SAMPLE PROGRAMS

1. Write a program to implement push,pop, peek and display operations on stack.

```
#include<stdio.h>

void push(int a[], int*top, int x);
int pop(int a[], int *top);
void display(int a[], int top);
void peek(int a[], int top);

void main(){
    int a[100],x,i,choice;
    int top = -1;
    printf("\n*****************************");
    printf("\nChoose any operation- \n1.Push \n2.Pop \n3.Peek \n4.Display \n5.Exit");
    printf("\n*****************************");
    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1: printf("Enter the element to be pushed: ");
        scanf("%d", &x);
        push(a,&top,x);
            break;

        case 2: x = pop(a, &top);
        printf("The popped element is: %d\n", x);
            break;

        case 3: peek(a, top);
            break;

        case 4: display(a, top);
            break;
        default:
            break;
        }
    }while(choice!=5);
}

void push(int a[], int*top, int x){
    int n = 100;
    if(*top == n-1){
        printf("The stack is full!");
    }
    else{
        *top = *top + 1;
        a[*top] = x;
    }
}

int pop(int a[], int*top){
    int x;
    if(*top<0){
        printf("The stack is empty!");
        return 0;
    }
    else{
        x = a[*top];
        *top = *top - 1;
```

```
        return x;
    }
}

void peek(int a[], int top){
    printf("%d",a[top]);
}

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

2. Write a program to check well formedness of parentheses.

```
#include<stdio.h>
#include<conio.h>
char s[20];
int top=-1;
void push(char);
char pop();

//main program starts here...
int main()
{
char a[20],t;
int i,f=1;
printf("Enter the string\n");
scanf("%s",a);
for(i=0;i<strlen(a);i++)
{
    if(a[i]=='('||a[i]=='{'||a[i]=='[')
        push(a[i]);
    if(a[i]==')'||a[i]=='}'||a[i]==']')
    {
        if(top==-1)
            f=0;
        else
        {
            t=pop();
                if(a[i]==')'&&(t=='['||t=='{'))
                    f=0;
                if(a[i]=='}'&&(t=='('||t=='['))
                    f=0;
                if(a[i]==']'&&(t=='{'||t=='('))
                    f=0;
        }
    }
}
    if(top>=0)
        f=0;
    if(f==0)
        printf("Unbalanced\n");
```

```
    else
        printf("Balanced\n");
return 0;
}
//PUSH and POP operations...
void push(char a)
{
s[++top]=a;
}
char pop()
{
return s[top--];
}
```

3. Write a program to implement insert, delete and display operations on a linear queue.

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

#define MAX 10
int queue[MAX];
int front= -1, rear= -1;
void insert();
void delete();
void display();

int main(){
    int choice;
    printf("\n*****************************");
    printf("\nChoose any operation-\n1.Insert \n2.Delete \n3.Display \n4.Exit");
    printf("\n*****************************");
    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: insert();
            break;

            case 2: delete();
            break;

            case 3: printf("The elements of Queue are: ");
            display();
            break;

            case 4:break;
        }
    }while(choice!=4);
    return 0;
}

void insert(){
    int n;
    printf("Enter the element to be inserted: ");
```

```
    scanf("%d", &n);

    if(rear== MAX-1){
        printf("The queue is full!");
    }
    else if(front== -1 && rear== -1){
        front= rear= 0;
    }
    else{
        rear= rear + 1;
    }
    queue[rear]= n;
}

void delete(){
    int val;
    if(front== -1 || front> rear){
        printf("The queue is empty!\n");
    }
    else{
        val = queue[front];
        if(front==rear){
            front = rear = -1;
        }
        else{
            front++;
        }
        printf("Value Deleted!");
    }
}

void display(){
    int i;
    printf("\n");
    if(front == -1 || front > rear){
        printf("The queue is empty");
    }
    else{
        for(i=front; i<= rear; i++){
            printf("%d \t",queue[i]);
        }
    }
}
```

4. Write a program to implement insert, delete and display operations on a circular queue.

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

# define max 10
int queue[max];
int front = -1;
int rear = -1;
```

```
void enqueue(int elem);
int dequeue();
void display();

int main(){
    int choice, x;
    printf("\n***********************************");
    printf("\nChoose any operation-\n1.Insert \n2.Delete \n3.Display \n4.Exit");
    printf("\n***********************************");

    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: printf("Enter the element: ");
            scanf("%d", &x);
            enqueue(x);
            break;

            case 2: dequeue();
            break;

            case 3: display();
            break;

            case 4: printf("Exiting...");
            break;

            default: printf("Invalid Choice!");
        }
    }while(choice!=4);
    return 0;
}

void enqueue(int elem){
    if(front== -1 && rear== -1){
        front = 0;
        rear = 0;
        queue[rear] = elem;
    }
    else if((rear + 1) % max == front){
        printf("Queue is full!");
    }
    else{
        rear = (rear + 1) % max;
        queue[rear] = elem;
    }
}

int dequeue(){
    if((front == -1) && (rear == -1)){
        printf("\nQueue is Empty!");
    }
```

