
         if(front==rear)
         {
            printf("\n Queue is empty");
         }
```

```
        else
        {
           printf("\n Deleted Element is %d",queue[front++]);
           x++;
        }
        break;
     case 3:
        printf("\nQueue Elements are:\n ");
        if(front==rear)
           printf("\n Queue is Empty");
        else
        {
           for(i=front; i<rear; i++)
           {
              printf("%d",queue[i]);
              printf("\n");
           }
           break;
        case 4:
           exit(0);
        default:
           printf("Wrong Choice: please see the options");
        }
     }}
  return 0;}
```

OUTPUT:

```
Queue using Array
1.Insertion
2.Deletion
3.Display
4.Exit
Enter the Choice:1

 Enter no 1:10

Enter the Choice:1

 Enter no 2:54

Enter the Choice:1

 Enter no 3:98

Enter the Choice:1

 Enter no 4:234
```

```
Enter the Choice:3

 Queue Elements are:
 10
54
98
234

Enter the Choice:2

 Deleted Element is 10
Enter the Choice:3

 Queue Elements are:
 54
98
234

Enter the Choice:4
```

RESULT:

Thus the C program for Implementation of Queue Using Arrays was written,executed and the output was verified successfully.

Ex No :9

BINARY-TREE ALGORITHM

AIM:

To write a Program in C to Implement Binary-Tree Algorithm

ALGORITHM:

STEP 1: Start the Program

STEP 2: Declare a Template for Key and Elements

STEP 3: Insert the Key-Element Pair into the Tree, overwriting Existing pair, if any, with same key

STEP 4: Find place to Insert. While P != NULL

STEP 5: Examine Element pointed to by Pair. Move P to a Child

STEP 6: If Pair First Element is Less Than Element of first P, assign P as Left Child.

STEP 7: If Pair First Element is Greater than Element of first P, assign P as Right Child

STEP 8: Else, Replace Old Value

STEP 9: Retrieve a Node for the Pair and assign to Pair Pointer Element

STEP 10: If Root != NULL, the tree is not Empty.

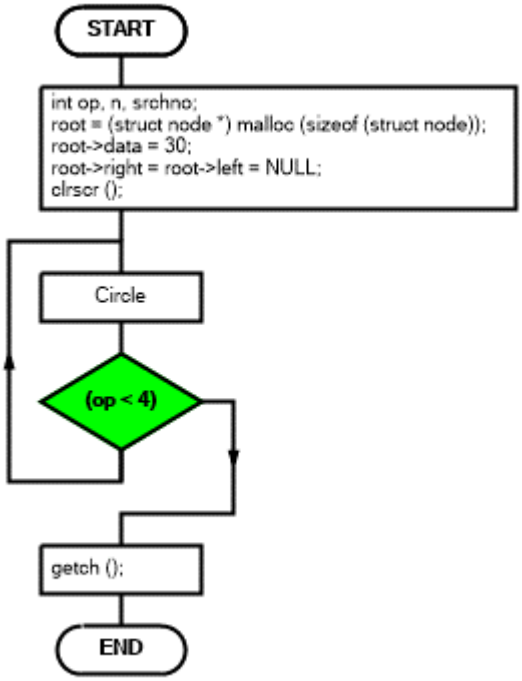
STEP 11: Now, Handle Input Operation – Insert, Delete, ListOrder Accordingly

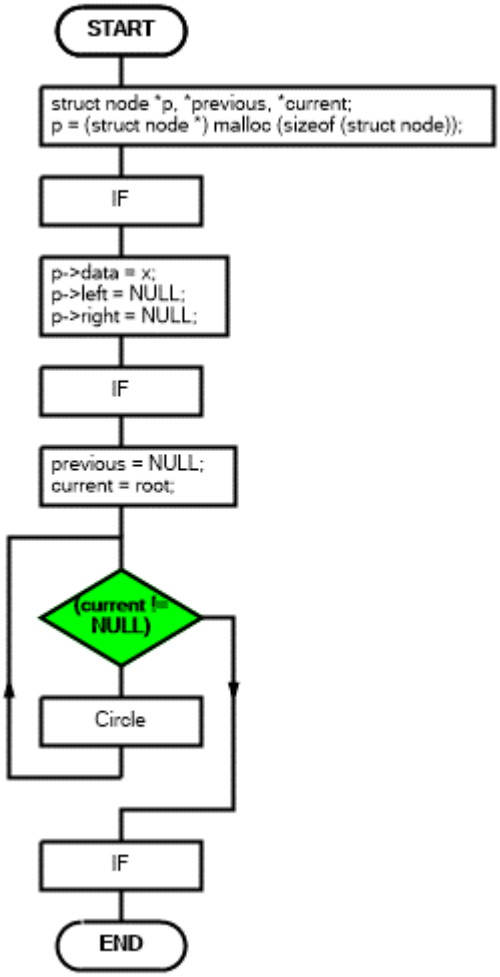
STEP 12: If Pair First Element is Less Than Pair Pointer First Element; LeftChildNode = NewNode

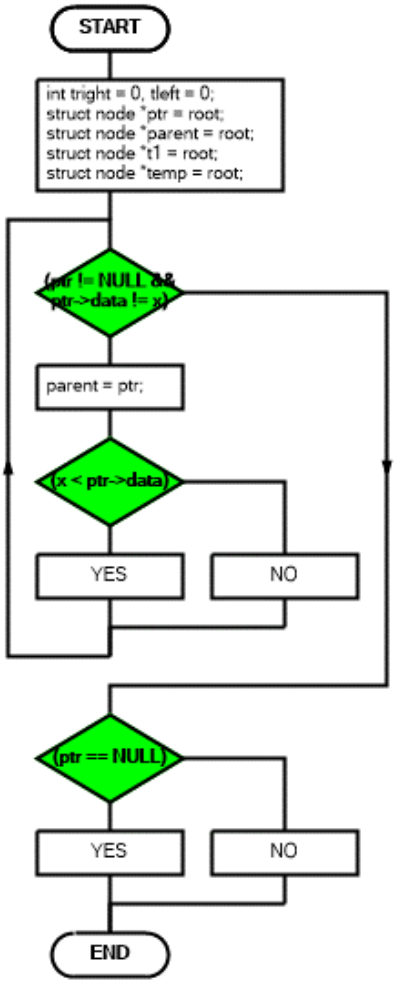
STEP 13: Else, Pair Pointer First Element RightChildNode = NewNode. Else, Root = NewNode

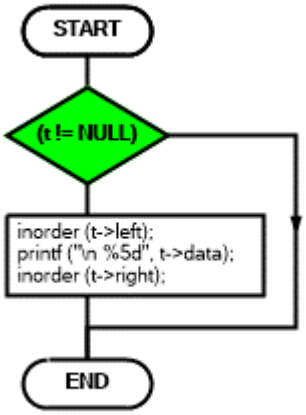
STEP 14: Insert Operation Into Tree Successful.

STEP 15: Stop


START
int op, n, srchno;
root = (struct node *) malloc (sizeof (struct node));
root->data = 30;
root->right = root->left = NULL;
clrscr ();
Circle
(op < 4)
getch ();
END


START
struct node *p, *previous, *current;
p = (struct node *) malloc (sizeof (struct node));
IF
p->data = x;
p->left = NULL;
p->right = NULL;
IF
previous = NULL;
current = root;
(current != NULL)
Circle
IF
END


START
int tright = 0, tleft = 0;
struct node *ptr = root;
struct node *parent = root;
struct node *t1 = root;
struct node *temp = root;
(ptr != NULL && ptr->data != x)
parent = ptr;
(x < ptr->data)
YES
NO
(ptr == NULL)
YES
NO
END


START
(t != NULL)
inorder (t->left);
printf ("\n %5d", t->data);
inorder (t->right);
END

