DATA AND FILE STRUCTURE

LAB

17BIT028

STACK

```
#include <stdio.h>
#include <stdlib.h>
int top=-1;

void push(int s[],int n,int x){
    if(top<n-1){
        s[++top]=x;
    }
    else
        printf("srtack is full\n");
}

int pop(int s[],int n){
    if(top==-1) {
            printf("stack is empty\n");
            return 0;
    }
    else return s[top--];
}

void display(int s[]){
    int n;
    n=top;
    int i;
for(i=n;i>=0;i--){
printf("\n|  %4d      |",s[i]);
```

```
    }
    printf("\n----------\n");
}
int main(){
int n,x;
printf("enter size of stack : ");
scanf("%d",&n);

int s[n];
int choise;
printf("\n\n ----------------------------------------------------\n");
printf("choise what do u whant\n");
printf("1. push \n2. pop \n3. disply \n4. exit\n");
printf("-----------------------------------------------------\n");

pick :

printf("enter your choise : ");
scanf("%d",&choise);

switch(choise){
case 1:
    printf("enter number you want to add in stack :");
    scanf("%d",&x);
    push(s,n,x);

break;
```

```
case 2:
    printf("you poped : %d\n", pop(s,n));
break;

case 3:
    display(s);
break;

case 4:
    exit(0);
}

goto pick;
return 0;
}
```

```
enter size of stack : 4


 ----------------------------------------------------
choise what do u whant
1. push
2. pop
3. disply
4. exit
----------------------------------------------------
enter your choise : 1
enter number you want to add in stack :12
enter your choise : 1
enter number you want to add in stack :15
enter your choise : 1
enter number you want to add in stack :17
enter your choise : 1
enter number you want to add in stack :11
enter your choise : 1
enter number you want to add in stack :13
```

```
srtack is full
enter your choise : 3

|   11    |
|   17    |
|   15    |
|   12    |
-------------
enter your choise : 2
you poped : 11
enter your choise : 3

|   17    |
|   15    |
|   12    |
-------------
enter your choise : 2
you poped : 17
enter your choise : 2
you poped : 15
enter your choise : 3

|   12    |
-------------
enter your choise : 2
you poped : 12
enter your choise : 3

-------------
enter your choise : 4

Process returned 0 (0x0)   execution time : 44.006 s
Press any key to continue.
```

INFIX TO POSTFIX

```
#include<stdio.h>
int top=-1,top2=-1;
int MAXSIZE=100;
char s[100];

int isOperand(char x){
        if((x>='a'&& x<='z')||(x>='A' && x<='Z')){
                return 1;
        }
        else
                return 0;
}

void push(char x){

        s[++top]=x;

}

char pop(){

        return s[top--];

}

int priority(char x){
```

```
        if(x=='('){
                return 0;
        }
        else if(x=='+' || x=='-'){
                return 1;
        }
        else if(x=='*' || x=='/'){
                return 2;
        }
}

int main(){
    int p=0;
        char exp[MAXSIZE],*e,postfix[MAXSIZE];
        printf("Enter your infix Expression : ");
        scanf("%s",exp);
        e=exp;
        printf(" your Postfix Expression is : ");
        while(*e!='\0'){

                if(isOperand(*e)){
            postfix[p]=*e;
                        printf("%c",postfix[p++]);

                }

                else if(*e=='('){
```

```
                push(*e);
            }

            else if(*e==')'){
                while(s[top]!='('){
        postfix[p]=pop();
                    printf("%c",postfix[p++]);


                }
                top--;
            }

            else{
                while(priority(s[top])>=priority(*e)){
          postfix[p]=pop();
                        printf("%c",postfix[p++]);


                }
                push(*e);
            }
      e++;
      }

  while(top!=-1){
    postfix[p]=pop();
```

```
        printf("%c",postfix[p++]);
    }
    printf("\n");
    return 0;
}
```

Enter your infix Expression : a+b(c/d)%e*
your Postfix Expression is : abcd/e%+*

Process returned 0 (0x0)   execution time : 21.678 s
Press any key to continue.

****

Enter your infix Expression : (10+20)(12/3)*(7%2)*
*your Postfix Expression is : 1020+213/*72%**

Process returned 0 (0x0)   execution time : 53.655 s
Press any key to continue.

Infix To Postfix To Evaluation

```
#include<stdio.h>
int top=-1,top2=-1;
int MAXSIZE=100;
char s[100];
int d[100];
int isOperand(char x){
        if((x>='a'&& x<='z')||(x>='A' && x<='Z')){
                return 1;
        }
        else
                return 0;
}

int isnumber(int n){

   if(n>=0 && n<=9)return 1;
   else return 0;
}
void push_n(int x){

        d[++top2]=x;

}

int pop_n(){
```

```
        return d[top2--];

}
void push(char x){

        s[++top]=x;

}

char pop(){

        return s[top--];

}

int priority(char x){
        if(x=='('){
                return 0;
        }
        else if(x=='+' || x=='-'){
                return 1;
        }
        else if(x=='*' || x=='/'){
                return 2;
        }
}
```

```
int main(){
    int p=0;
        char exp[MAXSIZE],*e,postfix[MAXSIZE];
        printf("Enter your infix Expression : ");
        scanf("%s",exp);
        e=exp;
        printf("\npostfix Expression is : ");
        while(*e!='\0'){

            if(isOperand(*e)){
          postfix[p]=*e;
                printf("%c",postfix[p++]);

            }

            else if(*e=='('){
                push(*e);
            }

            else if(*e==')'){
                while(s[top]!='('){
          postfix[p]=pop();
                    printf("%c",postfix[p++]);

                }
                top--;
            }
```

```
                else{
                        while(priority(s[top])>=priority(*e)){
                postfix[p]=pop();
                                        printf("%c",postfix[p++]);

                        }
                        push(*e);
                }
        e++;
        }

while(top!=-1){
   postfix[p]=pop();
        printf("%c",postfix[p++]);

}
printf("\n\n");



int a,b,c;
int j=0;
while(postfix[j]!='\0'){


      if(isOperand(postfix[j])){
```

```
            printf("Enter the value of %c : ",postfix[j]);
            int p;
            scanf("%d",&p);
            push_n(p);


        }else if(postfix[j]==' '){}
        else{
            a=pop_n();
            b=pop_n();


                if(postfix[j]=='+') c=b+a;
                if(postfix[j]=='-') c=b-a;
                if(postfix[j]=='/') c=b/a;
                if(postfix[j]=='*') c=b*a;
                if(postfix[j]=='%') c=b%a;
                push_n(c);
        }
        j=j+1;

    }
    printf("\nEvaluation answer is : %d\n\n",pop_n());
    }
```

1]

Enter your infix Expression : a+b*(c/d)

postfix Expression is : abcd/*+

Enter the value of a : 10
Enter the value of b : 20
Enter the value of c : 6
Enter the value of d : 2

Evaluation answer is : 70

Process returned 0 (0x0)   execution time : 20.787 s
Press any key to continue.

****

2]

Enter your infix Expression : (a+b)*(c/d)*(e%f)

postfix Expression is : ab+cd/*ef%*

Enter the value of a : 10
Enter the value of b : 20
Enter the value of c : 6
Enter the value of d : 2

```
Enter the value of e : 7
Enter the value of f : 2

Evaluation answer is : 90


Process returned 0 (0x0)   execution time : 84.253 s
Press any key to continue.
```

TOWER OF HANOI

```
#include <stdio.h>
void towers(int, char, char, char);
int main()

{
   int num;
   printf("Enter the number of disks : ");
   scanf("%d", &num);
   towers(num, 'A', 'C', 'B');
   return 0;
}

void towers(int num, char from, char to, char aux)

{   if (num == 1)
      {
      printf("\n Move disk 1 from %c to %c", from, to);
      return;
   }
   towers(num - 1, from, aux, to);
   printf("\n Move disk %d from %c to %c", num, from, to);
   towers(num - 1, aux, to, from);
}
```

Enter the number of disks : 4

Move disk 1 from A to B

```
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Process returned 0 (0x0)   execution time : 1.964 s
Press any key to continue.
```

Betting_Game

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int n,b,r,*a;

void start(){
   srand(time(0));

   a = (int*)malloc(3*sizeof(int));
   *a=rand()%3;
   *(a+1)= rand()%3;
   *(a+2)= rand()%3;

   if(n>0){

      printf("you have currently $%d \n",n);
      printf("enter the bet amount $");
      scanf("%d",&b);
      if(b<=n){
      printf("find the place of queen[chose 1,2,3] : ");
      scanf("%d",&r);
      if(*a==(r-1) ){
         printf("congratulations you win!\n");
         n=n-b+(3*b);
         printf("now you have $%d\n",n);
         printf("_____\n");
```

```
            }
        else{
            printf("Better luck Next time\n");
            n=n-b;
            printf("now you have $%d \n",n);
            printf("_____\n");
        }
        }else{
            printf("you have only $%d and you Bet $%d \n",n,b);


        }

    }
    else{
            printf("you are not eligible to play game \n you ave no money\n");
        }
        free(a);

}
int main(){




printf("_____\n");
printf("      BETTING GAME\n");
printf("_____\n");
printf("  *****    *****    *****\n");
```

```
printf("  *   *    *   *    *   *\n");
printf("  * j *    * Q *    * K *\n");
printf("  *   *    *   *    *   *\n");
printf("  *****    *****    *****\n");
printf("_____________________________________\n");
printf("rules : \n");
printf("1) you have to chose between 1 to 3 \n");
printf("2) you are allow to play till you have $0 \n");
printf("3) if you will win then you will get 3*(BET AMOUNT)\n\n\n");
printf("how many amount you have $");
scanf("%d",&n);
int f;
f=n;

    printf("\n\n*****************************************\n");
    printf("               GAME START    \n");
    printf("*****************************************\n\n");
//printf("you want to play game(y/n):");
//char m;
//scanf("%c",&m);
while(n!=0){
start();
//printf("you want to play game again!(y/n):");
//scanf("%c",&m);
}
printf("\n\n\n\nyou come with $%d\n",f);
printf("now you have $%d\n\n\n",n);
```

