

LAB FILE
DATA STRUCTURES USING C
(CSIT124)

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



Submitted to:
Dr Nisha Pahal
CSE Department, ASET

Submitted by:
Umang Sehgal
A2305219288
B.tech. C.S.E.
3CSE-4Y

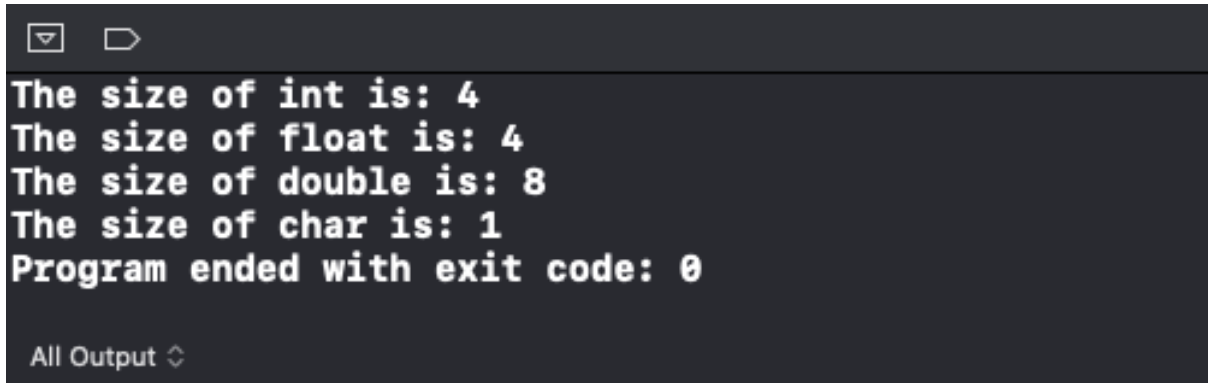
**AMITY SCHOOL OF ENGINEERING AND
TECHNOLOGY**
AMITY UNIVERSITY UTTAR PRADESH
NOIDA -201301

Q1. WAP to find the size of int, float, double, char.

Code:

```
#include <stdio.h>
int main() {
    printf("The size of int is: %ld\n",sizeof(int));
    printf("The size of float is: %ld\n",sizeof(float));
    printf("The size of double is: %ld\n",sizeof(double));
    printf("The size of char is: %ld\n",sizeof(char));
    return 0;
}
```

Output:



```
The size of int is: 4
The size of float is: 4
The size of double is: 8
The size of char is: 1
Program ended with exit code: 0

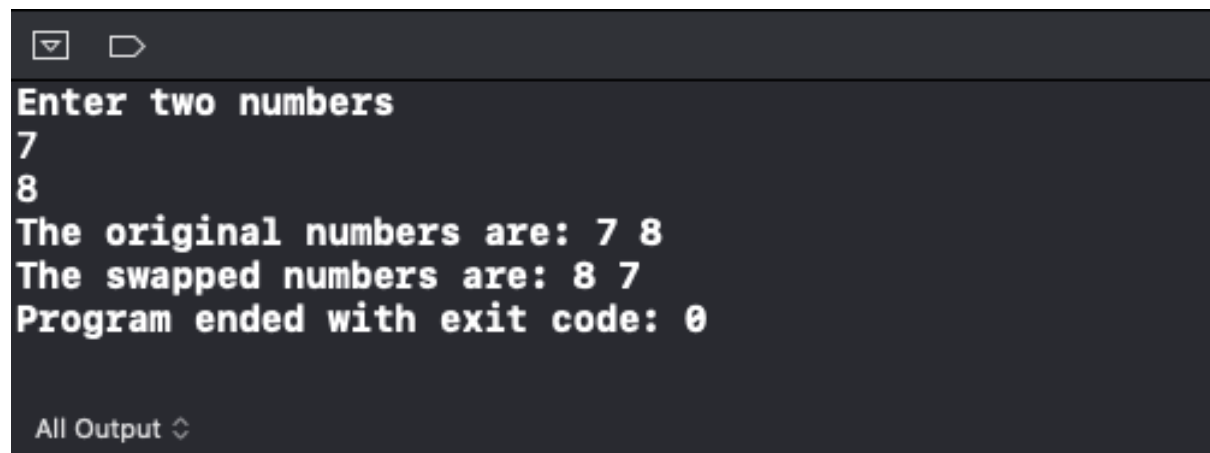
All Output ^
```

Q2. WAP to swap two numbers.

Code:

```
#include <stdio.h>
int main() {
    int num_1, num_2;
    printf("Enter two numbers \n");
    scanf("%d %d",&num_1, &num_2);
    printf("The original numbers are: %d %d\n", num_1, num_2);
    num_1 = num_1 + num_2;
    num_2 = num_1 - num_2;
    num_1 = num_1 - num_2;
    printf("The swapped numbers are: %d %d\n", num_1, num_2);
    return 0;
}
```

Output:



```
Enter two numbers
7
8
The original numbers are: 7 8
The swapped numbers are: 8 7
Program ended with exit code: 0
```

All Output ↕

Q3. WAP to check whether the given character is a vowel or consonant.

Code:

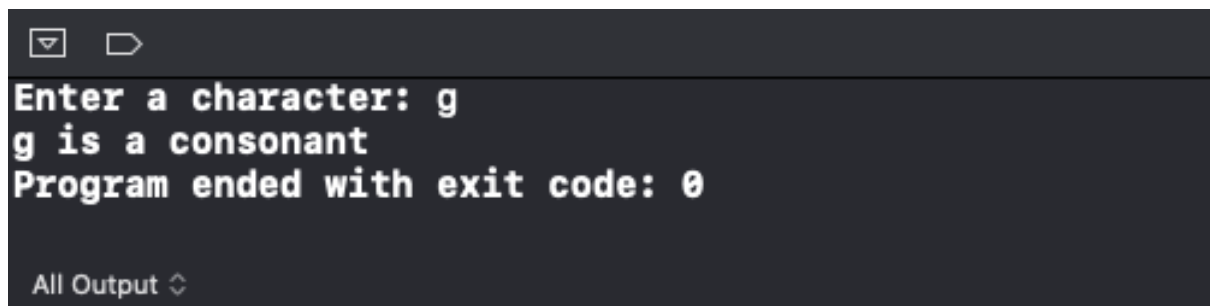
```
#include <stdio.h>
#include <ctype.h>
int main(){
    char ch;
    printf("Enter a character: ");
    scanf("%c",&ch);
    ch = tolower(ch);

    if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
        printf("%c is an vowel\n",ch);

    else if(ch>='a' && ch<='z')
        printf("%c is a consonant\n",ch);

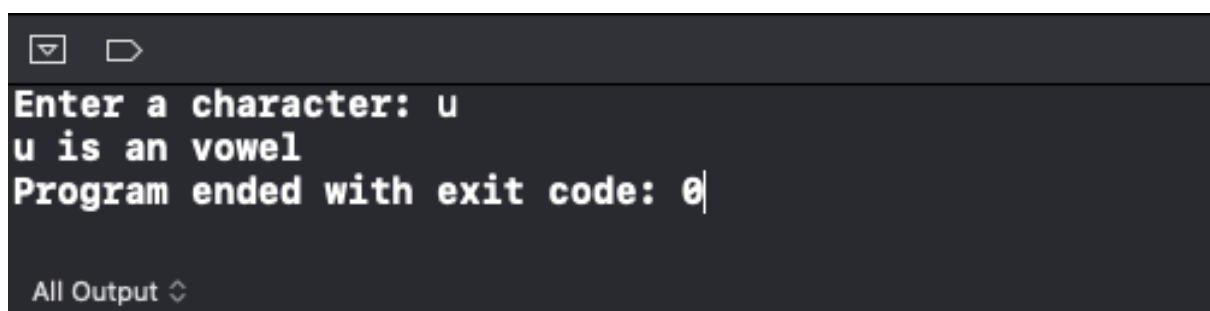
    else
        printf("%c is not an alphabet\n",ch);
    return 0;
}
```