```
    else if(front == rear){
        printf("\nThe dequeued element is: %d", queue[front]);
        front = -1;
        rear = -1;
    }
    else{
        printf("\nThe dequeued element is: %d", queue[front]);
        front = (front + 1) % max;
    }
}

void display(){
    int  i = front;
    if(front == -1 && rear == -1){
        printf("Queue is Empty");
    }
    else{
        printf("Elements in the queue are: \n");
        while(i<=rear){
            printf("%d\t", queue[i]);
            i = (i + 1) % max;
        }
    }
}
```

5. Write a program to perform following operations on SLL

1. **create a LL(Insert)**
2. **reverse**
3. **Display/Traverse**

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

struct node{
    int data;
    struct node *next;
};

struct node *start = NULL;
struct node *prev = NULL;
struct node *curr = NULL;
struct node *next = NULL;

struct node *create(struct node *);
struct node *insertBeg(struct node *);
struct node *insertEnd(struct node *);
struct node *insertMid(struct node *);
struct node *reverse(struct node *);
struct node *display(struct node *);

int main(int argc, char *argv[]){
    int choice;
    printf("\n*****************************");
```

```
    printf("\nChoose an Operation");
    printf("\n1.Create a Singly Linked List");
    printf("\n2.Insert node at beginning");
    printf("\n3.Insert node in middle");
    printf("\n4.Insert node at end");
    printf("\n5.Reverse the Linked List");
    printf("\n6.Display the linked list");
    printf("\n7.Exit");
    printf("\n*****************************");

    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: start = create(start);
                    printf("\nLinked List Created.");
                    break;
            case 2: start = insertBeg(start);
                    break;
            case 3: start = insertMid(start);
                    break;
            case 4: start = insertEnd(start);
                    break;
            case 5: start = reverse(start);
                    break;
            case 6: start = display(start);
                    break;
        }
    }while(choice!=7);
}

struct node *create(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data(enter -1 to stop): ");
    scanf("%d", &num);
    while(num!=-1){
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        if(start==NULL){
            new_node->next = NULL;
            start = new_node;
        }
        else{
            ptr = start;
            while(ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        printf("\nEnter the data: ");
        scanf("%d", &num);
    }
```

```
    return start;
}

struct node *insertBeg(struct node *start){
    struct node *new_node;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    new_node-> next = start;
    start = new_node;
    return start;
}

struct node *insertMid(struct node *start){
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    printf("\nEnter the value after which it has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    ptr = start;
    preptr = ptr;
    while(preptr->data != val){
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}

struct node *insertEnd(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    new_node-> next = NULL;
    ptr = start;
    while(ptr-> next != NULL){
        ptr = ptr-> next;
    }
    ptr-> next = new_node;
    return start;
}

struct node *reverse(struct node *start){
    struct node *prev = NULL;
```

```
    struct node *current = start;
    struct node *next = NULL;
    while(current != NULL){
        next = current-> next;
        current->next = prev;
        prev = current;
        current = next;
    }
    start = prev;
}

struct node *display(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr != NULL){
        printf("\t%d", ptr-> data);
        ptr = ptr-> next;
    }
    return start;
}
```

6. Write a program to perform following operations on SLL

1. **create a LL(Insert)**
2. **Sort it**
3. **Display/Traverse**

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

struct node{
    int data;
    struct node *next;
};

struct node *start = NULL;
struct node *prev = NULL;
struct node *curr = NULL;
struct node *next = NULL;

struct node *create(struct node *);
struct node *insertBeg(struct node *);
struct node *insertEnd(struct node *);
struct node *insertMid(struct node *);
struct node *sort(struct node *);
struct node *display(struct node *);

int main(int argc, char *argv[]){
    int choice;
    printf("\n*****************************");
    printf("\nChoose an Operation");
    printf("\n1.Create a Singly Linked List");
    printf("\n2.Insert node at beginning");
    printf("\n3.Insert node in middle");
    printf("\n4.Insert node at end");
```

```
    printf("\n5.Sort the Linked List");
    printf("\n6.Display the linked list");
    printf("\n7.Exit");
    printf("\n*******************************");

    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: start = create(start);
                    printf("\nLinked List Created.");
                    break;
            case 2: start = insertBeg(start);
                    break;
            case 3: start = insertMid(start);
                    break;
            case 4: start = insertEnd(start);
                    break;
            case 5: start = sort(start);
                    break;
            case 6: start = display(start);
                    break;
        }
    }while(choice!=7);
}

struct node *create(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data(enter -1 to stop): ");
    scanf("%d", &num);
    while(num!=-1){
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        if(start==NULL){
            new_node->next = NULL;
            start = new_node;
        }
        else{
            ptr = start;
            while(ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        printf("\nEnter the data: ");
        scanf("%d", &num);
    }
    return start;
}

struct node *insertBeg(struct node *start){
    struct node *new_node;
```

```
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    new_node-> next = start;
    start = new_node;
    return start;
}

struct node *insertMid(struct node *start){
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    printf("\nEnter the value after which it has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    ptr = start;
    preptr = ptr;
    while(preptr->data != val){
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}

struct node *insertEnd(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    new_node-> next = NULL;
    ptr = start;
    while(ptr-> next != NULL){
        ptr = ptr-> next;
    }
    ptr-> next = new_node;
    return start;
}

struct node *sort(struct node *start){
    struct node *current = start;
    struct node *index = NULL;
    int temp;

    if(start == NULL){
        return start;
```

```
    }
    else{
        while(current != NULL){
            index = current-> next;
            while(index != NULL){
                if(current-> data > index-> data){
                    temp = current-> data;
                    current-> data = index-> data;
                    index-> data = temp;
                }
                index = index-> next;
            }
            current = current-> next;
        }