```
printf("thank you for play!\n\n\n");
return 0;

}
```

```
_____
    BETTING GAME
_____
 *****   *****   *****
 *  *   *  *   *  *
 * j *   * Q *   * K *
 *  *   *  *   *  *
 *****   *****   *****

_____
rules :
1) you have to chose between 1 to 3
2) you are allow to play till you have $0
3) if you will win then you will get 3*(BET AMOUNT)


how many amount you have $15


****************************************
          GAME START
****************************************
```

you have currently $15

enter the bet amount $4

find the place of queen[chose 1,2,3] : 1

Better luck Next time

now you have $11

---

you have currently $11

enter the bet amount $7

find the place of queen[chose 1,2,3] : 3

congratulations you win!

now you have $25

---

you have currently $25

enter the bet amount $20

find the place of queen[chose 1,2,3] : 2

Better luck Next time

now you have $5

---

you have currently $5

enter the bet amount $3

find the place of queen[chose 1,2,3] : 2

Better luck Next time

now you have $2

---

you have currently $2

enter the bet amount $2

find the place of queen[chose 1,2,3] : 3

Better luck Next time

now you have $0

_____

you come with $15

now you have $0

thank you for play!

Process returned 0 (0x0)   execution time : 74.211 s

Press any key to continue.

SIMPLE QUEUE

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 5
int front=-1,rear=-1,q[MAXSIZE];
void insert(int a){
    if(rear>=MAXSIZE){
        printf("sorry!! \n queue is full \n");
    }
    else{
        if(front==-1) front=0;
        rear++;
        q[rear]=a;
        printf("element %d is successfully added\n",a);
        }
}

void delete(){
    if(front<0){
        printf("Error!! \n Underflow condition\n");
    }
    else if(front==rear){
        q[front]=0;
        front=-1;
        rear=-1;

        printf("element successfully delete\n");
```

```
    }
    else{
        q[front]=0;
        front++;
        printf("element successfully delete\n");
        }


}

void display(){
    int i=0;
    for(i=0;i<=MAXSIZE;i++){
        printf(" %d  |",q[i]);
    }
    printf("\n");
}

int main(){
    int choise=0,inse;
    printf(" 1) insert element \n 2) delete element \n 3) display queue  \n 4) exit\n ");
    while(choise != 4){
    printf("enter your choice : ");
    scanf("%d",&choise);
    switch(choise)
    {
    case 1:
```

```
        printf("enter a number which you want  to add in queue:");
        scanf("%d",&inse);
        insert(inse);
        break;
    case 2:
        printf("you chose delete option\n");
        delete();
        break;
    case 3:
        printf("you chose display option\n");
        display();
        break;
    case 4:
        printf("\nI hope u enjoy this program \n");
        break;
    }
}
return 0;


}
```

```
1) insert element
2) delete element
3) display queue
4) exit
enter your choice : 1
enter a number which you want  to add in queue:155
element 155 is successfully added
enter your choice : 1
```

```
enter a number which you want  to add in queue:34
element 34 is successfully added
enter your choice : 3
you chose display option
155  | 34  | 0  | 0  | 0  | 0  |
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 3
you chose display option
0  | 34  | 0  | 0  | 0  | 0  |
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 3
you chose display option
0  | 0  | 0  | 0  | 0  | 0  |
enter your choice : 1
enter a number which you want  to add in queue:27
element 27 is successfully added
enter your choice : 1
enter a number which you want  to add in queue:65
element 65 is successfully added
enter your choice : 1
enter a number which you want  to add in queue:87
element 87 is successfully added
enter your choice : 1
enter a number which you want  to add in queue:32
element 32 is successfully added
enter your choice : 3
you chose display option
27  | 65  | 87  | 32  | 0  | 0  |
enter your choice : 1
enter a number which you want  to add in queue:98
element 98 is successfully added
enter your choice : 1
enter a number which you want  to add in queue:185
element 185 is successfully added
enter your choice : 3
you chose display option
27  | 65  | 87  | 32  | 98  | 185  |
enter your choice : 1
enter a number which you want  to add in queue:188
sorry!!
queue is full
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 2
you chose delete option
```

```
element successfully delete
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 2
you chose delete option
element successfully delete
enter your choice : 2
you chose delete option
Error!!
Underflow condition
enter your choice : 4

I hope u enjoy this program

Process returned 0 (0x0)   execution time : 106.070 s
Press any key to continue.
```

CIRCULAR QUEUE

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 5
int front=-1,rear=-1,q[MAXSIZE];
void insert(int a){
   if(front==(rear+1)%MAXSIZE) printf("Error!!\nQueue is full!!!\n");
   else{
        if(front==-1) front=0;
        rear =(rear+1)%MAXSIZE;
        q[rear]=a;
        }
}

void delete(){
    if(front==rear){
        printf("Error!! \n Underflow condition\n Queue is empty\n");
    }
    else{
        q[front]=0;
        front=(front+1)%MAXSIZE;
        printf("element successfully delete\n");
        }
}

void display(){
    int i=0;
```

```
    for(i=0;i<MAXSIZE;i++){
        printf(" %d  |",q[i]);
    }
    printf("\n");
}

int main(){
    int choise=0,inse;
     printf(" 1) insert element \n 2) delete element \n 3) display queue  \n 4) exit\n");
    while(choise != 4){
    printf(" enter your choice :");
    scanf("%d",&choise);
    switch(choise)
    {
    case 1:
        printf("enter a number which you want  to add in queue:");
        scanf("%d",&inse);
        insert(inse);
        break;
    case 2:
        printf("you chose delete option\n");
        delete();
        break;
    case 3:
        printf("you chose display option\n");
        display();
        break;
```

```
    case 4:
        printf("\ni hope u enjoy this prg\n");
        break;
    }
}
return 0;

}
```

```
1) insert element
2) delete element
3) display queue
4) exit
enter your choice :1
enter a number which you want  to add in queue:13
enter your choice :1
enter a number which you want  to add in queue:18
enter your choice :1
enter a number which you want  to add in queue:37
enter your choice :1
enter a number which you want  to add in queue:27
enter your choice :3
you chose display option
13  | 18  | 37  | 27  | 0  |
enter your choice :1
enter a number which you want  to add in queue:15
enter your choice :3
you chose display option
13  | 18  | 37  | 27  | 15  |
enter your choice :2
you chose delete option
element successfully delete
enter your choice :3
you chose display option
0  | 18  | 37  | 27  | 15  |
enter your choice :1
enter a number which you want  to add in queue:16
enter your choice :3
you chose display option
16  | 18  | 37  | 27  | 15  |
```

```
enter your choice :2
you chose delete option
element successfully delete
enter your choice :3
you chose display option
16  | 0  | 37  | 27  | 15  |
enter your choice :2
you chose delete option
element successfully delete
enter your choice :2
you chose delete option
element successfully delete
enter your choice :3
you chose display option
16  | 0  | 0  | 0  | 15  |
enter your choice :2
you chose delete option
element successfully delete
enter your choice :2
you chose delete option
Error!!
Underflow condition
Queue is empty
enter your choice :4

i hope u enjoy this prg

Process returned 0 (0x0)   execution time : 112.151 s
Press any key to continue.
```

QUEUE USING STACK

```
#include <stdio.h>
#include <stdlib.h>
#define max 10
int top1=-1,top2=-1,s1[max],s2[max];
void insert(int p){
    if(top1==max)
        printf("queue is full");
    else{
        top1++;
        s1[top1] = p;
    }
}
void push2(int p){
if(top2==max)
        printf("queue is full");
    else{
        top2++;
        s2[top2] = p;
    }
}
int pop1(){
    int r;
    r=s1[top1];
    top1--;
    return r;
```

```
}
void delete1(){
    int i;
    if(top2==-1 && top1==-1){
        printf("queue is empty");
        }
    else if(top2==-1){
        for(i=top1;i>=0;i--){
            push2(pop1());
        }

        printf("Deleted iteam is %d\n",s2[top2--]);
    }
    else{
        printf("Deleted iteam is %d\n",s2[top2--]);
    }
}
void display(){
    int i;
    if(top1==-1 && top2==-1) printf("queue is empty\n");
    for(i=top2;i>=0;i--){
        printf("%d | ",s2[i]);
    }
    for(i=0;i<=top1;i++){
        printf("%d | ",s1[i]);
    }
    printf("\n");
```

```
}


int main()
{
    printf(" 1-insert \n 2-delete \n 3-display\n 4-exit\n");
    int ins,c=0;
    while(c!=4){
    printf("enter your choise : ");
    scanf("%d",&c);

    switch(c){
        case 1:
        printf("enter element: ");
        scanf("%d",&ins);
        insert(ins);
    break;
    case 2: delete1();
    break;
    case 3: display();

    break;
     default:       exit(0);
    break;
            }
                }
```