Output:

A terminal window with a dark background. At the top, there are two icons: a dropdown arrow and a rectangle. The text in the terminal reads: "Enter a character: g", "g is a consonant", and "Program ended with exit code: 0". At the bottom, there is a label "All Output" followed by a double-headed arrow icon.

```
Enter a character: g
g is a consonant
Program ended with exit code: 0

All Output ⇅
```

A terminal window with a dark background. At the top, there are two icons: a dropdown arrow and a rectangle. The text in the terminal reads: "Enter a character: u", "u is an vowel", and "Program ended with exit code: 0". At the bottom, there is a label "All Output" followed by a double-headed arrow icon.

```
Enter a character: u
u is an vowel
Program ended with exit code: 0

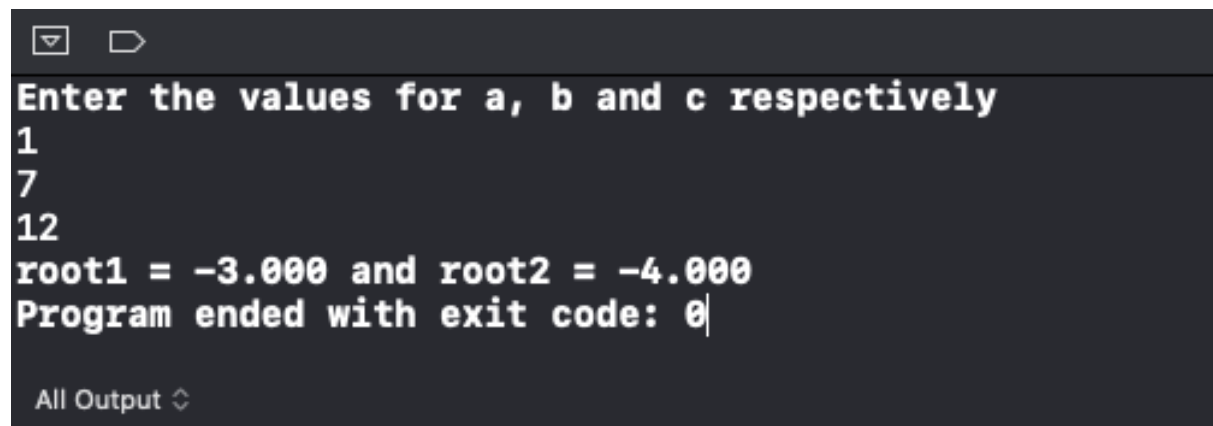
All Output ⇅
```

Q4. WAP to find the roots of a quadratic equation.

Code:

```
#include <stdio.h>
#include <math.h>
int main(){
    int a, b, c;
    float disct, root1, root2;
    printf("Enter the values for a, b and c respectively \n");
    scanf("%d %d %d", &a, &b, &c);
    disct = b * b - 4 * a * c;      //assuming the case of discriminant greater than 0.
    root1 = (-b + sqrt(disct)) / (2 * a);
    root2 = (-b - sqrt(disct)) / (2 * a);
    printf("root1 = %.3f and root2 = %.3f\n", root1, root2);
}
```

Output:



```
Enter the values for a, b and c respectively
1
7
12
root1 = -3.000 and root2 = -4.000
Program ended with exit code: 0
```

All Output ↕

Q5. WAP to check whether the character is an alphabet or not.

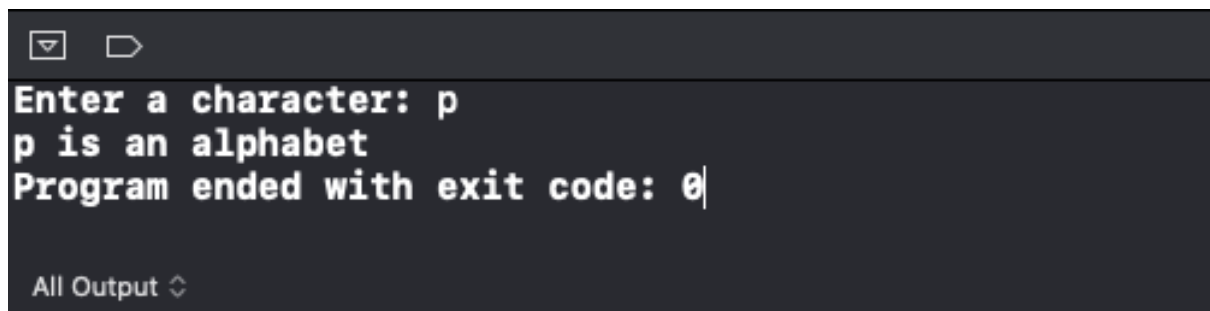
Code:

```
#include <stdio.h>
#include <ctype.h>
int main(){
    char ch, original;
    printf("Enter a character: ");
    scanf("%c",&original);
    ch = tolower(original);

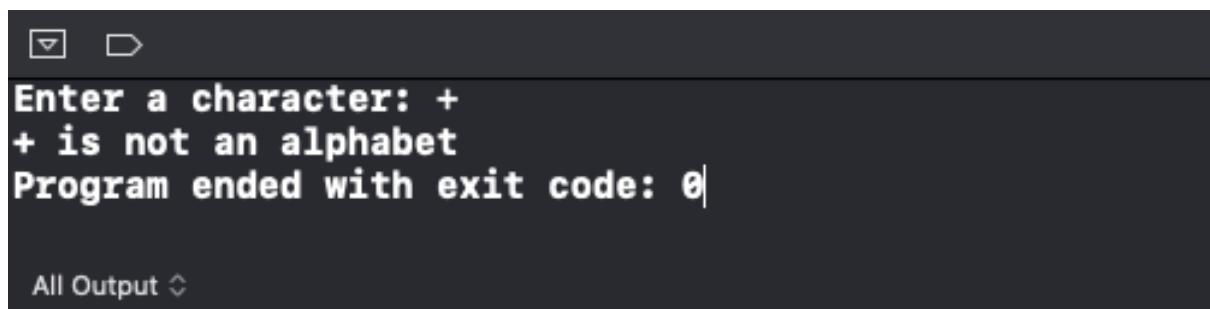
    if(ch>='a' && ch<='z')
        printf("%c is an alphabet\n",original);

    else
        printf("%c is not an alphabet\n",original);
    return 0;
}
```

Output:

A terminal window with a dark background and light gray text. The output shows the program prompting for a character, receiving 'p', and confirming it is an alphabet. The exit code is 0.

```
Enter a character: p
p is an alphabet
Program ended with exit code: 0
```

A terminal window with a dark background and light gray text. The output shows the program prompting for a character, receiving '+', and confirming it is not an alphabet. The exit code is 0.

```
Enter a character: +
+ is not an alphabet
Program ended with exit code: 0
```