    }
    return start;
}

struct node *display(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr != NULL){
        printf("\t%d", ptr-> data);
        ptr = ptr-> next;
    }
    return start;
}
```

7. Write a program to perform following operations on SLL

1. **Create a LL(Insert)**
2. **Delete a node**
3. **Display/Traverse**

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

struct node{
    int data;
    struct node *next;
};

struct node *start = NULL;
struct node *prev = NULL;
struct node *curr = NULL;
struct node *next = NULL;

struct node *create(struct node *);
struct node *insertBeg(struct node *);
struct node *insertEnd(struct node *);
struct node *insertMid(struct node *);
struct node *deleteBeg(struct node *);
struct node *deleteMid(struct node *);
struct node *deleteEnd(struct node *);
```

```
struct node *display(struct node *);

int main(int argc, char *argv[]){
    int choice;
    printf("\n*******************************");
    printf("\nChoose an Operation");
    printf("\n1.Create a Singly Linked List");
    printf("\n2.Insert node at beginning");
    printf("\n3.Insert node in middle");
    printf("\n4.Insert node at end");
    printf("\n5.Delete node at beginning");
    printf("\n6.Delete node in middle");
    printf("\n7.Delete node at end");
    printf("\n8.Display the Linked List");
    printf("\n9.Exit");
    printf("\n*******************************");

    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: start = create(start);
                    printf("\nLinked List Created.");
                    break;
            case 2: start = insertBeg(start);
                    break;
            case 3: start = insertMid(start);
                    break;
            case 4: start = insertEnd(start);
                    break;
            case 5: start = deleteBeg(start);
                    break;
            case 6: start = deleteMid(start);
                    break;
            case 7: start = deleteEnd(start);
                    break;
            case 8: start = display(start);
                    break;
        }
    }while(choice!=9);
}

struct node *create(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data(enter -1 to stop): ");
    scanf("%d", &num);
    while(num!=-1){
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        if(start==NULL){
            new_node->next = NULL;
            start = new_node;
```

```
        }
        else{
            ptr = start;
            while(ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        printf("\nEnter the data: ");
        scanf("%d", &num);
    }
    return start;
}

struct node *insertBeg(struct node *start){
    struct node *new_node;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    new_node-> next = start;
    start = new_node;
    return start;
}

struct node *insertMid(struct node *start){
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    printf("\nEnter the value after which it has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    ptr = start;
    preptr = ptr;
    while(preptr->data != val){
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}

struct node *insertEnd(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
```

```
    new_node-> next = NULL;
    ptr = start;
    while(ptr-> next != NULL){
        ptr = ptr-> next;
    }
    ptr-> next = new_node;
    return start;
}

struct node *deleteBeg(struct node *start){
    struct node *ptr;
    ptr = start;
    start = start-> next;
    free(ptr);
    return start;
}

struct node *deleteMid(struct node *start){
    struct node *ptr, *preptr;
    int val;
    printf("\nEnter the value to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    if(ptr-> data == val){
        start = deleteBeg(start);
        return start;
    }
    else{
        while(ptr-> data != val){
            preptr = ptr;
            ptr = ptr-> next;
            if(ptr == NULL){
                printf("\nValue not present");
                return start;
            }
        }
        preptr-> next = ptr-> next;
        free(ptr);
        return start;
    }
}

struct node *deleteEnd(struct node *start){
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr-> next != NULL){
        preptr = ptr;
        ptr = ptr-> next;
    }
    preptr-> next = NULL;
    free(ptr);
    return start;
}
```

```
struct node *display(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr != NULL){
        printf("\t%d", ptr-> data);
        ptr = ptr-> next;
    }
    return start;
}
```

8. Write a program to implement stack/queue operations using SLL.

Stack:

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

struct node{
    int data;
    struct node *next;
};

struct node *top = NULL;

void push(int value);
int pop();
void display();

int main(){
    int choice, val;
    printf("\n**********************");
    printf("\n1.Push \n2.Pop \n3.Display \n4.Exit");
    printf("\n**********************");
    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1: printf("\nEnter the value to push: ");
                scanf("%d", &val);
                push(val);
                break;
        case 2: printf("\nPopped value is: %d", pop());
                break;
        case 3: display();
                break;
        }
    }while(choice != 4);
}

void push(int value){
    struct node *new_node;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = value;
```