```
        return 0;
}
```

```
1-insert
 2-delete
 3-display
 4-exit
enter your choise : 1
enter element: 10
enter your choise : 1
enter element: 20
enter your choise : 1
enter element: 30
enter your choise : 3
10 | 20 | 30 |
enter your choise : 2
Deleted iteam is 10
enter your choise : 3
20 | 30 |
enter your choise : 2
Deleted iteam is 20
enter your choise : 3
30 |
enter your choise : 2
Deleted iteam is 30
enter your choise : 3
queue is empty

enter your choise : 4

Process returned 0 (0x0)   execution time : 57.880 s
Press any key to continue.
```

SINGLY LINK LIST

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

struct Node{
    int value;
    struct Node *ptr;
};

void InsertAtFront(struct Node **h,int val){
    struct Node* NewNode;
    NewNode = (struct Node*)malloc(sizeof(struct Node));
    NewNode->value=val;
    if(*h == NULL){
        NewNode->ptr = *h;
        *h = NewNode;
    }else{
        NewNode->ptr = *h ;
        *h = NewNode;
        }
}

void InsertAtEnd(struct Node **h,int val){
    struct Node *NewNode,*temp;
    NewNode = (struct Node*)malloc(sizeof(struct Node));
    NewNode->value = val;
    temp = *h;
```

```
    if(*h == NULL){
        *h = NewNode;
        NewNode->ptr = NULL;
    }else{
    while(temp->ptr != NULL)
        temp = temp->ptr;

        temp->ptr = NewNode;
        NewNode->ptr = NULL;
    }
}

void InsertAtOrder(struct Node **h,int val){
    struct Node *NewNode,*temp;
    NewNode = (struct Node*)malloc(sizeof(struct Node));
    NewNode->value = val;
    temp = *h;
    if(*h == NULL){
        *h = NewNode;
        NewNode->ptr = NULL;
    }else if(temp->value >= val ){
        InsertAtFront(h,val);
    }
    else{
        while(temp->ptr->value <= val){
            temp = temp->ptr;
            if(temp->ptr == NULL)
```

```
                break;
            }
            if(temp->ptr == NULL){
                InsertAtEnd(h,val);
            }else{
            NewNode->ptr = temp->ptr;
            temp->ptr = NewNode;
            }
        }

    }

    void ReverseLinklist(struct Node **h){
        struct Node *temp_current, *NEXT, *PREVOUS;
        temp_current = *h;
        PREVOUS = NULL;
        while(temp_current != NULL){
            NEXT = temp_current->ptr;
            temp_current->ptr = PREVOUS;
            PREVOUS = temp_current;
            temp_current = NEXT;
        }
        *h = PREVOUS;
    }

    void DELETE(struct Node **h){
        struct Node *temp,*NodetoBeDeleted;
```

```
    temp = *h;
    if(temp == NULL)printf("\nsorry, Link list is empty.\n");
    else if(temp->ptr == NULL){
        NodetoBeDeleted = temp;
        free(NodetoBeDeleted);
    }else{
        printf("Enter your choice:\n1 for Delete at first\n2 for delete at last\n3 for delete specific
value");
        int n;
        scanf("%d",&n);
        if(n == 1){
            NodetoBeDeleted = temp;
            temp = temp->ptr;
            (*h) = temp;
            free(NodetoBeDeleted);
        }else if(n == 2){
            while(temp->ptr->ptr != NULL)    temp = temp->ptr;
                free(temp->ptr);
                temp->ptr = NULL;
        }else if(n == 3){
            printf("\nenter specific value you want to delete:");
            int val;
            scanf("%d",&val);
            while(temp->ptr->value != val)
                temp = temp->ptr;
                NodetoBeDeleted = temp->ptr;
                temp->ptr = temp->ptr->ptr;
```

```
                free(NodetoBeDeleted);
            }
        }
    }

void a_shorting(struct Node **h){
    struct Node *temp,*c,*i,*j;
    int temp_value;
 /*  int n=0,p,q;
    temp = *h;
    c = *h;
    while(c != NULL){
        c = c->ptr;
        n=n+1;
    }
    printf("%d",n);
    for(p=0;p<n;p++){
        for(q=0;q<n-p-1;q++){
            if(temp->value > temp->ptr->value  && temp != NULL){
                temp_value = temp->value;
                temp->value = temp->ptr->value;
                temp->ptr->value = temp_value;
            }
        temp = temp->ptr;
    }
    temp=*h;
    }*/
```

```
    i=j=temp=*h;
    while(i != NULL){
        while(j->ptr != NULL){
                if(j->value > j->ptr->value ){
                temp_value = j->value;
                j->value = j->ptr->value;
                j->ptr->value = temp_value;
            }
            j = j->ptr;
        }
        i = i->ptr;
        j= *h;
    }

}
void clear_L(struct Node **h){
    struct Node *temp,*NodetobeDelete;
    temp = *h;
    while(temp != NULL){
            NodetobeDelete = temp;
            temp = temp->ptr;
            free(NodetobeDelete);

    }
    (*h) = temp;
}
```

```
void Display(struct Node**h){
    struct Node *temp;
    temp = *h;
    if(temp == NULL) printf("\n Ops!!! link list is null \n");
    else {while(temp != NULL){
        printf("%d ",temp->value);
        temp = temp->ptr;
    }
    }
}


int main(){
    struct Node *HEAD;
    int n,val;
    HEAD = NULL;
    printf("1 for insert value at front:\n");
    printf("2 for insert value at rear:\n");
    printf("3 for insert value according to order:\n");
    printf("4 for clear Link list:\n");
    printf("5 for reverse Link list\n");
    printf("6 for delete  Link Node\n");
    printf("7 for sorting Link list in ascending order \n");
    printf("8 for display the list\n");
    printf("9 for Exit\n");
```

```
printf("Enter which type of the operation you want to apply:  ");
scanf("%d",&n);

while(n !=9){

    switch(n){
    case 1:
        printf("Enter the value you want to insert:");
        scanf("%d",&val);
        InsertAtFront(&HEAD,val);
        break;
    case 2:
        printf("Enter the value you want to insert:");
        scanf("%d",&val);
        InsertAtEnd(&HEAD,val);
        break;
    case 3:
        printf("Enter the value you want to insert:");
        scanf("%d",&val);
        InsertAtOrder(&HEAD,val);
        break;
    case 4:
        clear_L(&HEAD);
        break;
    case 5:
        printf("Link list Reverse Successful\n");
        ReverseLinklist(&HEAD);
```

```
            break;
        case 6:
            DELETE(&HEAD);
            break;
        case 7:
            a_shorting(&HEAD);
            break;
        case 8:
            Display(&HEAD);
            printf("\n");
            break;
        default:
            printf("Enter specific value:\n");
        }

        printf("\nEnter which type of the operation you want to apply:  ");
        scanf("%d",&n);
    }
    return 0;

}
```

1 for insert value at front:
2 for insert value at rear:
3 for insert value according to order:
4 for clear Link list:
5 for reverse Link list
6 for delete Link Node
7 for sorting Link list in ascending order
8 for display the list
9 for Exit
Enter which type of the operation you want to apply: 1

Enter the value you want to insert:15

Enter which type of the operation you want to apply: 1
Enter the value you want to insert:27

Enter which type of the operation you want to apply: 1
Enter the value you want to insert:10

Enter which type of the operation you want to apply: 1
Enter the value you want to insert:17

Enter which type of the operation you want to apply: 1
Enter the value you want to insert:39

Enter which type of the operation you want to apply: 1
Enter the value you want to insert:45

Enter which type of the operation you want to apply: 8
45 39 17 10 27 15

Enter which type of the operation you want to apply: 5
Link list Reverse Successful

Enter which type of the operation you want to apply: 8
15 27 10 17 39 45

Enter which type of the operation you want to apply: 7

Enter which type of the operation you want to apply: 8
10 15 17 27 39 45

Enter which type of the operation you want to apply: 3
Enter the value you want to insert:25

Enter which type of the operation you want to apply: 8
10 15 17 25 27 39 45

Enter which type of the operation you want to apply: 5
Link list Reverse Successful

Enter which type of the operation you want to apply: 8
45 39 27 25 17 15 10

Enter which type of the operation you want to apply: 6
Enter your choice:
1 for Delete at first
2 for delete at last
3 for delete specific value1

Enter which type of the operation you want to apply: 8

39 27 25 17 15 10

Enter which type of the operation you want to apply: 6
Enter your choice:
1 for Delete at first
2 for delete at last
3 for delete specific value2

Enter which type of the operation you want to apply: 8
39 27 25 17 15

Enter which type of the operation you want to apply: 6
Enter your choice:
1 for Delete at first
2 for delete at last
3 for delete specific value3

enter specific value you want to delete:25

Enter which type of the operation you want to apply: 8
39 27 17 15

Enter which type of the operation you want to apply: 2
Enter the value you want to insert:17

Enter which type of the operation you want to apply: 8
39 27 17 15 17

Enter which type of the operation you want to apply: 4

Enter which type of the operation you want to apply: 8

Ops!!! link list is null

Enter which type of the operation you want to apply: 9

Process returned 0 (0x0)   execution time : 236.883 s
Press any key to continue.

CIRCULAR LINK LIST

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

struct node{
    int value;
    struct node *ptr;
};

struct node* insertOrder(struct node *head,int val){
    struct node* newNode,*temp;
    temp = head;
    newNode = (struct node*)malloc(sizeof(struct node));

    newNode->value = val;

    if(head == NULL){
        newNode->ptr = newNode;
        head = newNode;
    }
    else if(val <= head->value){
        newNode->ptr = head;

        while(temp->ptr != head)
            temp = temp->ptr;

        head = newNode;    // initialize head part here always
```

```
        temp->ptr = head;
    }
    else{
        while(temp->ptr != head && val > temp->ptr->value)
            temp = temp->ptr;

        newNode->ptr = temp->ptr;
        temp->ptr = newNode;
    }
    return head;
}


struct node* deleteVal(struct node* head,int val){
    struct node* temp,*nodeDeleted;
    temp = head;
    if(head == NULL){
        printf("List is empty\n");
    }
    else if(head->value == val){
        while(temp->ptr != head) {
            temp = temp->ptr;
        }
        temp->ptr = head->ptr;
        free(head);
        head = temp->ptr;
    }
```

```
    else{
        temp = head;
        while (temp->ptr != head && val != temp->ptr->value) {
            temp = temp->ptr;
        }

        if(temp->ptr == head)
            printf("Given value is not Found\n");
        else{
            nodeDeleted = temp->ptr;
            temp->ptr = temp->ptr->ptr;
            free(nodeDeleted);
        }
    }
    return head;
}

void display(struct node* head){
    struct node*temp = head;
    if(head == NULL)
        printf("The list is Empty\n");
    else{
        do{
            printf("%d ",temp->value);
            temp = temp->ptr;
        } while(temp != head);
        printf("\n");
```