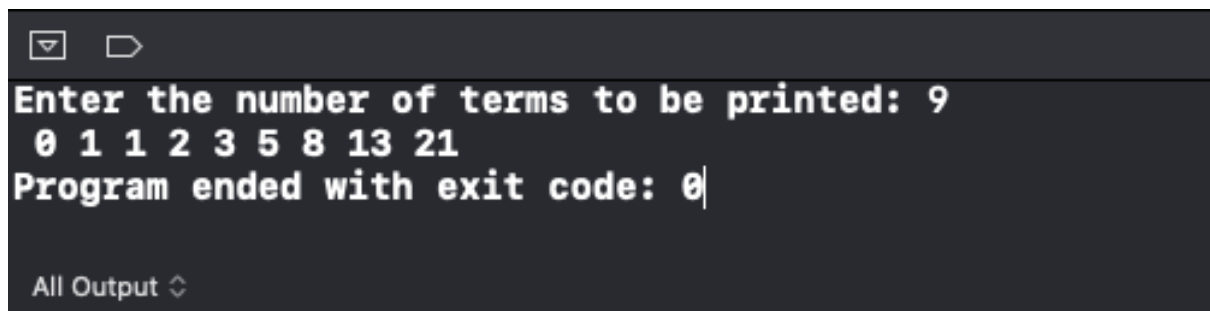
Q6. WAP to display Fibonacci series.

Code:


```
#include <stdio.h>
int main(){
    int first = 0, second = 1, sum=0, terms;
    printf("Enter the number of terms to be printed \n");
    scanf("%d",&terms);

    for(int i=0; i<terms; i++)
    {
        printf(" %d",first);
        sum = first + second;
        first = second;
        second = sum;
    }
    return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The window has a title bar with a dropdown arrow and a close button. The text inside the terminal is white and shows the program's execution: it prompts for the number of terms, the user enters 9, the program prints the Fibonacci sequence (0 1 1 2 3 5 8 13 21), and finally displays 'Program ended with exit code: 0'. At the bottom of the terminal, there is a label 'All Output' followed by a small expand/collapse icon.

```
Enter the number of terms to be printed: 9
0 1 1 2 3 5 8 13 21
Program ended with exit code: 0
```

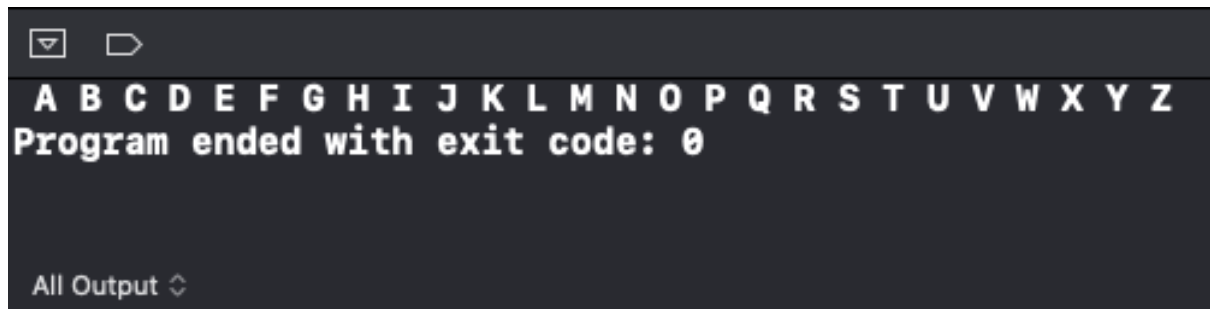
All Output 

Q7. WAP to display characters from A to Z using loop.

Code:

```
#include <stdio.h>
#include <ctype.h>
int main(){
    char ch = 'A';
    for(ch;ch<='Z';ch++)
        printf(" %c",ch);
    printf("\n");
    return 0;
}
```

Output:

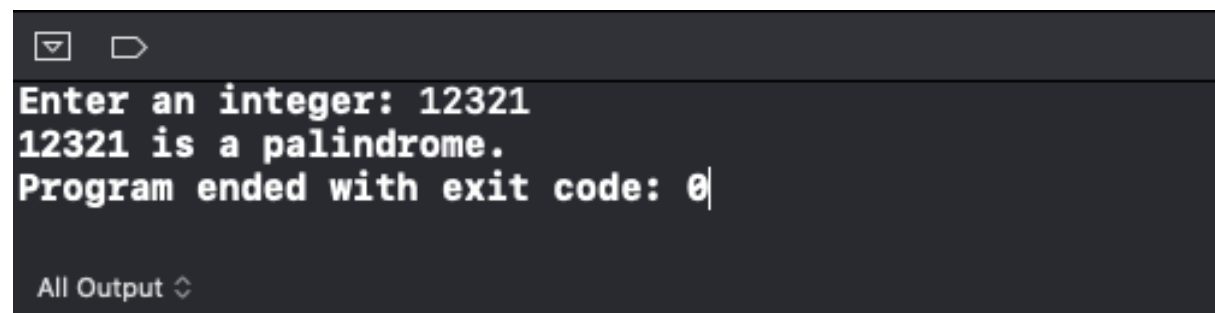
A screenshot of a terminal window with a dark background. The top bar shows a dropdown menu icon and a right-pointing arrow. The main area displays the output of the program: the letters 'A' through 'Z' in a monospaced font, followed by the text 'Program ended with exit code: 0'. At the bottom, there is a label 'All Output' with a small expand/collapse icon.

Q8. WAP to check whether a number is palindrome or not.

Code:

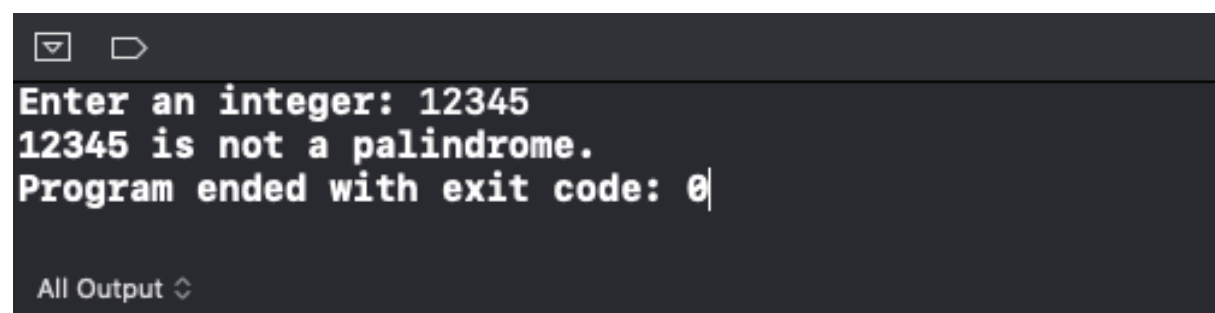
```
#include <stdio.h>
int main(){
    int num, reversed = 0, original, remainder;
    printf("Enter an integer: ");
    scanf("%d", &num);
    original = num;
    while (num > 0) {
        remainder = num % 10;
        reversed = reversed * 10 + remainder;
        num /= 10;
    }
    if (original == reversed)
        printf("%d is a palindrome.", original);
    else
        printf("%d is not a palindrome.", original);
}
```

Output:

A terminal window with a dark background and light gray text. The output shows the program prompting for an integer, receiving 12321, and confirming it is a palindrome. The window title bar includes a dropdown arrow and a close button. At the bottom, there is a link labeled 'All Output' with a double-headed arrow icon.