```
    if(top == NULL){
        new_node-> next = NULL;
    }
    else{
        new_node-> next = top;
    }
    top = new_node;
}

int pop(){
    if(top == NULL){
        printf("\nStack is empty");
    }
    else{
        struct node *temp = top;
        int tempData = top-> data;
        top = top-> next;
        free(temp);
        return tempData;
    }
}

void display(){
    if(top == NULL){
        printf("\nStack is empty");
    }
    else{
        printf("The stack is \n");
        struct node *temp = top;
        while (temp->next != NULL){
            printf("%d\t", temp->data);
            temp = temp-> next;
        }
        printf("%d\n", temp->data);
    }
}
```

Queue:

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

struct node{
    int data;
    struct node *next;
};

struct node *front;
struct node *rear;

void insert();
void delete();
void display();
```

```
void main(){
    int choice;
    printf("\n*************************");
    printf("\nOperations are: ");
    printf("\n1.Insert element \n2.Delete element \n3.Display queue \n4.Exit");
    printf("\n*************************");
    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1: insert();
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
        }
    }while(choice != 4);
}

void insert(){
    struct node *ptr;
    int val;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL){
        printf("\nThe queue is full");
        return;
    }
    else{
        printf("\nEnter the value: ");
        scanf("%d", &val);
        ptr-> data = val;
        if(front == NULL){
            front = ptr;
            rear = ptr;
            front-> next = NULL;
            rear-> next = NULL;
        }
        else{
            rear-> next = ptr;
            rear = ptr;
            rear-> next = NULL;
        }
    }
}

void delete(){
    struct node *ptr;
    if(front == NULL){
        printf("\nQueue is empty");
        return;
    }
```

```
    else{
        printf("\nElement is deleted");
        ptr = front;
        front = front-> next;
        free(ptr);
    }
}

void display(){
    struct node *ptr;
    ptr = front;
    if(front == NULL){
        printf("\nQueue is empty");
    }
    else{
        printf("\nPrinting queue...\n");
        while(ptr != NULL){
            printf("%d\t", ptr-> data);
            ptr = ptr-> next;
        }
    }
}
```

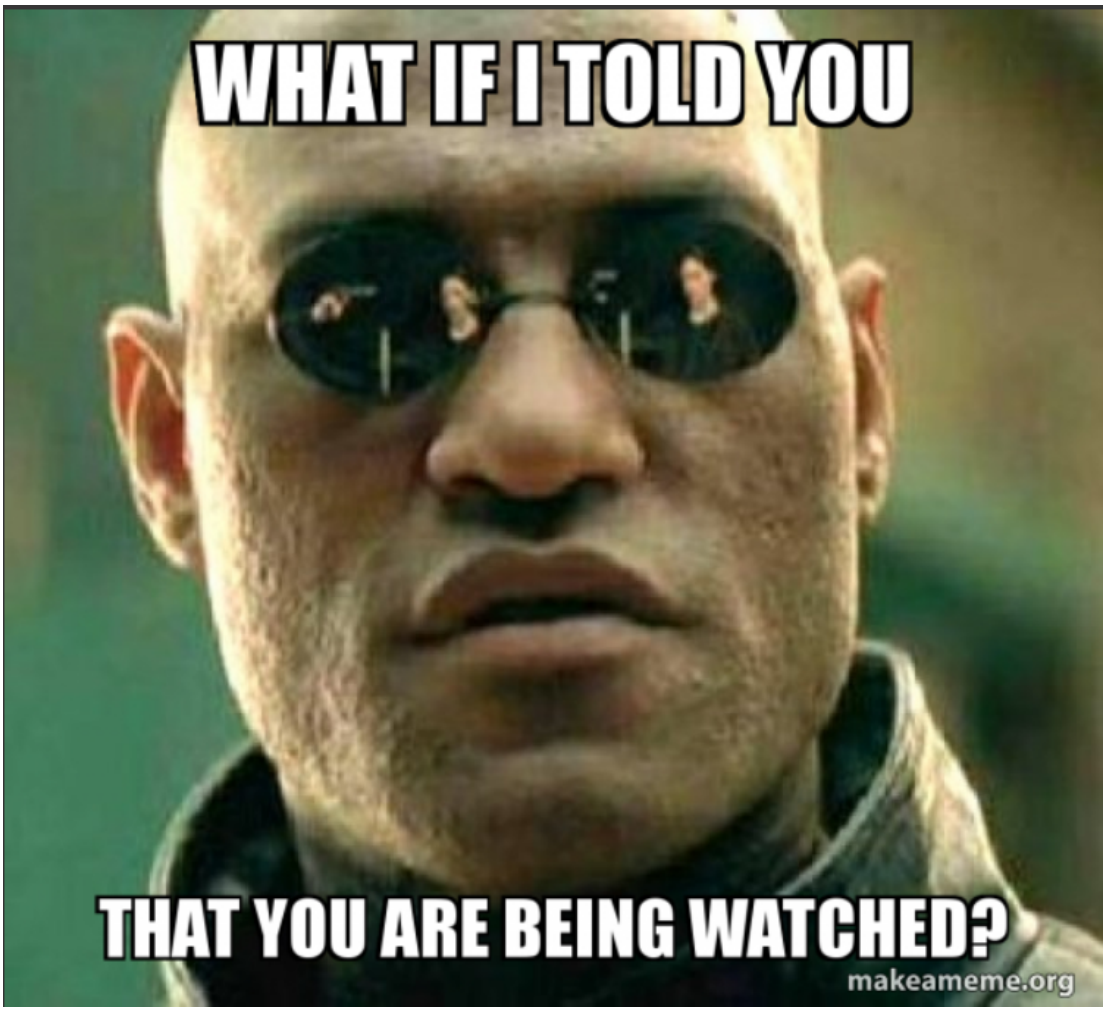
9. Write a program to evaluate Prefix / Postfix expression.