```
    }
}

void main(){
    struct node * head;
    head= NULL;
    int n,val;

    printf("Enter 1 for insert in order\n");
    printf("Enter 2 for delete value\n");
    printf("Enter 3 for display\n");
    printf("Enter 4 for EXIT\n");
    scanf("%d",&n);

    while(n!= 4){
    switch(n){
    case 1:
        printf("Enter value you want to inserted\n");
        scanf("%d",&val);
        head = insertOrder(head,val);
        break;
    case 2:
        printf("Enter value you want to delete\n");
        scanf("%d",&val);
        head = deleteVal(head,val);
        break;
    case 3:
```

```
        display(head);
        break;
    default:
        printf("Enter proper value\n");
    }
    printf("Enter 1/2/3/4\n");
    scanf("%d",&n);
    }

}
```

```
Enter 1 for insert in order
Enter 2 for delete value
Enter 3 for display
Enter 4 for EXIT
1
Enter value you want to inserted
12
Enter 1/2/3/4
1
Enter value you want to inserted
54
Enter 1/2/3/4
1
Enter value you want to inserted
87
Enter 1/2/3/4
3
12 54 87
Enter 1/2/3/4
2
Enter value you want to delete
54
Enter 1/2/3/4
3
12 87
Enter 1/2/3/4
2
Enter value you want to delete
12
Enter 1/2/3/4
3
```

```
87
Enter 1/2/3/4
2
Enter value you want to delete
87
Enter 1/2/3/4
2
Enter value you want to delete
32
Given value is not Found
Enter 1/2/3/4
4

Process returned 4 (0x4)   execution time : 80.470 s
Press any key to continue.
```

DOBELY LINK LIST

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

struct node{
    int value;
    struct node *lptr,*rptr;
};

void insert(struct node **LH,struct node **RH,int val){
        struct node *NewNode,*temp;
        temp = *LH;
        NewNode = (struct node*)malloc(sizeof(struct node));
        NewNode->value = val;
        if(*LH == NULL && *RH == NULL){
            (*LH) = (*RH) = NewNode;
            NewNode->lptr = NewNode->rptr =NULL;
        }else{
            printf("1) insert at first\n2) insert at last\n3) insert at order\nWhere u want to insert :");
            int n;
            scanf("%d",&n);
            if(n==1){
                // insert at front
                NewNode->rptr = *LH;
                NewNode->lptr = NULL;
                (*LH)->lptr = NewNode;
                *LH = NewNode;
```

```
}else if(n==2){
    NewNode->rptr = NULL;
    NewNode->lptr = *RH;
    (*RH)->rptr = NewNode;
    *RH = NewNode;
}else if(n==3){
    if(temp->lptr == NULL){
        if(temp->value >= val){
    NewNode->rptr = *LH;
    NewNode->lptr = NULL;
    (*LH)->lptr = NewNode;
    *LH = NewNode;
        }
        else{
    NewNode->rptr = NULL;
    NewNode->lptr = *RH;
    (*RH)->rptr = NewNode;
    *RH = NewNode;
        }
    return;
    }
    if(temp->value >= val){
    NewNode->rptr = *LH;
    NewNode->lptr = NULL;
    (*LH)->lptr = NewNode;
    *LH = NewNode;
    return;
```

```
            }

            while(temp->rptr->value <= val)
                temp = temp->rptr;
             if(temp->rptr == *RH){
            NewNode->rptr = NULL;
            NewNode->lptr = *RH;
            (*RH)->rptr = NewNode;
            *RH = NewNode;
            }else{
                    NewNode->lptr = temp;
                    NewNode->rptr = temp->rptr;
                    temp->rptr->lptr = NewNode;
                    temp->rptr = NewNode;
            }
          }
          else printf("\nsomething wrong");
        }
}
void Display(struct node **LH,struct node ** RH){
    struct node *temp;
    temp = *LH;
    while(temp != NULL){
        printf(" %d ",temp->value);
        temp = temp->rptr;
    }
    printf("\n");
```

```
}


void delete_Link(struct node **LH,struct node **RH){
        struct node *NodetoBeDeleted,*temp;
        int n, val;
        temp = *LH;
        if(*LH == NULL ){
            printf("\nsorry Link List is Empty\n");
            return;
        }
        if(temp->lptr == NULL && temp->rptr == NULL){
           NodetoBeDeleted = temp;
           free(NodetoBeDeleted);
           *LH= *RH= NULL;
        }else{
           printf("1) Delete first Link\n2) Delete Last link\n3) Delete specific value  \nwhich value
u want to Delete");
           scanf("%d",&n);
           if(n==3){
              printf("Enter value :");
              scanf("%d",&val);
           }
           if(n==1){
              NodetoBeDeleted = *LH;
              (*LH) = (*LH)->rptr;
              free(NodetoBeDeleted);
              return;
```

```
}else if(n==2){
    NodetoBeDeleted = *RH;
     (*RH)->lptr->rptr = NULL;
    *RH = (*RH)->lptr;
    free(NodetoBeDeleted);
    return;

}else if(n==3){
    if((*RH)->value == val){
         NodetoBeDeleted = *RH;

    *RH = (*RH)->lptr;
    free(NodetoBeDeleted);
    return;
    }
    if((*LH)->value == val){
        NodetoBeDeleted = *LH;
    *LH = (*LH)->rptr;
    free(NodetoBeDeleted);
    return;
    }else{


    while(temp->rptr->value != val)
        temp = temp->rptr;

     NodetoBeDeleted = temp->rptr;
```

```
                temp->rptr = temp->rptr->rptr;
                temp->rptr->lptr = temp;
                free(NodetoBeDeleted);
               }
            }else{
            printf("sorry something wrong\n");
            }

        }
}

int main(){
    struct node *LH,*RH;
    LH = RH = NULL;
    int c,n;
    printf("1) Insert\n2) Delete\n3) Display\n4) Exit");
    while(c!=4){
        printf("\nEnter your choice :");
        scanf("%d",&c);
    switch(c){
    case  1:
        printf("Enter value :");
        scanf("%d",&n);
        insert(&LH,&RH,n);
        break;
    case 2:
        delete_Link(&LH,&RH);
```

```
        break;
    case 3:
        Display(&LH,&RH);
        break;
    case 4:
        exit(0);
        break;
    default:
        printf("Please Enter Between 1 to 4");
        break;

    }
          }

    return 0;
}
```

```
1) Insert
2) Delete
3) Display
4) Exit
Enter your choice :1
Enter value :12

Enter your choice :1
Enter value :37
1) insert at first
2) insert at last
3) insert at order
Where u want to insert :1

Enter your choice :1
Enter value :19
1) insert at first
2) insert at last
3) insert at order
Where u want to insert :3
```

Enter your choice :3
19 37 12

Enter your choice :1
Enter value :57
1) insert at first
2) insert at last
3) insert at order
Where u want to insert :2

Enter your choice :1
Enter value :47
1) insert at first
2) insert at last
3) insert at order
Where u want to insert :1

Enter your choice :3
47 19 37 12 57

Enter your choice :2
1) Delete first Link
2) Delete Last link
3) Delete specific value
which value u want to Delete1

Enter your choice :3
19 37 12 57

Enter your choice :2
1) Delete first Link
2) Delete Last link
3) Delete specific value
which value u want to Delete2

Enter your choice :2
1) Delete first Link
2) Delete Last link
3) Delete specific value
which value u want to Delete3
Enter value :37

Enter your choice :3
19 12

Enter your choice :2
1) Delete first Link
2) Delete Last link
3) Delete specific value

```
which value u want to Delete2

Enter your choice :3
19

Enter your choice :2
1) Delete first Link
2) Delete Last link
3) Delete specific value
which value u want to Delete1

Enter your choice :2

sorry Link List is Empty

Enter your choice :4

Process returned 0 (0x0)   execution time : 302.976 s
Press any key to continue.
```

CIRCULAR DOBELY LINK LIST

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

struct Node{
    int value;
    struct Node* lptr;
    struct Node* rptr;
};

void insert_front(struct Node**L,struct Node**R,int val){
    struct Node* newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->value = val;

    if(*L==NULL){
        newNode->lptr= newNode;
        newNode->rptr= newNode;
        *L=*R=newNode;
        return;
    }
    newNode->rptr = *L;
    newNode->lptr = *R;
    (*L)->lptr = newNode;
    *L = newNode;
    (*R)->rptr = *L;    // make circular List
```

```
}

void insert_rear(struct Node**L,struct Node**R,int val){
    struct Node* newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->value = val;

    if(*L==NULL){
        newNode->lptr= newNode;
        newNode->rptr= newNode;
        *L=*R=newNode;
        return;
    }
    newNode->lptr = *R;
    newNode->rptr = *L;
    (*R)->rptr = newNode;
    *R = newNode;
    (*L)->lptr = *R;      //make Circular List
}

void insert_order(struct Node**L,struct Node**R,int val){
    struct Node* newNode;
    struct Node* temp = *L;
    newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->value = val;
```

```
if(*L==NULL){
    newNode->lptr= newNode;
    newNode->rptr= newNode;
    *L=*R=newNode;
    return;
}

if(val <= (*L)->value){
    newNode->rptr = *L;
    newNode->lptr = *R;
    (*L)->lptr = newNode;
    *L = newNode;
    (*R)->rptr = *L;
}
else if(val >= (*R)->value){
    newNode->lptr = *R;
    newNode->rptr = *L;
    (*R)->rptr = newNode;
    *R = newNode;
    (*L)->lptr = *R;
}
else{
    while(temp->value <= val)
        temp = temp->rptr;

    newNode->lptr = temp->lptr;
```

```
        newNode->rptr = temp;
        temp->lptr->rptr = newNode;
        temp->lptr = newNode;
    }
}

void insert_specs(struct Node**L,struct Node**R,int val){
    int sval,n;
    struct Node* newNode;
    struct Node* temp = *L;
    newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->value = val;
    //here our list is not empty because user give us value which we have to put after some value,
    //so the list is not empty and we dont have to write condition for null list

    do{
    printf("Enter after which value you want to insert:\n");
    scanf("%d",&sval);
    n=1; //initialize n here.
    temp = *L; //initialize temp also because if one time value not found then temp comes to head
part again for new loop
    while(temp->value != sval){
        temp = temp->rptr;

        if(temp== NULL){ //this condition is special because written in while loop
            printf("your given value is not found\n");
```