```
# include <stdio.h>
# include <malloc.h>

struct node
{
        int info;
        struct node *lchild;
        struct node *rchild;
}*root;



void find(int item,struct node **par,struct node **loc)
{
        struct node *ptr,*ptrsave;

        if(root==NULL)  /*tree empty*/
        {
                *loc=NULL;
                *par=NULL;
                return;
        }
        if(item==root->info) /*item is at root*/
        {
                *loc=root;
                *par=NULL;
                return;
        }
        /*Initialize ptr and ptrsave*/
        if(item<root->info)
                ptr=root->lchild;
        else
                ptr=root->rchild;
        ptrsave=root;

        while(ptr!=NULL)
        {
                if(item==ptr->info)
                {       *loc=ptr;
                        *par=ptrsave;
                        return;
                }
                ptrsave=ptr;
                if(item<ptr->info)
                        ptr=ptr->lchild;
                else
                        ptr=ptr->rchild;
         }/*End of while */
         *loc=NULL;   /*item not found*/
```

```
        *par=ptrsave;
}/*End of find()*/

void insert(int item)
{       struct node *tmp,*parent,*location;
        find(item,&parent,&location);
        if(location!=NULL)
        {
                printf("Item already present");
                return;
        }

        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=item;
        tmp->lchild=NULL;
        tmp->rchild=NULL;

        if(parent==NULL)
                root=tmp;
        else
                if(item<parent->info)
                        parent->lchild=tmp;
                else
                        parent->rchild=tmp;
}/*End of insert()*/


void case_a(struct node *par,struct node *loc )
{
        if(par==NULL) /*item to be deleted is root node*/
                root=NULL;
        else
                if(loc==par->lchild)
                        par->lchild=NULL;
                else
                        par->rchild=NULL;
}/*End of case_a()*/

void case_b(struct node *par,struct node *loc)
{
        struct node *child;