Postfix:

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

typedef struct{
    int a[100];
    int top;
}stack;

void push(stack *s, int x);
int pop(stack *s);
int operation(char op, int p1, int p2);
int eval(char post[]);

void main(){
    char postfix[20];
    printf("\nEnter a postfix expression: ");
    gets(postfix);
    printf("The result is: %d", eval(postfix));
}

void push(stack *s, int x){
    if(s-> top == 20){
        printf("Stack is full");
    }
    else{
        s-> a[++s-> top] = x;
    }
}
```

```
int pop(stack *s){
    int x;
    if(s-> top < 0){
        printf("Empty!");
    }
    else{
        x = s->a[s-> top--];
        return x;
    }
}

int operation(char op, int p1, int p2){
    switch(op){
        case '+': return p1+p2;
        case '-': return p1-p2;
        case '*': return p1*p2;
        case '/': return p1/p2;
    }
}

int eval(char post[]){
    stack s;
    int i,p1,p2,p;
    s.top = -1;
    for(i=0; post[i] != '\0'; ++i){
        if(isdigit(post[i])){
            push(&s, post[i] - '0');
        }
        else{
            p2 = pop(&s);
            p1 = pop(&s);
            p = operation(post[i], p1, p2);
            push(&s, p);
        }
    }
    return pop(&s);
}
```

Prefix: bhai kay ghabarlo beyyyyyy 😂😂
Ha prefix vala nahi jamat aahe :`(
Mala tr postfix wala pn samjat nahi ahe 😭😭sed te mla pan nahi kallay jast
🫠🫂



10. Write a program to perform following operations on circular SLL

1. **Insert a node**
2. **Delete a node**
3. **Display/Traverse**

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

struct node{
    int data;
    struct node *next;
};

struct node *start = NULL;
struct node *createCll(struct node *);
struct node *insertBeg(struct node *);
struct node *insertMid(struct node *);
struct node *insertEnd(struct node *);
struct node *deleteBeg(struct node *);
struct node *deleteEnd(struct node *);
struct node *deleteMid(struct node *);
struct node *display(struct node *);
// struct node *deleteList(struct node *);

int main(){
```

```
    int choice;
    printf("\n*********************************");
    printf("\n1.Create Circular LL");
    printf("\n2.Insert in beginning");
    printf("\n3.Insert in middle");
    printf("\n4.Insert at end");
    printf("\n5.Delete the beginning");
    printf("\n6.Delete in middle");
    printf("\n7.Delete at end");
    printf("\n8.Display the Linked List");
    printf("\n9.Exit");
    printf("\n*********************************");

    do{
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1: start = createCll(start);
                    break;
            case 2: start = insertBeg(start);
                    break;
            case 3: start = insertMid(start);
                    break;
            case 4: start = insertEnd(start);
                    break;
            case 5: start = deleteBeg(start);
                    break;
            case 6: start = deleteMid(start);
                    break;
            case 7: start = deleteEnd(start);
                    break;
            case 8: start = display(start);
                    break;
        }
    }while(choice != 9);
}

struct node *createCll(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data to insert(-1 to end): ");
    scanf("%d", &num);
    while(num != -1){
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node-> data = num;
        if(start == NULL){
            new_node-> next = new_node;
            start = new_node;
        }
        else{
            ptr = start;
            while(ptr-> next != start){
```

```
                ptr = ptr-> next;
            }
            ptr-> next = new_node;
            new_node-> next = start;
        }
        printf("\nEnter the data: ");
        scanf("%d", &num);
    }
    return start;
}

struct node *insertBeg(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->next != start){
        ptr = ptr->next;
    }
    ptr->next = new_node;
    new_node->next = start;
    start = new_node;
    return start;
}

struct node *insertMid(struct node *start){
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\nEnter the data: ");
    scanf("%d", &num);
    printf("\nEnter the value after which it has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    ptr = start;
    preptr = ptr;
    while(preptr->data != val){
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}

struct node *insertEnd(struct node *start){
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data: ");
    scanf("%d", &num);
```

```
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node-> data = num;
    ptr = start;
    while(ptr-> next != start){
        ptr = ptr-> next;
    }
    ptr-> next = new_node;
    new_node-> next = start;
    return start;
}

struct node *deleteBeg(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr-> next != start){
        ptr = ptr-> next;
    }
    ptr-> next = start-> next;
    free(start);
    start = ptr-> next;
    return start;
}

struct node *deleteMid(struct node *start){
    struct node *ptr, *preptr;
    int val;
    printf("\nEnter the value after which node has to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    preptr = ptr;
    while(preptr-> data != val){
        preptr = ptr;
        ptr = ptr-> next;
    }
    preptr-> next = ptr-> next;
    if(ptr == start){
        start = preptr-> next;
    }
    free(ptr);
    return start;
}

struct node *deleteEnd(struct node *start){
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr-> next != start){
        preptr = ptr;
        ptr = ptr-> next;
    }
    preptr-> next = ptr-> next;
    free(ptr);
    return start;
}
```

```
struct node *display(struct node *start){
    struct node *ptr;
    ptr = start;
    while(ptr-> next != start){
        printf("\t%d", ptr-> data);
        ptr = ptr-> next;
    }
    printf("\t%d", ptr-> data);
    return start;
}
```