```
Enter an integer: 12321
12321 is a palindrome.
Program ended with exit code: 0
```

All Output ↕

A terminal window with a dark background and light gray text. The output shows the program prompting for an integer, receiving 12345, and confirming it is not a palindrome. The window title bar includes a dropdown arrow and a close button. At the bottom, there is a link labeled 'All Output' with a double-headed arrow icon.

```
Enter an integer: 12345
12345 is not a palindrome.
Program ended with exit code: 0
```

All Output ↕

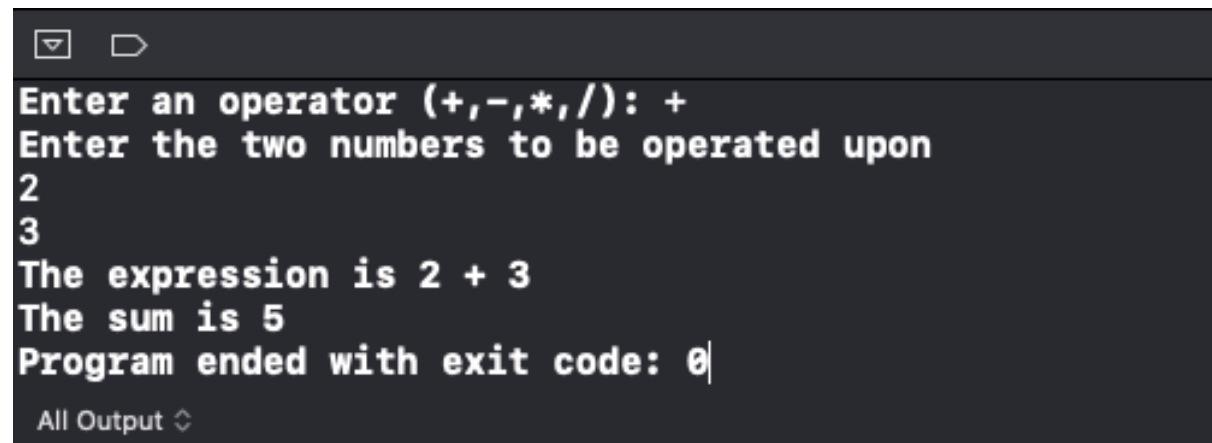
Q9. WAP to make a simple calculator using switch case.

Code:

```
#include <stdio.h>
int main(){
    char operator;
    int first,second;
    printf("Enter an operator (+,-,*,/) \n");
    scanf("%c",&operator);
    printf("Enter the two numbers to be operated upon \n");
    scanf("%d %d",&first,&second);

    printf("The expression is %d %c %d \n",first,operator,second);
    switch(operator){
        case '+': printf("The sum is %d",first+second);
                    break;
        case '-': printf("The difference is %d",first-second);
                    break;
        case '*': printf("The multiplication is %d",first*second);
                    break;
        case '/': printf("The division is %d",first/second);
                    break;
        default : printf("Please enter a valid operator.");
    }
}
```

Output:



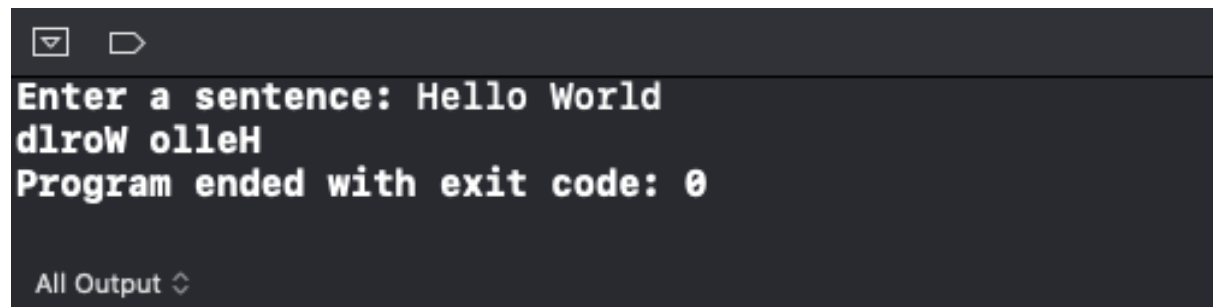
```
Enter an operator (+,-,*,/): +
Enter the two numbers to be operated upon
2
3
The expression is 2 + 3
The sum is 5
Program ended with exit code: 0
All Output
```

Q10. WAP to reverse a sentence using recursion.

Code:

```
#include <stdio.h>
#include <ctype.h>
void reverse(){
    char ch;
    scanf("%c", &ch);
    if (ch!='\n')
    {
        reverse();
        printf("%c", ch);
    }
}
int main(){
    printf("Enter a sentence: ");
    reverse();
    printf("\n");
}
```

Output:



```
Enter a sentence: Hello World
dlrow olleH
Program ended with exit code: 0
```

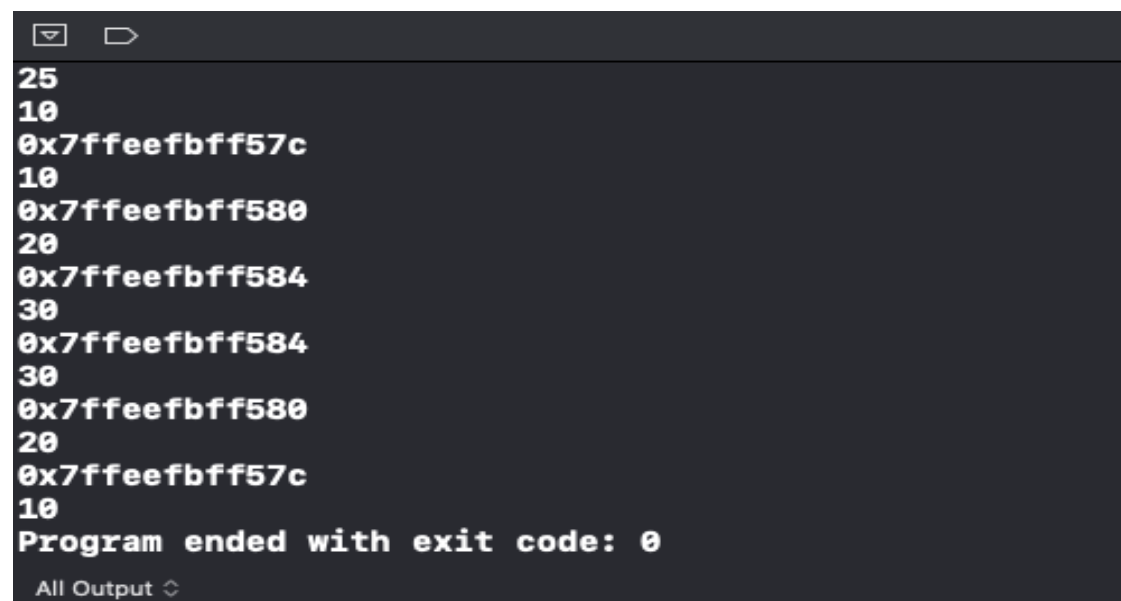
All Output ↕

Q11. There are four arithmetic operators that can be used in pointers ++,--,+-,*. TRUE

Code:

```
#include <stdio.h>
int main() {
    int *ptr,num=20,array[]={10,20,30};
    ptr=&num;
    *ptr=*ptr+5;          //adding 5 to the value of the pointer
    //using (*ptr)++ would add 1 to the value stored in the pointer
    printf("%d",*ptr);
    *ptr=*ptr-15;         //subtracting 15 from the new value of pointer
    //using (*ptr)-- would subtract 1 to the value stored in the pointer
    printf("\n%d",*ptr);
    ptr=array;
    for(int i=0; i<3; i++)
    {
        printf("\n%p",ptr);
        printf("\n%d",*ptr);
        ptr++;           //increment to the address of the pointer
    }
    ptr--;               /*the last instance of the above loop makes the pointer point at a garbage
    value as it exceeds the size of the array*/
    for(int i=0; i<3; i++)
    {
        printf("\n%p",ptr);
        printf("\n%d",*ptr);
        ptr--;           //decrement to the address of the pointer
    }
    printf("\n");
    return 0;
}
```

Output:



```
25
10
0x7ffeefbff57c
10
0x7ffeefbff580
20
0x7ffeefbff584
30
0x7ffeefbff584
30
0x7ffeefbff580
20
0x7ffeefbff57c
10
Program ended with exit code: 0
All Output
```

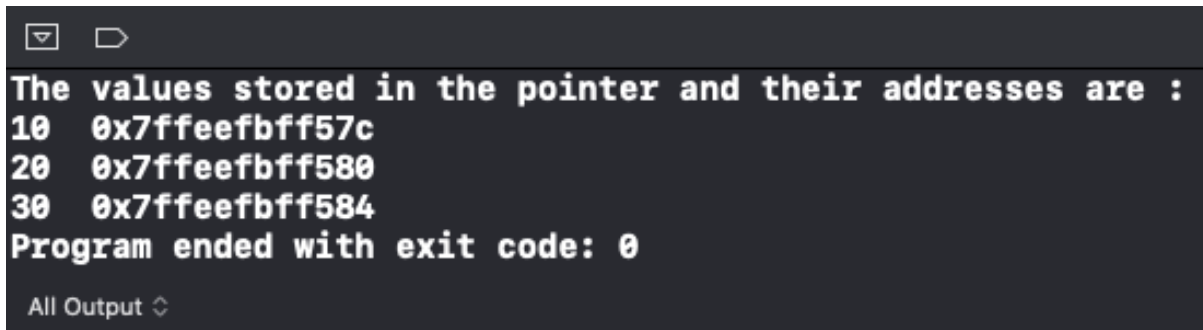
Q12. You can define arrays to hold a number of pointers. TRUE

Code:

```
#include <stdio.h>
int main() {
    int array[]={10,20,30},*ptr[3];
    for(int i=0 ; i<3; i++)
        ptr[i]=&array[i];    //stores the address of the various elements of the array//

    printf("The values stored in the pointer and their addresses are :\n");
    for(int i=0;i<3;i++)
    {
        printf("%d",*ptr[i]);
        printf("  %p\n",ptr[i]);
    }
    return 0;
}
```

Output:



```
The values stored in the pointer and their addresses are :
10  0x7ffefbfff57c
20  0x7ffefbfff580
30  0x7ffefbfff584
Program ended with exit code: 0
```

All Output

Q13. C allows you to have pointer on a pointer and so on. TRUE

Code:

```
#include <stdio.h>
int main() {
    int *ptr,num,**dptr;
    num=50;

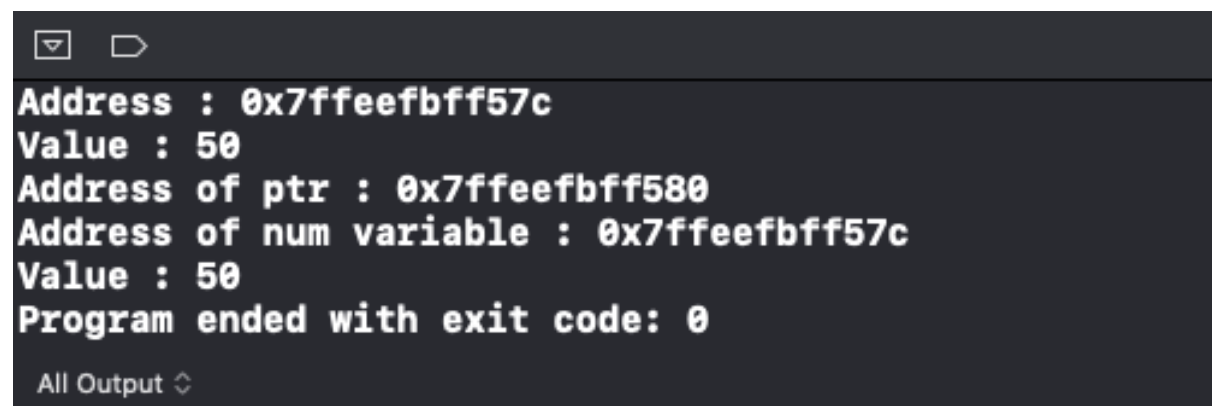
    ptr=&num;
    printf("Address : %p",ptr);
    printf("\nValue : %d",*ptr);

    dptr=&ptr;    // double pointer or pointer to a pointer
    printf("\nAddress of ptr : %p",dptr);    /*shows the address of the pointer, dptr pointer is
pointing to*/

    printf("\nAddress of num variable : %p",*dptr);    /*shows the address stored in the
pointer, dptr is pointing to*/

    printf("\nValue : %d\n",**dptr);    //show the value stored in the ptr pointer
    return 0;
}
/* Similarly, a triple pointer (***tptr) to the double pointer can be made which will store the
value and addresses of the double as well as the single pointer (ptr) and so on...*/
```

Output:



```
Address : 0x7ffeefbfff57c
Value : 50
Address of ptr : 0x7ffeefbfff580
Address of num variable : 0x7ffeefbfff57c
Value : 50
Program ended with exit code: 0

All Output
```

Q14. Passing an argument by reference or by address enables the passed argument to be changed in the calling function by the called function. TRUE

Code:

```
#include <stdio.h>
void increment_20(int *x, int *y)
{
    *x=*x+20;
    *y=*y+20;
}

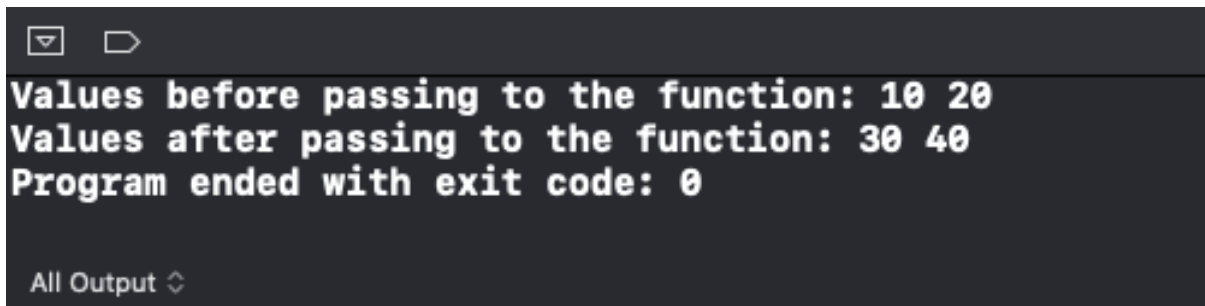
int main() {
    int a = 10, b=20;
    printf("Values before passing to the function: %d %d",a,b);
    increment_20(&a,&b);    /* increases the passed arguments by 20, the address of the
                             variables is being passed as arguments to the function*/

    printf("\nValues after passing to the function: %d %d\n",a,b);