```
            n=0;
            break;
        }
    }
    }while(n==0);

    newNode->lptr = temp;
    newNode->rptr = temp->rptr;
    if(temp != *R)  temp->rptr->lptr = newNode; // write this always above the below sentences.
    temp->rptr = newNode;

    if(temp == *R){ //upgradation of rear and front value
        *R = newNode;
        (*L)->lptr = *R;
    }
}


void del_front(struct Node** Head,struct Node ** Rear){
            struct Node * delete;

            if(*Head == NULL){
                        printf("The list is Empty\n");
                        return;
            }
    if((*Head)->rptr == NULL && (*Head)->lptr == NULL){
        printf("%d is removed\n",(*Head)->value);
```

```
        free(*Head);
        *Head = *Rear = NULL;
        return;
    }
            delete = *Head;
            *Head = (*Head)->rptr;
            (*Head)->lptr = *Rear;
    (*Rear)->rptr = *Head;
    printf("%d is removed\n",delete->value);
            free(delete);
}

void del_end(struct Node** Head,struct Node** Rear){
            struct Node * delete;

            if(*Head == NULL){
                        printf("The list is Empty\n");
                        return;
            }
    if(*Head == *Rear){
        printf("%d is removed\n",(*Head)->value);
        free(*Head);
        *Head = *Rear = NULL;
        return;
    }

    delete = *Rear;
```

```
            (*Rear) = (*Rear)->lptr;
            (*Rear)->rptr = *Head;
    (*Head)->lptr = *Rear;
    printf("%d is removed\n",delete->value);
            free(delete);
}

void del_specific(struct Node** Head,struct Node** Rear,int val){
            struct Node * temp;
            temp = *Head;

            if(*Head == NULL){
                        printf("The list is Empty\n");
                        return;
            }

            if( val == (*Head)->value){
        if(*Head == *Rear){         //condition for only Node
            printf("%d is removed\n",(*Head)->value);
            free(*Head);
            *Head = *Rear = NULL;
        }
        else{
            temp = *Head;
            *Head = (*Head)->rptr;
            (*Head)->lptr = *Rear;
            (*Rear)->rptr = *Head;
```

```
    printf("%d is removed\n",temp->value);
    free(temp);
}
    }
    else if(val == (*Rear)->value){

            temp = *Rear;
            (*Rear) = (*Rear)->lptr;
            (*Rear)->rptr = *Head;
(*Head)->lptr = *Rear;
printf("%d is removed\n",temp->value);
            free(temp);
    }
    else{
            while (temp != NULL && temp->value != val)
                    temp = temp->rptr;

            if(temp == NULL){
                    printf("Value is not found\n");
                    return;
            }
            //temp points to the node to be deleted

            temp->lptr->rptr = temp->rptr;
            temp->rptr->lptr = temp->lptr;
printf("%d is removed\n",temp->value);
            free(temp);
```

```
            }
    }

    void display(struct Node * H){
        struct Node *temp;
        temp = H;

        if(temp == NULL){
            printf("List is empty\n");
            return;
        }

                printf("The Numbers in list is..");
        do{
            printf("%d ",temp->value);
            temp = temp->rptr;
        }while(temp!= H);
        printf("\n");

    }


    int main(){
        struct Node* Head;
        struct Node* Rear;
        Head = Rear = NULL;
        int val,n,sval;
```

```
printf("1 for insert value at front:\n");
printf("2 for insert value at rear:\n");
printf("3 for insert value according to order:\n");
printf("4 for insert value after specific value:\n");
printf("5 for delete from front\n");
        printf("6 for delete from end\n");
        printf("7 for delete specific value\n");
printf("8 for display\n");
        printf("9 for Exit\n");
scanf("%d",&n);

while(n !=9){

    switch(n){
    case 1:
        printf("Enter the value you want to insert:\n");
        scanf("%d",&val);
        insert_front(&Head,&Rear,val);
        break;
    case 2:
        printf("Enter the value you want to insert:\n");
        scanf("%d",&val);
        insert_rear(&Head,&Rear,val);
        break;
    case 3:
        printf("Enter the value you want to insert:\n");
```

```
    scanf("%d",&val);
    insert_order(&Head,&Rear,val);
    break;
case 4:
    printf("Enter the value you want to insert:\n");
    scanf("%d",&val);
    insert_specs(&Head,&Rear,val);
    break;
        case 5:
            del_front(&Head,&Rear);
            break;
        case 6:
            del_end(&Head,&Rear);
            break;
        case 7:
            printf("Enter value you want to be deleted\n");
            scanf("%d",&val);
            del_specific(&Head,&Rear,val);
            break;
case 8:
    display(Head);
    break;
default:
    printf("Enter specific value:\n");
}

printf("Enter which type of the operation you want to apply:\n");
```

```
        scanf("%d",&n);
    }
    return 0;
}
```

```
1 for insert value at front:
2 for insert value at rear:
3 for insert value according to order:
4 for insert value after specific value:
5 for delete from front
6 for delete from end
7 for delete specific value
8 for display
9 for Exit
1
Enter the value you want to insert:
12
Enter which type of the operation you want to apply:
1
Enter the value you want to insert:
14
Enter which type of the operation you want to apply:
2
Enter the value you want to insert:
54
Enter which type of the operation you want to apply:
2
Enter the value you want to insert:
32
Enter which type of the operation you want to apply:
3
Enter the value you want to insert:
5
Enter which type of the operation you want to apply:
4
Enter the value you want to insert:
54
Enter after which value you want to insert:
32
Enter which type of the operation you want to apply:
8
The Numbers in list is..5 14 12 54 32 54
Enter which type of the operation you want to apply:
5
5 is removed
Enter which type of the operation you want to apply:
8
The Numbers in list is..14 12 54 32 54
```

```
Enter which type of the operation you want to apply:
6
54 is removed
Enter which type of the operation you want to apply:
8
The Numbers in list is..14 12 54 32
Enter which type of the operation you want to apply:
7
Enter value you want to be deleted
12
12 is removed
Enter which type of the operation you want to apply:
8
The Numbers in list is..14 54 32
Enter which type of the operation you want to apply:
6
32 is removed
Enter which type of the operation you want to apply:
7
Enter value you want to be deleted
14
14 is removed
Enter which type of the operation you want to apply:
8
The Numbers in list is..54
Enter which type of the operation you want to apply:
2
Enter the value you want to insert:
10
Enter which type of the operation you want to apply:
6
10 is removed
Enter which type of the operation you want to apply:
6
54 is removed
Enter which type of the operation you want to apply:
6
The list is Empty
Enter which type of the operation you want to apply:
9

Process returned 0 (0x0)   execution time : 172.692 s
Press any key to continue.
```

CAR AGENCY PROBLEM

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXS 20


int front = 0,rear= -1,top=-1;
struct Node{
    char car[20];
    struct Node *ptr;
    int n;
};

struct stack1{
        char soldCar[20];
};

struct stack1 sold[20];

struct Node *insCar(struct Node *head){
        struct Node *NewNode;
        struct Node *temp;
        temp = head;

        NewNode = (struct Node*)malloc(sizeof(struct Node));
                if(NewNode == NULL){
```

```
            printf("SORRY MALLOC FAILD\n");
        }

        printf("enter the car brand name :");
        scanf("%s",(NewNode->car));

        printf("How many %s car u have :",NewNode->car);
        scanf("%d",&(NewNode->n));
        printf("\n");

        NewNode->ptr = NULL;
        if(temp == NULL){
        head = NewNode;
        }else{
        while(temp->ptr != NULL)
        temp = temp->ptr;
        temp->ptr = NewNode;
        }

        return head;

}
void push(char c[]){
    strcpy(sold[++top].soldCar,c);
}

void Buy(struct Node *head){
```

```
printf("\n");
printf("costumer %d :\n",top+2);
printf("which car u Want to Buy :");
char costumerCar[20];
scanf("%s",costumerCar);
struct Node *temp;
temp = head;
while(strcmp(temp->car,costumerCar)){
        temp = temp->ptr;
        if(temp == NULL){
            break;
        }
        }
if(temp == NULL){
        printf("that car Brand is not available\n");
        Buy(head);

}else{
    if(temp->n == 0){
        printf("Sorry Out of stock\n");
        Buy(head);
    }
    else{
        (temp->n)--;
        push(costumerCar);

    }
```

```
        }

}


void display(struct Node *head){
    struct Node *temp;
    temp = head;
    printf("\n\n");

    while(temp != NULL){
        printf("car : %s | amount : %d\n", temp->car,temp->n);
        temp = temp->ptr;
    }
    }
int main(){
    struct Node *Head;
    Head = NULL;
        printf("**********************************\n\tWelcome to car
shop\n**********************************");
        int b,i;
        printf("\nHow many car Brand u Want to add :");
        scanf("%d",&b);
        printf("\n");
        for(i=0;i<b;i++){
                Head = insCar(Head);
        }
```

```
        printf("\n");
        printf("How many costumers :");
        int c;
        scanf("%d",&c);
        printf("\n");
   for(i=0;i<c;i++){
                Buy(Head);
        }

        display(Head);
        printf("\n\nlast sold car is %s\n\n",sold[top].soldCar);
}
```

```
**********************************
Welcome to car shop
**********************************
How many car Brand u Want to add :4

enter the car brand name :BMW
How many BMW car u have :3

enter the car brand name :Audi
How many Audi car u have :2

enter the car brand name :Bugatti
How many Bugatti car u have :1

enter the car brand name :Ford
How many Ford car u have :6


How many costumers :7


costumer 1 :
which car u Want to Buy :Audi

costumer 2 :
which car u Want to Buy :BMW
```

```
costumer 3 :
which car u Want to Buy :Bugatti

costumer 4 :
which car u Want to Buy :Ford

costumer 5 :
which car u Want to Buy :Ford

costumer 6 :
which car u Want to Buy :Bugatti
Sorry Out of stock

costumer 6 :
which car u Want to Buy :Audi

costumer 7 :
which car u Want to Buy :Ford


car : BMW | amount : 2
car : Audi | amount : 0
car : Bugatti | amount : 0
car : Ford | amount : 3


last sold car is Ford


Process returned 0 (0x0)   execution time : 168.662 s
Press any key to continue.
```