        /*Initialize child*/
        if(loc->lchild!=NULL) /*item to be deleted has lchild */
                child=loc->lchild;
        else            /*item to be deleted has rchild */
                child=loc->rchild;

        if(par==NULL )   /*Item to be deleted is root node*/
```

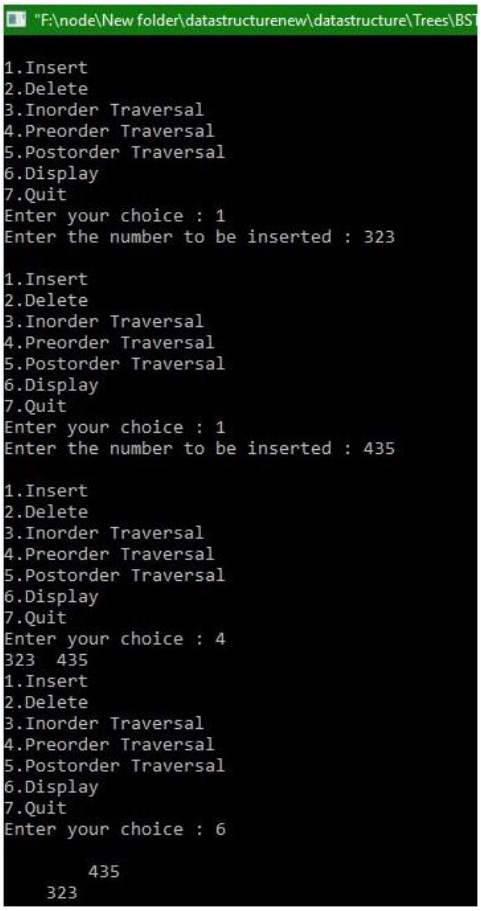
```
                root=child;
        else
                if( loc==par->lchild)   /*item is lchild of its parent*/
                        par->lchild=child;
                else                    /*item is rchild of its parent*/
                        par->rchild=child;
}/*End of case_b()*/

void case_c(struct node *par,struct node *loc)
{
        struct node *ptr,*ptrsave,*suc,*parsuc;

        /*Find inorder successor and its parent*/
        ptrsave=loc;
        ptr=loc->rchild;
        while(ptr->lchild!=NULL)
        {
                ptrsave=ptr;
                ptr=ptr->lchild;
        }
        suc=ptr;
        parsuc=ptrsave;

        if(suc->lchild==NULL && suc->rchild==NULL)
                case_a(parsuc,suc);
        else
                case_b(parsuc,suc);

        if(par==NULL) /*if item to be deleted is root node */
                root=suc;
        else
                if(loc==par->lchild)
                        par->lchild=suc;
                else
                        par->rchild=suc;

        suc->lchild=loc->lchild;
        suc->rchild=loc->rchild;
}/*End of case_c()*/
int del(int item)
{
        struct node *parent,*location;
        if(root==NULL)
        {
                printf("Tree empty");
                return 0;
        }

        find(item,&parent,&location);
```

```
        if(location==NULL)
        {
                printf("Item not present in tree");
                return 0;
        }

        if(location->lchild==NULL && location->rchild==NULL)
                case_a(parent,location);
        if(location->lchild!=NULL && location->rchild==NULL)
                case_b(parent,location);
        if(location->lchild==NULL && location->rchild!=NULL)
                case_b(parent,location);
        if(location->lchild!=NULL && location->rchild!=NULL)
                case_c(parent,location);
        free(location);
}/*End of del()*/

int preorder(struct node *ptr)
{
        if(root==NULL)
        {
                printf("Tree is empty");
                return 0;
        }
        if(ptr!=NULL)
        {
                printf("%d  ",ptr->info);
                preorder(ptr->lchild);
                preorder(ptr->rchild);
        }
}/*End of preorder()*/

void inorder(struct node *ptr)
{
        if(root==NULL)
        {
                printf("Tree is empty");
                return;
        }
        if(ptr!=NULL)
        {
                inorder(ptr->lchild);
                printf("%d  ",ptr->info);
                inorder(ptr->rchild);
        }
}/*End of inorder()*/

void postorder(struct node *ptr)
{
```

```
        if(root==NULL)
        {
                printf("Tree is empty");
                return;
        }
        if(ptr!=NULL)
        {
                postorder(ptr->lchild);
                postorder(ptr->rchild);
                printf("%d  ",ptr->info);
        }
}/*End of postorder()*/