    /*Hence passing the arguments using call by reference method changes the original
    values of the variables.*/

    return 0;
}
```

Output:



```
Values before passing to the function: 10 20
Values after passing to the function: 30 40
Program ended with exit code: 0
```

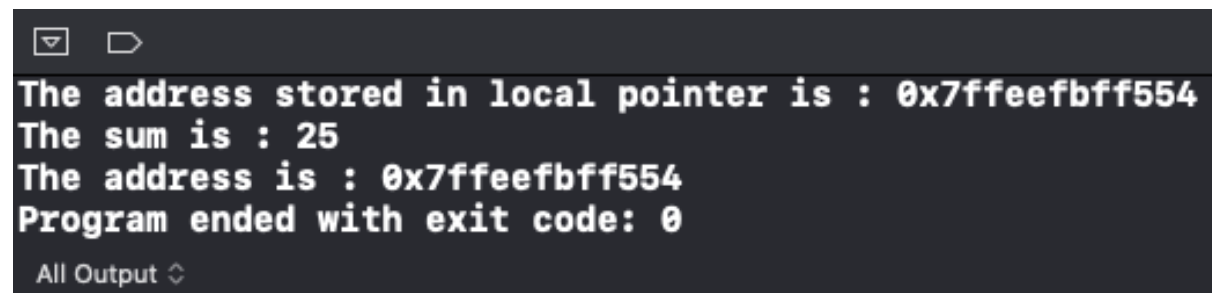
Q15. C allows a function to return a pointer to the local variable. TRUE

Code:

```
#include <stdio.h>
int *add(int x,int y)
{
    int addition, *result;
    addition = x+y;
    result = &addition;
    printf("The address stored in local pointer is : %p", result);
    return result;    //returning a local pointer
}

int main() {
    int a = 10,b=15;
    int *sum = add(a,b);    //function to add two numbers
    printf("\nThe sum is : %d",*sum);
    printf("\nThe address is : %p\n", sum); //this shows that the local pointer has been returned
    return 0;
}
```

Output:



```
The address stored in local pointer is : 0x7ffeefbfff554
The sum is : 25
The address is : 0x7ffeefbfff554
Program ended with exit code: 0
```

All Output ↕

Q16. WAP to implement Stack using array.

Code:

```
#include <stdio.h>
#include <ctype.h>
int top=-1,max=5;
char input;
void push(char stack[]){
    printf("Enter the element to be added : ");
    scanf("%s",&input);
    if(top==max-1)
    {
        printf("The stack is completely filled. OVERFLOW!!");
    }
    else{
        top++;
        stack[top]=input;
    }
}
void pop(char stack[]){
    if(top== -1)
    {
        printf("The stack is empty. UNDERFLOW!!");
    }
    else{
        top--;
    }
}
void peek(char stack[]){
    if(top== -1){
        printf("The stack is empty !!\n");
        printf("Top: %c",stack[top]);
    }
    else
        printf("Top: %c",stack[top]);
}
void display(char stack[]){
    if(top== -1)
        printf("The stack is empty.");
    else{
        printf("Stack elements: ");
        for(int i=0; i<=top; i++)
            printf("%c ",stack[i]);
    }
}
void reverse(char stack[]){
    char temp;
    for(int i=0,j=top; i<j; i++,j--)
```

```

    {
        temp=stack[i];
        stack[i]=stack[j];
        stack[j]=temp;
    }
}
int main() {
    int selection;
    char stack[max];
    char conti;
    printf("Enter the max size of the stack : ");
    scanf("%d",&max);
    do{
        printf("\nChoose one of the following:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Traverse\n");
        printf("5. Reverse\n");
        scanf("%d",&selection);
        switch(selection){
            case 1: push(stack);
                break;
            case 2: pop(stack);
                break;
            case 3: peek(stack);
                break;
            case 4: display(stack);
                break;
            case 5: reverse(stack);
                break;
            default : printf("Choose a valid operation from the list.");
                break;
        }

        printf("\nDo you want to continue? (Y/N) ");
        scanf("%s", &conti);

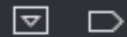
        conti=tolower(conti);

    } while(conti!='n');

    return 0;
}

```

Output:



Enter the max size of the stack : 5

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

1

Enter the element to be added : 1

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

1

Enter the element to be added : 2

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

1

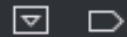
Enter the element to be added : 3

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

All Output ↕



Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

4

Stack elements: 1 2 3

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

3

Top: 3

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

2

Do you want to continue? (Y/N) y

Choose one of the following:

1. Push
2. Pop
3. Peek
4. Traverse
5. Reverse

4

Stack elements: 1 2

Do you want to continue? (Y/N) n

All Output ↕

```
Stack elements: 1 2
Do you want to continue? (Y/N) y
```

```
Choose one of the following:
```

1. Push
 2. Pop
 3. Peek
 4. Traverse
 5. Reverse
- 5

```
Do you want to continue? (Y/N) y
```

```
Choose one of the following:
```

1. Push
 2. Pop
 3. Peek
 4. Traverse
 5. Reverse
- 4

```
Stack elements: 2 1
```

```
Do you want to continue? (Y/N) n
```

```
Program ended with exit code: 0
```

All Output ↕

Q17. WAP to check nesting of parentheses using stack.

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define max 20
int top=-1;

char stack[max];

void push(char input){
    if(top==max-1)
    {
        printf("The stack is completely filled. OVERFLOW!!");
    }
    else{
        top++;
        stack[top]=input;
    }
}

void pop(){
    if(top== -1)
    {
        printf("The stack is empty. UNDERFLOW!!");
    }
    else{
        top--;
    }
}

int main(){
    char string[20];
    printf("Enter an expression: ");
    scanf("%s",string);
    //printf("Input: %s\n",string);

    for(int i = 0; i<strlen(string); i++){
        if(string[i]=='(' || string[i]=='[' || string[i]=='{ '){
            push(string[i]);
            continue;
        }

        else if(string[i]==')' || string[i]==']' || string[i]=='}')
        {
            switch(string[i]){
                case ')':
```

```

        if(stack[top]=='(')
            pop();
        else{
            printf("Unbalanced expression.\n");
            exit(1);
        }
        break;

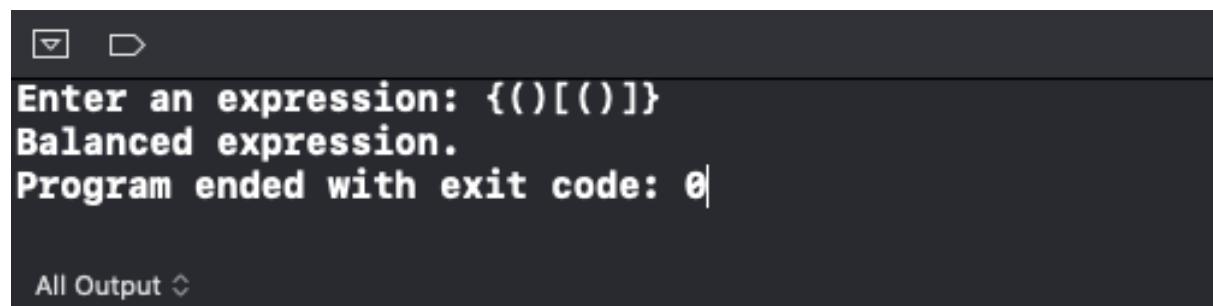
    case ']':
        if(stack[top]=='[')
            pop();
        else{
            printf("Unbalanced expression.\n");
            exit(1);
        }
        break;

    case '}':
        if(stack[top]=='{')
            pop();
        else{
            printf("Unbalanced expression.\n");
            exit(1);
        }
        break;
    }
}
}
if(top==1){
    printf("Balanced expression.\n");
}
else
    printf("Unbalanced expression.\n");

return 0;
}