BINARY SEARCH TREE

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

struct BSTnode{
  int Data;
  struct BSTnode* LeftNode;
  struct BSTnode* RightNode;
};

 struct BSTnode* InsertNode(struct BSTnode* Root,int Data){
  struct BSTnode *temp,*NewNode;
  NewNode = (struct BSTnode*)malloc(sizeof(struct BSTnode));
  if(NewNode == NULL){
    printf("Sorry Malloc is fail...!!!\n");
    return Root;
  }

  NewNode->Data = Data;
  NewNode->LeftNode = NULL;
  NewNode->RightNode = NULL;



  temp = Root;

  /*while (temp != NULL ) {
    if(temp->Data > Root->Data){
```

```
      temp = Root->RightNode;
    }else{
      temp = Root->LeftNode;
    }
  }
  temp = NewNode;*/

  if(Root == NULL){
    Root = NewNode;
    return Root;
  }else if(Data > Root->Data){
    Root->RightNode = InsertNode(Root->RightNode,Data);
  }else{
    Root->LeftNode = InsertNode(Root->LeftNode,Data);
  }
  return temp;
}

int Search(struct BSTnode* Root,int Data){
  if(Root == NULL) return 0;
  if(Root->Data == Data) return 1;
  else if(Root->Data < Data) return Search(Root->RightNode,Data);
  else return Search(Root->LeftNode,Data);

/*  while(Data != Root->Data && Root != NULL){

    if(Data > Root->Data)
```

```
      Root = Root->RightNode;
    else
      Root = Root->LeftNode;
    }
    if(Root == NULL){
      printf("Node Not Avalable.....\n");
      return 0;
    }else{
      return 1;
    }*/

}

int FindMin(struct BSTnode *Root){
  if(Root->LeftNode == NULL)
    return Root->Data;
  else
    FindMin(Root->LeftNode);

/*  while(Root != NULL){
    Root = Root->LeftNode;
  }
  Return Root->Data;*/

}

int FindMax(struct BSTnode *Root){
```

```
  if(Root->RightNode == NULL)
    return Root->Data;
  else
    FindMax(Root->RightNode);

/*  while(Root != NULL){
    Root = Root->RightNode;
  }
  return Root->Data;*/
}

int MaxHight(struct BSTnode *Root){
  int L_Hight=0,R_Hight=0;
  if(Root == NULL)
    return -1;
  if(Root->LeftNode == NULL && Root->RightNode == NULL)
    return 0;
  L_Hight = MaxHight(Root->LeftNode);
  R_Hight = MaxHight(Root->RightNode);
  if(L_Hight > R_Hight)
    return L_Hight+1;
  else
    return R_Hight+1;
}

 void INORDER_Traversal(struct BSTnode *Root){
   if(Root == NULL)
```

```
  return;
  INORDER_Traversal(Root->LeftNode);
  printf("%d ",Root->Data);
  INORDER_Traversal(Root->RightNode);
}

void PREORDER_Traversal(struct BSTnode *Root){
  if(Root == NULL)
  return;
  printf("%d ",Root->Data);
  PREORDER_Traversal(Root->LeftNode);
  PREORDER_Traversal(Root->RightNode);
}

  void POSTORDER_Traversal(struct BSTnode *Root){
    if(Root == NULL)
    return;
    POSTORDER_Traversal(Root->LeftNode);
    POSTORDER_Traversal(Root->RightNode);
    printf("%d ",Root->Data);
}
int front=-1,rear=-1;
#define MaxSize 100
struct BSTnode* Q[MaxSize];

void InQueue(struct BSTnode* Data){
  if(rear == MaxSize){
```

```
    printf("Sorry Queue is Full\n");
  }else{
    if(front == -1 ) front++;
    Q[++rear] = Data;
  }
}

struct BSTnode* Dequeue(){
  //if(front == -1) return
  if(front == rear){
    struct BSTnode* r = Q[front];
    front = -1;
    rear = -1;
    return r;
  }else{
  return Q[front++];
  }
}

void LEVELORDER_Traversal(struct BSTnode *Root){
  InQueue(Root);
  struct BSTnode *Temp;
  while (front != -1) {
    Temp = Dequeue();
    if( Temp != NULL ){
      InQueue(Temp->LeftNode);
      InQueue(Temp->RightNode);
```

```
      printf("%d ",Temp->Data);
      }
    }
    printf("\n");
  }

struct BSTnode * DeleteNode(struct BSTnode* Root,int key){
struct BSTnode *Temp;
  if(Root == NULL )
    return Root;
  if(key > Root->Data)
    Root->RightNode = DeleteNode(Root->RightNode,key);
  else if(key < Root->Data)
    Root->LeftNode = DeleteNode(Root->LeftNode,key);
    else{ // Root->Data == key;
      if(Root->LeftNode == NULL && Root->RightNode == NULL){
        // node with 0 child
        free(Root);
        return NULL;
      }
      else if(Root->LeftNode == NULL){
        // node with 1 child
        Temp = Root->RightNode;
        free(Root);
        return Temp;
      }else if(Root->RightNode == NULL){
        // node with 1 child
```

```
        Temp = Root->LeftNode;
        free(Root);
        return Temp;
      }else{
        // node with 2 childe
        // find min in right sub-Tree
        Temp = Root->RightNode;
        while(Temp->LeftNode != NULL){
          Temp = Temp->LeftNode;
        }
        Root->Data = Temp->Data;
        Root->RightNode = DeleteNode(Root->RightNode,Temp->Data);
      }
      return Root;
    }

}

int main(){
  int c,n,i;
  struct BSTnode *Root= NULL;
  printf("1) InsertNode\n2) Hight of Tree\n3) Find Max Data\n4) Find Min Data\n5) Search
Data\n6) INORDER_Traversal\n7) PREORDER_Traversal\n8) POSTORDER_Traversal\n9)
LEVELORDER_Traversal\n10) DeleteNode\n11) Exit\n\n");
  while (c != 11) {
    printf("Enter Your choice :");
    scanf("%d",&c);
    switch (c) {
```

```
case 1:
  printf("Enter Data :");
  scanf("%d",&n);
  Root = InsertNode(Root,n);
  break;
case 2:
  i = MaxHight(Root);
  printf("Hight Of tree is %d\n",i);
  break;
case 3:
  i = FindMax(Root);
  printf("Max Data of Tree is %d\n",i);
  break;
case 4:
  i = FindMin(Root);
  printf("Min Data of tree is %d\n",i);
  break;
case 5:
  printf("Find Data in Tree :");
  scanf("%d",&n);
  i= Search(Root,n);
  if(i == 1){
    printf("Data found\n");
  }else{
    printf("Data Not found\n");
  }
  break;
```

```
case 6:
  printf("INORDER_Traversal of Tree is....");
  INORDER_Traversal(Root);
  printf("\n");
  break;
case 7:
  printf("PREORDER_Traversal of Tree is....");
  PREORDER_Traversal(Root);
  printf("\n");
  break;
case 8:
  printf("POSTORDER_Traversal of Tree is....");
  POSTORDER_Traversal(Root);
  printf("\n");
  break;
case 9:
  printf("LEVELORDER_Traversal of Tree is....");
  LEVELORDER_Traversal(Root);
  break;
case 10:
  printf("which Node you want to delete :");
  scanf("%d",&n);
  DeleteNode(Root,n);
  break;
case 11:
  exit(0);
break;
```

```
        default:
            printf("Enter valid choice between 1-10\n");
        break;
      }


   }
   return 0;
}
```

```
1) InsertNode
2) Hight of Tree
3) Find Max Data
4) Find Min Data
5) Search Data
6) INORDER_Traversal
7) PREORDER_Traversal
8) POSTORDER_Traversal
9) LEVELORDER_Traversal
10) DeleteNode
11) Exit

Enter Your choice :1
Enter Data :50
Enter Your choice :1
Enter Data :60
Enter Your choice :1
Enter Data :40
Enter Your choice :1
Enter Data :55
Enter Your choice :1
Enter Data :45
Enter Your choice :1
Enter Data :35
Enter Your choice :1
Enter Data :47
Enter Your choice :1
Enter Data :70
Enter Your choice :1
Enter Data :67
Enter Your choice :1
Enter Data :52
Enter Your choice :1
Enter Data :57
Enter Your choice :2
```

```
Hight Of tree is 3
Enter Your choice :3
Max Data of Tree is 70
Enter Your choice :4
Min Data of tree is 35
Enter Your choice :5
Find Data in Tree :45
Data found
Enter Your choice :5
Find Data in Tree :43
Data Not found
Enter Your choice :6
INORDER_Traversal of Tree is....35 40 45 47 50 52 55 57 60 67 70
Enter Your choice :7
PREORDER_Traversal of Tree is....50 40 35 45 47 60 55 52 57 70 67
Enter Your choice :8
POSTORDER_Traversal of Tree is....35 47 45 40 52 57 55 67 70 60 50
Enter Your choice :9
LEVELORDER_Traversal of Tree is....50 40 60 35 45 55 70 47 52 57 67
Enter Your choice :10
which Node you want to delete :50
Enter Your choice :6
INORDER_Traversal of Tree is....35 40 45 47 52 55 57 60 67 70
Enter Your choice :10
which Node you want to delete :60
Enter Your choice :6
INORDER_Traversal of Tree is....35 40 45 47 52 55 57 67 70
Enter Your choice :9
LEVELORDER_Traversal of Tree is....52 40 67 35 45 55 70 47 57
Enter Your choice :10
which Node you want to delete :52
Enter Your choice :9
LEVELORDER_Traversal of Tree is....55 40 67 35 45 57 70 47
Enter Your choice :11