12. Write a program to perform following operations on BST

1. Insert a node

2. Delete a node

3. Inorder/preorder/postorder traversal

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node {
int data;
struct node *left;
struct node *right;
};

struct node *tree;
```

```
void create_tree (struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorederTraversal(struct node *);
void postorderTraversal(struct node *);
struct node *deleteElement(struct node *,int);

int main (){
 int option, val;
 struct node *ptr;
 create_tree(tree);
 do {
    printf("\n ----------MENU--------- \n");
    printf("\n 1. Insert Element");
    printf("\n 2. Preorder Traversal");
    printf("\n 3. Inorder Traversal");
    printf("\n 4. Postoder Traversal");
    printf("\n 5. Delete an element");
    printf("\n 6. Exit");
    printf("\n\n Enter your Option ");
    scanf("%d", &option);
    switch(option)
    {
        case 1:
            printf("\n Enter the value of new element ");
            scanf("%d", &val);
            tree = insertElement(tree, val);
            break;
        case 2:
            printf("\n The elements of the tree are : ");
            preorderTraversal(tree);
            break;
        case 3:
            printf("\n The elements of the tree are : ");
            inorderTraversal(tree);
            break;
        case 4:
            printf("\n The elements of the tree are : ");
            postorderTraversal(tree);
            break;
        case 5:
            printf("\n Enter the elements to be deleted: ");
            scanf("%d",val);
            tree = deleteElement(tree,val);
            break;

    }

 }while (option!=6);
```

```
  getch();
  return 0;

}
void create_tree (struct node *tree)
{
    tree = NULL;
}
struct node *insertElement(struct node *tree, int val)
{
   struct node *ptr, *nodeptr, *parentptr;
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr-> data = val;
   ptr-> left =NULL;
   ptr-> right=NULL;
   if (tree == NULL)
   {
       tree= ptr;
       tree->left = NULL;
       tree->right =NULL;
   }
   else
   {
       parentptr=NULL;
       nodeptr=tree;
       while(nodeptr!=NULL)
       {
           parentptr = nodeptr;
           if (val<nodeptr ->data)
            nodeptr = nodeptr->left;
           else
            nodeptr = nodeptr->right;
       }
       if (val<parentptr -> data)
        parentptr -> left = ptr;
       else
        parentptr -> right= ptr;

   }
   return tree;
}

void preorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        printf("%d\t",tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
```

```
    }
}
void inorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        inorderTraversal(tree->left);
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t",tree->data);
    }
}
struct node *deleteElement(struct node *tree,int val)
{
   struct node *cur, *parent, *suc, *psuc, *ptr;
   if (tree ->left==NULL)
   {
       printf("The tree is Empty");
       return(tree);
   }
   parent = tree;
   cur = tree -> left;
   while (cur!=NULL && val!= cur->data)
   {
       parent = cur;
       cur = (val<cur->data)?cur->left:cur->right;
   }
   if (cur == NULL)
   {
       printf("\n The value to be deleted is not present in the tree");
       return(tree);
   }
   if (cur->left== NULL)
    ptr = cur->right;
   else if (cur->right == NULL)
    ptr = cur->left;
   else
   {
      psuc = cur;
      cur = cur->left;
      while(suc->left!=NULL)
```