```

Output:



```

Enter an expression: {}()[()]
Balanced expression.
Program ended with exit code: 0

```

All Output



```
Enter an expression: {()[}  
Unbalanced expression.  
Program ended with exit code: 0
```

All Output ↕

Q18. WAP to convert infix to postfix using stacks.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define maxsize 100
char stack[maxsize];
int top = -1;

void push(char item)
{
    if(top >= maxsize-1)
    {
        printf("\nStack Overflow.");
    }
    else
    {
        top = top+1;
        stack[top] = item;
    }
}

char pop()
{
    char item ;
    if(top <0)
    {
        printf("stack under flow: invalid infix expression");
        exit(1);
    }
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
        return 1;

    else
        return 0;
```

```

}

int precedence(char symbol)
{

    if(symbol == '^')
        return 3;

    else if(symbol == '*' || symbol == '/')
        return 2;

    else if(symbol == '+' || symbol == '-') /* lowest precedence */
        return 1;

    else
        return 0;

}

void Infix_To_Postfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;
    push('(');
    strcat(infix_exp, "");
    i=0;
    j=0;

    item=infix_exp[i];

    while(item != '\0')
    {
        if(item == '(')
        {
            push(item);
        }
        else if( isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;
        }
        else if(is_operator(item) == 1)
        {
            x=pop();
            while(is_operator(x) == 1 && precedence(x)>= precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
    }
}

```

```

    }
    push(x);
    push(item);
}
else if(item == ')')
{
    x = pop();
    while(x != '(')
    {
        postfix_exp[j] = x;
        j++;
        x = pop();
    }
}
else
{
    printf("\nInvalid infix Expression.\n");
    exit(1);
}
i++;
item = infix_exp[i];
}
if(top>0)
{
    printf("\nInvalid infix Expression.\n");
    exit(1);
}
if(top>0)
{
    printf("\nInvalid infix Expression.\n");
    exit(1);
}
postfix_exp[j] = '\0';
}

int main()
{
    char infix[maxsize], postfix[maxsize];
    printf("Enter the expression : ");
    scanf("%s",infix);
    Infix_To_Postfix(infix,postfix);
    printf("Postfix Expression: ");
    printf("%s\n",postfix);
    return 0;
}

```

Output:

```
Enter the expression : a+b*(c^d-e)^(f+g*h)-i
Postfix Expression: abcd^e-fgh*+^*+i-
Program ended with exit code: 0
```

All Output ↕

```
Enter the expression : a+b+c*d-e
Postfix Expression: ab+cd*+e-
Program ended with exit code: 0
```

All Output ↕

Q19. WAP to implement Tower of Hanoi using recursion.

Code:

```
#include <stdio.h>

#include <math.h>

void tower_of_hanoi(int n, char OR, char DR, char AR)
//OR/A- Original Rod, AR/B- Auxilliary Rod, DR/C- Destination Rod
{
    if (n == 1)
    {
        printf("Move disk 1 from rod %c to rod %c\n", OR, DR);
        return;
    }
    tower_of_hanoi(n - 1, OR, AR, DR);
    printf("Move disk %d from rod %c to rod %c\n", n, OR, DR);
    tower_of_hanoi(n - 1, AR, DR, OR);
}

int main()
{
    int n;

    printf("Enter the number of disks : ");
    scanf("%d", &n);

    printf("The order of disk movement to solve Tower of Hanoi are :");
    tower_of_hanoi(n, 'A', 'C', 'B');

    return 0;
}
```

Output:

```
Enter the number of disks : 3
The order of disk movement to solve Tower of Hanoi are :
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
Program ended with exit code: 0

All Output ↕
```

Q20. WAP to implement Queue using array.

Code:

```
#include <stdio.h>
#include <ctype.h>
int max=10;
int rear=-1;
int front=-1;
void enqueue(int queue[]){
    int item;

    if(rear==max-1)
        printf("OVERFLOW! The queue is full.");

    else if(front==0){
        printf("Enter the element to be added: ");
        scanf("%d",&item);
        front++;
        rear++;
        queue[rear]=item;
    }

    else{
        printf("Enter the element to be added: ");
        scanf("%d",&item);
        rear++;
        queue[rear]=item;
    }
}

void dequeue(int queue[]){
    if(front==0 && rear==0)
        printf("UNDERFLOW! The queue is empty");
    else
        front++;
}

void peek(int queue[]){
    if(rear==0)
        printf("UNDERFLOW! The queue is empty");
    else
        printf("The top of the queue is: %d",queue[front]);
}

void traverse(int queue[]){
    if(front==0)
        printf("UNDERFLOW! The queue is empty.");
    else{
        printf("The queue is:\n");
```

```

        for(int i = front; i<=rear; i++)
            printf("%d ",queue[i]);
    }
}

int main()
{
    printf("Enter the size of the queue: ");
    scanf("%d",&max);

    int queue[max];
    int selection;
    char conti;

    do{
        printf("\nChoose one of the following:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Traverse\n");
        scanf("%d",&selection);

        switch(selection){
            case 1: enqueue(queue);
                    break;

            case 2: dequeue(queue);
                    break;

            case 3: peek(queue);
                    break;

            case 4: traverse(queue);
                    break;

            default : printf("Choose a valid operation from the list.");
                    break;
        }

        printf("\nDo you want to continue? (Y/N) ");
        scanf("%s", &conti);

        conti=tolower(conti);

    }while(conti!='n');

    return 0;
}

```


Output:

```
Enter the size of the queue: 3

Choose one of the following:
1. Enqueue
2. Dequeue
3. Peek
4. Traverse
1
Enter the element to be added: 1

Do you want to continue? (Y/N) y

Choose one of the following:
1. Enqueue
2. Dequeue
3. Peek
4. Traverse
1
Enter the element to be added: 2

Do you want to continue? (Y/N) y

Choose one of the following:
1. Enqueue
2. Dequeue
3. Peek
4. Traverse
1
Enter the element to be added: 3
```

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

3

The top of the queue is: 1

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

4

The queue is:

1 2 3

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

2

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

4

The queue is:

2 3

Do you want to continue? (Y/N) n

Program ended with exit code: 0

Q21. WAP to implement Circular queue using array.

Code:

```
#include <stdio.h>
#include <ctype.h>
int max=10;
int rear=-1;
int front=-1;
void enqueue(int queue[]){
    int item;

    if(front!=0 && rear==max-1){
        printf("Enter the element to be added: ");
        scanf("%d",&item);
        rear=0;
        queue[rear]=item;
    }

    else if(rear==max-1 || rear==front-1)
        printf("OVERFLOW! The queue is full.");

    else if(front==-1){
        printf("Enter the element to be added: ");
        scanf("%d",&item);
        front++;
        rear++;
        queue[rear]=item;
    }

    else{
        printf("Enter the element to be added: ");
        scanf("%d",&item);
        rear++;
        queue[rear]=item;
    }
}

void dequeue(int queue[]){
    if(front==-1 && rear==-1)
        printf("UNDERFLOW! The queue is empty");
    else if(front==max-1 && rear<max-1)
        front=0;
    else
        front++;
}

void peek(int queue[]){
    if(rear==-1)
        printf("UNDERFLOW! The queue is empty");
```