void display(struct node *ptr,int level)
{
        int i;
        if ( ptr!=NULL )
        {
                display(ptr->rchild, level+1);
                printf("\n");
                for (i = 0; i < level; i++)
                        printf("    ");
                printf("%d", ptr->info);
                display(ptr->lchild, level+1);
        }/*End of if*/
}/*End of display()*/
main()
{
        int choice,num;
        root=NULL;
        while(1)
        {
                printf("\n");
                printf("1.Insert\n");
                printf("2.Delete\n");
                printf("3.Inorder Traversal\n");
                printf("4.Preorder Traversal\n");
                printf("5.Postorder Traversal\n");
                printf("6.Display\n");
                printf("7.Quit\n");
                printf("Enter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                 case 1:
                        printf("Enter the number to be inserted : ");
                        scanf("%d",&num);
                        insert(num);
```

```
                    break;
                case 2:
                    printf("Enter the number to be deleted : ");
                    scanf("%d",&num);
                    del(num);
                    break;
                case 3:
                    inorder(root);
                    break;
                case 4:
                    preorder(root);
                    break;
                case 5:
                    postorder(root);
                    break;
                case 6:
                    display(root,1);
                    break;
                case 7:
        break;
                default:
                    printf("Wrong choice\n");
                }/*End of switch */
        }/*End of while */
}/*End of main()*/
```

OUTPUT:



RESULT:

Thus the C program for Implementation of Implement Binary-Tree Algorithm was written,executed and the output was verified successful

Ex No :10

QSORT Function To Implement Linear Sorting

AIM:

Write a Program in C using the QSORT function to Implement Linear Sorting on the given Array Object.

ALGORITHM:

STEP 1. low=1, high=h, key a[(l+h)/2]

STEP 2. Repeat through step 7 while(low <= high)

STEP 3. Repeat Step 4 while (a[low] < key)

STEP 4. low = low + 1

STEP 5. Repeat step 6 while(a[high] > key)

STEP 6. high = high – 1

STEP 7. if(low <= high)

a) temp = a[low]

b) a[low] = a[high]

c) a[high] = temp

d) low = low + 1

e) high = high + 1

STEP 8. if (l < high) quicksort(a, l , high)

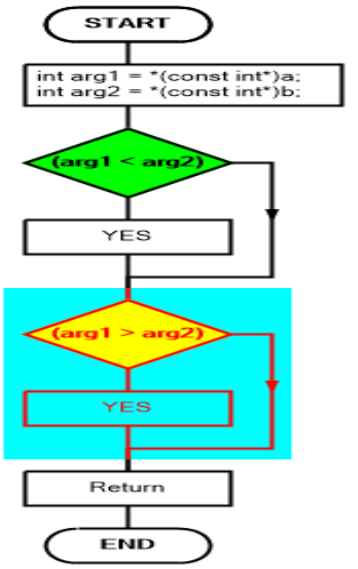
STEP 9. if (h > low) quicksort(a, low, h)

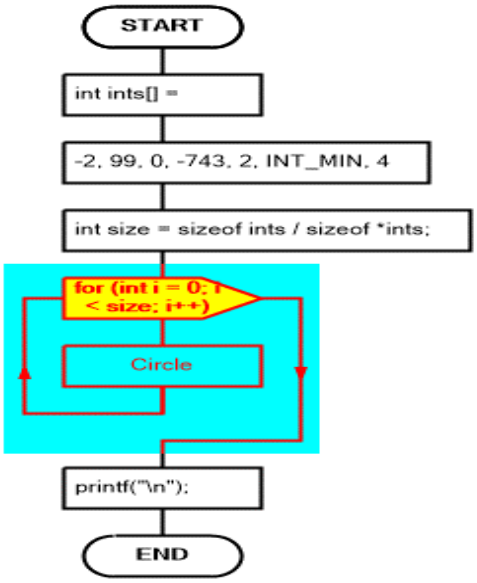
SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
static int comparator(const void* str1, const void* str2) {
  if(strcmp(*(const char**)str1, *(const char**)str2) >= 0)
    return 1;
  else return 0;
}
int main() {
  const char* arr[] = {"Rishabh", "Jyoti", "Palak", "Akash"};
  int n = sizeof(arr) / sizeof(arr[0]);
  printf("\nGiven array of names: \t");
  for (int i = 0; i < n; i++) printf("%s \t", arr[i]);
    qsort(arr, n, sizeof(const char*), comparator);
  printf("\nSorted array of names: \t");
  for (int i = 0; i < n; i++)
    printf("%s \t", arr[i]);
  return 0;
}
```

OUTPUT:

Given array of names: Rishabh Jyoti Palak Akash
Sorted array of names: Akash Jyoti Palak Rishabh





RESULT:

Thus the C program for the QSORT function to Implement Linear Sorting was written,executed and the output was verified successfully