```
      {
          psuc = suc;
          suc = suc ->left;
      }
      if(cur==psuc)
      {
          suc ->left = cur ->right;
      }
      else
      {
          suc->left= cur->left;
          psuc -> left = suc ->right;
          suc ->right = cur ->right;
      }
      ptr = suc;
   }
   if (parent->left == cur)
    parent->left=ptr;
   else
    parent ->right=ptr;
   free(cur);
   return tree;
}
```

13. Write a program to perform following operations on BST

1. Insert a node

2. Find Height of tree

3. Inorder/preorder/postorder traversal

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node {
int data;
struct node *left;
struct node *right;
};

struct node *tree;
void create_tree (struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorederTraversal(struct node *);
void postorderTraversal(struct node *);
int Height(struct node *);
int main (){
 int option, val;
 struct node *ptr;
 create_tree(tree);
 do {
    printf("\n ----------MENU--------- \n");
```

```
    printf("\n 1. Insert Element");
    printf("\n 2. Preorder Traversal");
    printf("\n 3. Inorder Traversal");
    printf("\n 4. Postoder Traversal");
    printf("\n 5. Height of a Tree");
    printf("\n 6. Exit");
    printf("\n\n Enter your Option ");
    scanf("%d", &option);
    switch(option)
    {
        case 1:
            printf("\n Enter the value of new element ");
            scanf("%d", &val);
            tree = insertElement(tree, val);
            break;
        case 2:
            printf("\n The elements of the tree are : ");
            preorderTraversal(tree);
            break;
        case 3:
            printf("\n The elements of the tree are : ");
            inorderTraversal(tree);
            break;
        case 4:
            printf("\n The elements of the tree are : ");
            postorderTraversal(tree);
            break;
        case 5:
            printf("\n The height of the tree = %d",Height(tree));
            break;

    }
 } while (option!=6);
   getch();
   return 0;

}
void create_tree (struct node *tree)
{
    tree = NULL;
}
struct node *insertElement(struct node *tree, int val)
{
   struct node *ptr, *nodeptr, *parentptr;
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr-> data = val;
   ptr-> left =NULL;
   ptr-> right=NULL;
   if (tree == NULL)
   {
       tree= ptr;
       tree->left = NULL;
       tree->right =NULL;
```

```
   }
   else
   {
       parentptr=NULL;
       nodeptr=tree;
       while(nodeptr!=NULL)
       {
           parentptr = nodeptr;
           if (val<nodeptr ->data)
            nodeptr = nodeptr->left;
           else
            nodeptr = nodeptr->right;
       }
       if (val<parentptr -> data)
        parentptr -> left = ptr;
       else
        parentptr -> right= ptr;

   }
   return tree;
}

void preorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        printf("%d\t",tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}
void inorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        inorderTraversal(tree->left);
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t",tree->data);
    }
}
int Height(struct node *tree)
{
int leftheight, rightheight;
if(tree==NULL)
```

```
return 0;
else
{
leftheight = Height(tree->left);
rightheight = Height(tree->right);
if(leftheight > rightheight)
return (leftheight + 1);
else
return (rightheight + 1);
}
}
```

14. Write a program to perform following operations on BST

1. Insert a node

2. Find total no. of nodes in a tree.

3. Inorder/preorder/postorder traversal

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node {
int data;
struct node *left;
struct node *right;
};

struct node *tree;
void create_tree (struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
void postorderTraversal(struct node *);
int totalNodes(struct node *);
int main (){
 int option, val;
 struct node *ptr;
 create_tree(tree);
 do {
    printf("\n ----------MENU--------- \n");
    printf("\n 1. Insert Element");
    printf("\n 2. Preorder Traversal");
    printf("\n 3. Inorder Traversal");
    printf("\n 4. Postorder Traversal");
    printf("\n 5. Total no. of Nodes");
    printf("\n 6. Exit");
    printf("\n\n Enter your Option ");
    scanf("%d", &option);
    switch(option)
    {
```

```
    case 1:
            printf("\n Enter the value of new element ");
            scanf("%d", &val);
            tree = insertElement(tree, val);
            break;
        case 2:
            printf("\n The elements of the tree are : ");
            preorderTraversal(tree);
            break;
        case 3:
            printf("\n The elements of the tree are : ");
            inorderTraversal(tree);
            break;
        case 4:
            printf("\n The elements of the tree are : ");
            postorderTraversal(tree);
            break;
        case 5:
            printf("\n Total no. of nodes = %d", totalNodes(tree));
            break;


    }
 } while (option!=6);
   getch();
   return 0;
   }
void create_tree (struct node *tree)
{
    tree = NULL;
}
struct node *insertElement(struct node *tree, int val)
{
   struct node *ptr, *nodeptr, *parentptr;
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr-> data = val;
   ptr-> left =NULL;
   ptr-> right=NULL;
   if (tree == NULL)
   {
       tree= ptr;
       tree->left = NULL;
       tree->right =NULL;
   }
   else
   {
       parentptr=NULL;
       nodeptr=tree;
       while(nodeptr!=NULL)
```

```
        {
            parentptr = nodeptr;
            if (val<nodeptr ->data)
             nodeptr = nodeptr->left;
            else
             nodeptr = nodeptr->right;
        }
        if (val<parentptr -> data)
         parentptr -> left = ptr;
        else
         parentptr -> right= ptr;

    }
    return tree;
}

void preorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        printf("%d\t",tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}
void inorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        inorderTraversal(tree->left);
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t",tree->data);
    }
}
int totalNodes(struct node *tree)
{
if(tree==NULL)
return 0;
else
return(totalNodes(tree->left) + totalNodes(tree->right) + 1);
```