Press any key to continue . . .
```

BT TO BST

```
#include<stdio.h>
#include<stdlib.h>
int current = 0;
struct BSTnode{
  int Data;
  struct BSTnode* LeftNode;
  struct BSTnode* RightNode;
};

 struct BSTnode* InsertNode(struct BSTnode* Root,int Data){
  struct BSTnode *temp,*NewNode;
  NewNode = (struct BSTnode*)malloc(sizeof(struct BSTnode));
  if(NewNode == NULL){
    printf("Sorry Malloc is fail...!!!\n");
    return Root;
  }

  NewNode->Data = Data;
  NewNode->LeftNode = NULL;
  NewNode->RightNode = NULL;



  temp = Root;

  /*while (temp != NULL ) {
    if(temp->Data > Root->Data){
```

```
      temp = Root->RightNode;
    }else{
      temp = Root->LeftNode;
    }
  }
  temp = NewNode;*/

  if(Root == NULL){
    Root = NewNode;
    return Root;
  }else if(Data > Root->Data){
    Root->RightNode = InsertNode(Root->RightNode,Data);
  }else{
    Root->LeftNode = InsertNode(Root->LeftNode,Data);
  }
  return temp;
}
void INORDER_Traversal(struct BSTnode *Root){
  if(Root == NULL)
  return;
  INORDER_Traversal(Root->LeftNode);
  printf("%d ",Root->Data);
  INORDER_Traversal(Root->RightNode);
}

void PREORDER_Traversal(struct BSTnode *Root){
  if(Root == NULL)
```

```
  return;
  printf("%d ",Root->Data);
  PREORDER_Traversal(Root->LeftNode);
  PREORDER_Traversal(Root->RightNode);
}

 void POSTORDER_Traversal(struct BSTnode *Root){
   if(Root == NULL)
   return;
   POSTORDER_Traversal(Root->LeftNode);
   POSTORDER_Traversal(Root->RightNode);
   printf("%d ",Root->Data);
}
int front=-1,rear=-1;
#define MaxSize 100
struct BSTnode* Q[MaxSize];

void InQueue(struct BSTnode* Data){
 if(rear == MaxSize){
   printf("Sorry Queue is Full\n");
 }else{
   if(front == -1 ) front++;
   Q[++rear] = Data;
 }
}

struct BSTnode* Dequeue(){
```

```
 //if(front == -1) return
 if(front == rear){
   struct BSTnode* r = Q[front];
   front = -1;
   rear = -1;
   return r;
 }else{
 return Q[front++];
 }
}

void LEVELORDER_Traversal(struct BSTnode *Root){
 InQueue(Root);
 struct BSTnode *Temp;
 while (front != -1) {
   Temp = Dequeue();
   if( Temp != NULL ){
     InQueue(Temp->LeftNode);
     InQueue(Temp->RightNode);
     printf("%d ",Temp->Data);
     }
   }
   printf("\n");
 }
int main(){
  struct BSTnode*Root = NULL;
  int n[100]={0},c,h=0;
```

```
  printf("1) InsertNode In Binary tree in LEVELORDER\n2) Convert Tree in Binary-Search-Tree
Display in INORDER_Traversal\n");

  printf("3) Convert Tree in Binary-Search-Tree Display in PREORDER_Traversal\n4) Convert
Tree in Binary-Search-Tree Display in POSTORDER_Traversal\n");

  printf("5) Convert Tree in Binary-Search-Tree Display in LEVELORDER_Traversal\n6) Exit\n");

  while(c != 6){

    printf("Enter your choice :");

    scanf("%d",&c);

    switch (c) {

      case 1:

        printf("Enter Node in Binary Value in LEVELORDER: ");

        scanf("%d",&n[current++]);

      break;

      case 2:

       h=0;

        printf("your Binary-Search-Tree in INORDER_Traversal is...");

        while (n[h] != 0) {

        Root= InsertNode(Root,n[h++]);

        }

        INORDER_Traversal(Root);

        Root = NULL;

        printf("\n");

      break;

      case 3:

       h=0;

      printf("your Binary-Search-Tree in PREORDER_Traversal is...");

      while (n[h] != 0) {

      Root= InsertNode(Root,n[h++]);
```

```
}
PREORDER_Traversal(Root);
printf("\n");
Root = NULL;
break;
case 4:
 h=0;
printf("your Binary-Search-Tree in POSTORDER_Traversal is...");
while (n[h] != 0) {
Root= InsertNode(Root,n[h++]);
}
POSTORDER_Traversal(Root);
Root = NULL;
printf("\n");
break;
case 5:
 h=0;
printf("your Binary-Search-Tree in LEVELORDER_Traversal is...");
while (n[h] != 0) {
Root= InsertNode(Root,n[h++]);
}
LEVELORDER_Traversal(Root);
Root = NULL;
break;
case 6:
  exit(0);
  break;
```

```
    }
  }
  return 0;
}
```

```
1) InsertNode In Binary tree in LEVELORDER
2) Convert Tree in Binary-Search-Tree Display in INORDER_Traversal
3) Convert Tree in Binary-Search-Tree Display in PREORDER_Traversal
4) Convert Tree in Binary-Search-Tree Display in POSTORDER_Traversal
5) Convert Tree in Binary-Search-Tree Display in LEVELORDER_Traversal
6) Exit
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 12
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 14
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 1
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 14
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 87
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 5
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 153
Enter your choice :1
Enter Node in Binary Value in LEVELORDER: 6
Enter your choice :2
your Binary-Search-Tree in INORDER_Traversal is...1 5 6 12 14 14 87 153
Enter your choice :3
your Binary-Search-Tree in PREORDER_Traversal is...12 1 5 6 14 14 87 153
Enter your choice :4
your Binary-Search-Tree in POSTORDER_Traversal is...6 5 1 14 153 87 14 12
Enter your choice :5
your Binary-Search-Tree in LEVELORDER_Traversal is...12 1 14 5 14 87 6 153
Enter your choice :6