```
}
```

15. Write a program to perform following operations on BST

1. Insert a node

2. Find mirror image

3. Inorder/preorder/postorder traversal

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node {
int data;
struct node *left;
struct node *right;
};

struct node *tree;
void create_tree (struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
void postorderTraversal(struct node *);
struct node *mirrorImage(struct node *);
int main (){
 int option, val;
 struct node *ptr;
 create_tree(tree);
 do {
    printf("\n ----------MENU--------- \n");
    printf("\n 1. Insert Element");
    printf("\n 2. Preorder Traversal");
    printf("\n 3. Inorder Traversal");
    printf("\n 4. Postorder Traversal");
    printf("\n 5. Find mirror Image of a Tree");
    printf("\n 6. Exit");
    printf("\n\n Enter your Option ");
    scanf("%d", &option);
    switch(option)
    {
    case 1:
            printf("\n Enter the value of new element ");
            scanf("%d", &val);
            tree = insertElement(tree, val);
            break;
        case 2:
            printf("\n The elements of the tree are : ");
            preorderTraversal(tree);
```

```
            break;
         case 3:
            printf("\n The elements of the tree are : ");
            inorderTraversal(tree);
            break;
         case 4:
            printf("\n The elements of the tree are : ");
            postorderTraversal(tree);
            break;
         case 5:
            tree = mirrorImage(tree);
            break;



     }
  } while (option!=6);
    getch();
    return 0;
    }
void create_tree (struct node *tree)
{
     tree = NULL;
}
struct node *insertElement(struct node *tree, int val)
{
    struct node *ptr, *nodeptr, *parentptr;
    ptr = (struct node*)malloc(sizeof(struct node));
    ptr-> data = val;
    ptr-> left =NULL;
    ptr-> right=NULL;
    if (tree == NULL)
    {
        tree= ptr;
        tree->left = NULL;
        tree->right =NULL;
    }
    else
    {
        parentptr=NULL;
        nodeptr=tree;
        while(nodeptr!=NULL)
        {
            parentptr = nodeptr;
            if (val<nodeptr ->data)
             nodeptr = nodeptr->left;
            else
             nodeptr = nodeptr->right;
        }
```

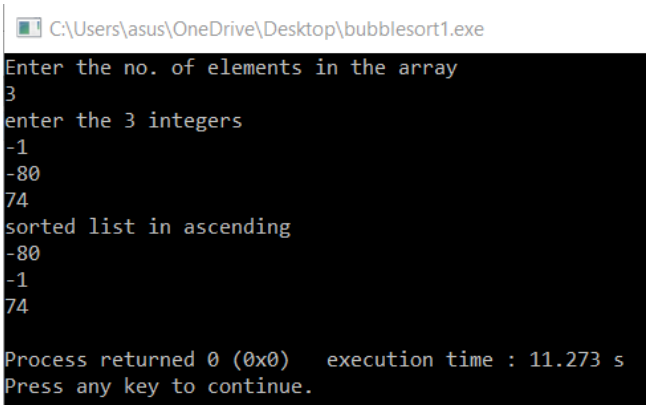
```
       if (val<parentptr -> data)
        parentptr -> left = ptr;
       else
        parentptr -> right= ptr;

   }
   return tree;
}

void preorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        printf("%d\t",tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}
void inorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        inorderTraversal(tree->left);
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree!=NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t",tree->data);
    }
}
struct node *mirrorImage(struct node *tree)
{
struct node *ptr;
if(tree!=NULL)
{
mirrorImage(tree->left);
mirrorImage(tree->right);
ptr=tree->left;
ptr->left = ptr->right;
tree->right = ptr;
}
}
```

17. Write a program to implement a bubble and selection sort algorithm.

BUBBLE SORT

```
#include<stdio.h>
int main(){
//initializing the array
int arr[100], i, size, d,temp =0;
printf("Enter the no. of elements in the array\n");
scanf("%d",&size);
printf("enter the %d integers\n",size);
//inputting the values
for (d=0;d<size;d++){
    scanf("%d", &arr[d]);
}
//processing the data
```

```
// d is the steps
// i is the iterations
for (int d = 0; d < size -1; d++){
    for (int i =0; i<size-d-1;i++){
        //comparing adjacent elements
        if(arr[i]>arr[i+1]){
            temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;

        }
    }
}
printf("sorted list in ascending\n");
for(d=0;d<size;++d){
    printf("%d\n", arr[d]);}
    return 0;
}
```



SELECTION SORT

```
#include<stdio.h>
int main(){
int arr[100], c, small, size, d,temp;

printf("Enter the size of the array ");
scanf("%d",&size);

printf("Enter %d elements\n",size);
for(c=0;c<size;c++){
    scanf("%d",&arr[c]);
}
for (c = 0;c<size-1;c++){
    small = c;
    for (d=c+1;d<size;d++){
            if (arr[small]>arr[d])
            small= d;}
            if (small != c){
                temp = arr[c];
                arr[c] = arr[small];
                arr[small]= temp;
            }
```

```
    }


printf("Sorted elements in ascending order are\n");
for(c=0;c<size;c++)
    printf("%d\n",arr[c]);
return 0;
}
```