Press any key to continue . . .
```

AVL TREE

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

struct AVLnode{
  int Data;
  int Hight;
  struct AVLnode* LeftNode;
  struct AVLnode* RightNode;
};
int GetHight(struct AVLnode* Node){
  if(Node == NULL)  return -1;
  if(Node->LeftNode == NULL && Node->RightNode == NULL) return 0;
  int Hight,LeftChild_Hight,RightChild_Hight;
  LeftChild_Hight = GetHight(Node->LeftNode);
  RightChild_Hight = GetHight(Node->RightNode);
  if(LeftChild_Hight > RightChild_Hight)
    return LeftChild_Hight+1;
  else
    return RightChild_Hight+1;
}
int GetBalance(struct AVLnode * Node){
  if(Node == NULL)  return 0;
  return GetHight(Node->LeftNode)-GetHight(Node->RightNode);
}
```

```
struct AVLnode *SingleRightRotation(struct AVLnode *Parent){
    struct AVLnode* LeftChild;
     LeftChild = Parent->LeftNode;
     Parent->LeftNode = LeftChild->RightNode;
     LeftChild->RightNode = Parent;
     Parent->Hight =GetHight(Parent);
     LeftChild->Hight = GetHight(LeftChild);
     return LeftChild;
}
struct AVLnode *SingleLeftRotation(struct AVLnode *Parent){
    struct AVLnode* RightChild;
     RightChild = Parent->RightNode;
     Parent->RightNode = RightChild->LeftNode;
     RightChild->LeftNode = Parent;
     Parent->Hight =GetHight(Parent);
     RightChild->Hight = GetHight(RightChild);
     return RightChild;
}
struct AVLnode *DobleLeftRightRotation(struct AVLnode *Parent){
struct AVLnode *LeftChild;;
  LeftChild = Parent->LeftNode;
  Parent->LeftNode = SingleLeftRotation(Parent->LeftNode);
  Parent = SingleRightRotation(Parent);
  return Parent;
}
struct AVLnode *DobleRightLeftRotation(struct AVLnode *Parent){
  Parent->RightNode = SingleRightRotation(Parent->RightNode);
```

```
  Parent = SingleLeftRotation(Parent);
  return Parent;
}

struct AVLnode* InsertNode(struct AVLnode* Root,int val){
  if(Root == NULL){
    struct AVLnode *NewNode;
    NewNode = (struct AVLnode*)malloc(sizeof(struct AVLnode));
      if(NewNode == NULL){
        printf(" Sorry! malloc is fail\n");
        return Root;
      }
      NewNode->Data = val;
      NewNode->Hight = 0;
      NewNode->LeftNode = NULL;
      NewNode->RightNode = NULL;
      return NewNode;
      }
    if(val < Root->Data){
      Root->LeftNode = InsertNode(Root->LeftNode,val);
      if(GetBalance(Root) == 2){
        // heavy Left Sub-tree
        if(val < (Root->LeftNode)->Data){
          // left-left case
          Root = SingleRightRotation(Root);
        }else{
          // left-Right case
```

```
        Root = DobleLeftRightRotation(Root);
      }
     }
    }else{
     Root->RightNode = InsertNode(Root->RightNode,val);
     if(GetBalance(Root) == -2){
      // right-sub tree heavy
      if(val > (Root->RightNode)->Data){
       // right - right case
       Root= SingleLeftRotation(Root);
      }else{
       // right - left case
       Root = DobleRightLeftRotation(Root);
      }
     }
    }

  Root->Hight = GetHight(Root);
  return Root;
}

struct AVLnode* DeleteNode(struct AVLnode* Root,int val){
  struct AVLnode *temp;
  int Balance;
  if(Root == NULL)  return Root;
  else if(val < Root->Data)
    Root->LeftNode = DeleteNode(Root->LeftNode,val);
```

```
else if(val > Root->Data)
  Root->RightNode = DeleteNode(Root->RightNode,val);
else{
  // Root->Data == val
  if(Root->LeftNode == NULL &&  Root->RightNode == NULL){
    // Node with 0 childe
    free(Root);
    return NULL;
  }else if(Root->LeftNode == NULL){
    // Node with only Right child
    temp = Root->RightNode;
    free(Root);
    return temp;
  }else if(Root->RightNode == NULL ){
    // Node with only Left Child
    temp = Root->LeftNode;
    free(Root);
    return temp;
  }else{
    // Node with 2 child
    temp = Root->RightNode;
    while(temp->LeftNode != NULL){
      temp = temp->LeftNode;
    }
    printf("%d\n",temp->Data);
    Root->Data = temp->Data;
    Root->RightNode = DeleteNode(Root->RightNode,temp->Data);
```

```
    }
    if(GetBalance(Root)== 2){
      if(GetBalance(Root->LeftNode)>0){
        //left-left case
        return SingleRightRotation(Root);
      }else{
        // left-right case
        return DobleLeftRightRotation(Root);
      }
    if(GetBalance(Root)== -2){
      if(GetBalance(Root->RightNode)<0){
        // Right-right case
        return SingleRightRotation(Root);
      }else{
        //Right-left case
        return DobleLeftRightRotation(Root);
      }
    }
    }
  }
  return Root;
}
void INORDER_Traversal(struct AVLnode *Root){
  if(Root == NULL)
  return;
  INORDER_Traversal(Root->LeftNode);
  printf("%d ",Root->Data);
```

```
  INORDER_Traversal(Root->RightNode);
}
void PREORDER_Traversal(struct AVLnode *Root){
  if(Root == NULL)
  return;
  printf("%d ",Root->Data);
  PREORDER_Traversal(Root->LeftNode);
  PREORDER_Traversal(Root->RightNode);
}

 void POSTORDER_Traversal(struct AVLnode *Root){
   if(Root == NULL)
   return;
   POSTORDER_Traversal(Root->LeftNode);
   POSTORDER_Traversal(Root->RightNode);
   printf("%d ",Root->Data);
}
int front=-1,rear=-1;
#define MaxSize 100
struct AVLnode* Q[MaxSize];

void InQueue(struct AVLnode* Data){
  if(rear == MaxSize){
    printf("Sorry Queue is Full\n");
  }else{
    if(front == -1 ) front++;
    Q[++rear] = Data;
```

```
  }
}

struct AVLnode* Dequeue(){
  //if(front == -1) return
  if(front == rear){
    struct AVLnode* r = Q[front];
    front = -1;
    rear = -1;
    return r;
  }else{
  return Q[front++];
  }
}

void LEVELORDER_Traversal(struct AVLnode *Root){
  InQueue(Root);
  struct AVLnode *Temp;
  while (front != -1) {
    Temp = Dequeue();
    if( Temp != NULL ){
      InQueue(Temp->LeftNode);
      InQueue(Temp->RightNode);
      printf("%d ",Temp->Data);
      }
    }
    printf("\n");
```

```
}
int Search(struct AVLnode* Root,int Data){
  if(Root == NULL) return 0;
  if(Root->Data == Data) return 1;
  else if(Root->Data < Data) return Search(Root->RightNode,Data);
  else return Search(Root->LeftNode,Data);

/*  while(Data != Root->Data && Root != NULL){

    if(Data > Root->Data)
      Root = Root->RightNode;
    else
      Root = Root->LeftNode;
    }
    if(Root == NULL){
      printf("Node Not Avalable.....\n");
      return 0;
    }else{
      return 1;
    }*/

}


int main(){
  int c,n,i;
  struct AVLnode *Root= NULL;
```

```
  printf("1) InsertNode\n2) Hight of Tree\n3) Search Data\n4) INORDER_Traversal\n5)
PREORDER_Traversal\n6) POSTORDER_Traversal\n7) LEVELORDER_Traversal\n8)
DeleteNode\n9) Exit\n\n");
  while(c != 9) {
    printf("Enter Your choice :");
    scanf("%d",&c);
    switch (c) {
      case 1:
        printf("Enter Data :");
        scanf("%d",&n);
        Root = InsertNode(Root,n);
        break;
      case 2:
        i = GetHight(Root);
        printf("Hight Of tree is %d\n",i);
        break;
      case 3:
        printf("Find Data in Tree :");
        scanf("%d",&n);
        i= Search(Root,n);
        if(i == 1){
          printf("Data found\n");
        }else{
          printf("Data Not found\n");
        }
        break;
      case 4:
        printf("INORDER_Traversal of Tree is....");
```

```
  INORDER_Traversal(Root);
  printf("\n");
  break;
case 5:
  printf("PREORDER_Traversal of Tree is....");
  PREORDER_Traversal(Root);
  printf("\n");
  break;
case 6:
  printf("POSTORDER_Traversal of Tree is....");
  POSTORDER_Traversal(Root);
  printf("\n");
  break;
case 7:
  printf("LEVELORDER_Traversal of Tree is....");
  LEVELORDER_Traversal(Root);
  break;
case 8:
  printf("which Node you want to delete :");
  scanf("%d",&n);
  DeleteNode(Root,n);
  break;
case 9:
  exit(0);
break;
default:
    printf("Enter valid choice between 1-10\n");
```

```
      break;
    }


  }
  return 0;
}
```

```
1) InsertNode
2) Hight of Tree
3) Search Data
4) INORDER_Traversal
5) PREORDER_Traversal
6) POSTORDER_Traversal
7) LEVELORDER_Traversal
8) DeleteNode
9) Exit

Enter Your choice :1
Enter Data :50
Enter Your choice :1
Enter Data :60
Enter Your choice :1
Enter Data :40
Enter Your choice :1
Enter Data :70
Enter Your choice :1
Enter Data :80
Enter Your choice :7
LEVELORDER_Traversal of Tree is....50 40 70 60 80
Enter Your choice :1
Enter Data :90
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 40 60 90
Enter Your choice :1
Enter Data :65
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 40 60 90 65
Enter Your choice :1
Enter Data :64
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 40 64 90 60 65
Enter Your choice :4
INORDER_Traversal of Tree is....40 50 60 64 65 70 80 90
Enter Your choice :5
PREORDER_Traversal of Tree is....70 50 40 64 60 65 80 90
Enter Your choice :6
```

```
POSTORDER_Traversal of Tree is....40 60 65 64 50 90 80 70
Enter Your choice :2
Hight Of tree is 3
Enter Your choice :8
which Node you want to delete :40
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 64 90 60 65
Enter Your choice :8
which Node you want to delete :90
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 64 60 65
Enter Your choice :4
INORDER_Traversal of Tree is....50 60 64 65 70 80
Enter Your choice :5
PREORDER_Traversal of Tree is....70 50 64 60 65 80
Enter Your choice :6
POSTORDER_Traversal of Tree is....60 65 64 50 80 70
Enter Your choice :8
which Node you want to delete :60
Enter Your choice :4
INORDER_Traversal of Tree is....50 64 65 70 80
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 80 64 65
Enter Your choice :8
which Node you want to delete :80
Enter Your choice :7
LEVELORDER_Traversal of Tree is....70 50 64 65
Enter Your choice :9

Press any key to continue . . .
```

D_F_S

```
#include<stdio.h>

int n;

int visited[20] = {0};


/* void DFS(vertex v){

    visited[vertex]  = true;
```

```
    for each w adjacent to v
        if (!visited[w])
            DFS(w);
*/

void DFS(int a[n][n],int v){
    int i;
    visited[v] = 1;
    printf("%d ",v);
    for(i=0;i<n;i++)
        if(a[v][i] == 1 && !visited[i])
            DFS(a,i);
}

int main(){
    int i,j,s;
    printf("Enter how many points are there in your graph:\n");
    scanf("%d",&n);
    int a[n][n];
    printf("Enter the adjacency list of graph\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);

    printf("Enter the source vertex\n");
    scanf("%d",&s);
```

```
    printf("The Depth First Search for your graph is \n");

    DFS(a,s);

    return 0;

}
```

```
Enter how many points are there in your graph:
4
Enter the adjacency list of graph
1 1 1 0
0 1 1 1
0 0 1 1
0 0 0 1
Enter the source vertex
1
The Depth First Search for your graph is
1 2 3
Process returned 0 (0x0)   execution time : 40.555 s
Press any key to continue.
```

B_F_S

```
#include<stdio.h>

int q[20],front=-1,rear=-1,a[20][20],vis[20]={0};

void insert(int item){
    if(rear==19)
    printf("QUEUE FULL");
    else{
        if(rear==-1){
            q[++rear]=item;
            front++;
        }
        else
        q[++rear]=item;
    }
}

int delete() {
    int k;
    if((front>rear)||(front==-1))
        return(0);
    else {
        k=q[front++];
        return(k);
    }
}
```

```
void bfs(int s,int n) {
    int p,i;
    insert(s);
    vis[s]=1;
    p=delete();
    if(p!=0)
    printf(" %d",p);
    while(p!=0){
        for(i=1;i<=n;i++)
            if((a[p][i]!=0)&&(vis[i]==0)){
                insert(i);
                vis[i]=1;
            }
        p=delete();
        if(p!=0)
            printf(" %d ",p);
    }
    for(i=1;i<=n;i++)
        if(vis[i]==0)
            bfs(i,n);
}

void main() {
    int n,i,s,j;
    printf("ENTER THE NUMBER VERTICES ");
    scanf("%d",&n);
```

```
    printf("ENter the adjancency matrix of your graph\n");
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("ENTER THE SOURCE VERTEX :");
    scanf("%d",&s);

    bfs(s,n);
}
```

```
ENTER THE NUMBER VERTICES 4
ENter the adjancency matrix of your graph
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1

ENTER THE SOURCE VERTEX :1
 1 3  2 4
Process returned 5 (0x5)   execution time : 29.719 s
Press any key to continue.
```