```

    else
        printf("The top of the queue is: %d",queue[front]);
}
void traverse(int queue[]){
    if(front==-1)
        printf("UNDERFLOW! The queue is empty.");
    else{
        if(front<rear){
            printf("The queue is:\n");
            for(int i = front; i<=rear; i++)
                printf("%d ",queue[i]);
        }

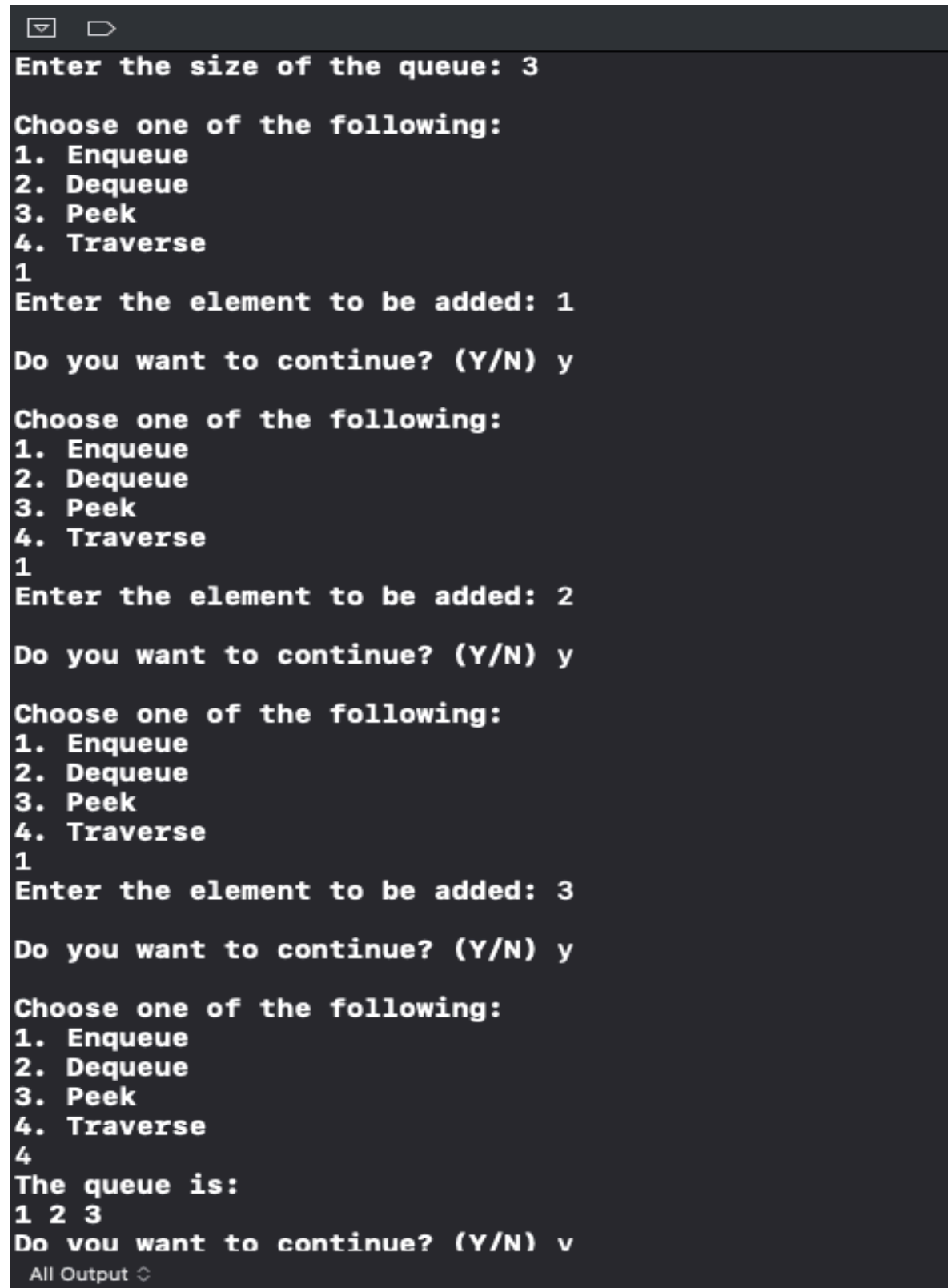
        else if(rear<front){
            printf("The queue is:\n");
            for(int i = front; i<max; i++)
                printf("%d ",queue[i]);
            for(int i = 0; i<=rear; i++)
                printf("%d ",queue[i]);
        }
    }
}

int main()
{
    printf("Enter the size of the queue: ");
    scanf("%d",&max);
    int queue[max];
    int selection;
    char conti;
    do{
        printf("\nChoose one of the following:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Traverse\n");
        scanf("%d",&selection);
        switch(selection){
            case 1: enqueue(queue);
                    break;
            case 2: dequeue(queue);
                    break;
            case 3: peek(queue);
                    break;
            case 4: traverse(queue);
                    break;
            default : printf("Choose a valid operation from the list.");
                     break;
        }
        printf("\nDo you want to continue? (Y/N) ");
        scanf("%s", &conti);
    }
}

```

```
        conti=tolower(conti);  
    }while(conti!='n');  
    return 0;  
}
```

Output:



```
Enter the size of the queue: 3  
Choose one of the following:  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Traverse  
1  
Enter the element to be added: 1  
Do you want to continue? (Y/N) y  
Choose one of the following:  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Traverse  
1  
Enter the element to be added: 2  
Do you want to continue? (Y/N) y  
Choose one of the following:  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Traverse  
1  
Enter the element to be added: 3  
Do you want to continue? (Y/N) y  
Choose one of the following:  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Traverse  
4  
The queue is:  
1 2 3  
Do you want to continue? (Y/N) n  
All Output
```

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

2

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

4

The queue is:

2 3

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

1

Enter the element to be added: 5

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

4

The queue is:

2 3 5

Do you want to continue? (Y/N) y

Choose one of the following:

1. Enqueue
2. Dequeue
3. Peek
4. Traverse

3

The top of the queue is: 2

Do you want to continue? (Y/N) n

Program ended with exit code: 0

All Output ↕

Q22. WAP to check for palindrome using queue.

Code:

```
#include <stdio.h>
#include <stdlib.h>
int max=20;
int rear=-1;
int front=-1;
void enqueue(char queue[],char item){
    if(rear==max-1)
        printf("OVERFLOW! The queue is full.");

    else if(front==-1){
        front++;
        rear++;
        queue[rear]=item;
    }

    else{
        rear++;
        queue[rear]=item;
    }
}

void dequeue(char queue[]){
    if(front==-1 && rear==-1)
        printf("UNDERFLOW! The queue is empty");
    else{
        front++;
        rear--;
    }
}

void traverse(char queue[], int length){
    for(int i=0; i<=length/2; i++)
    {
        if(queue[front]==queue[rear])
            dequeue(queue);
        else{
            printf("It is not a palindrome\n");
            exit(1);
        }
    }
    printf("It is a palindrome.");
}

int main()
{
```



```
char queue[max],string[max];

printf("Enter the string: ");
scanf("%[^\\n]%*c",string);

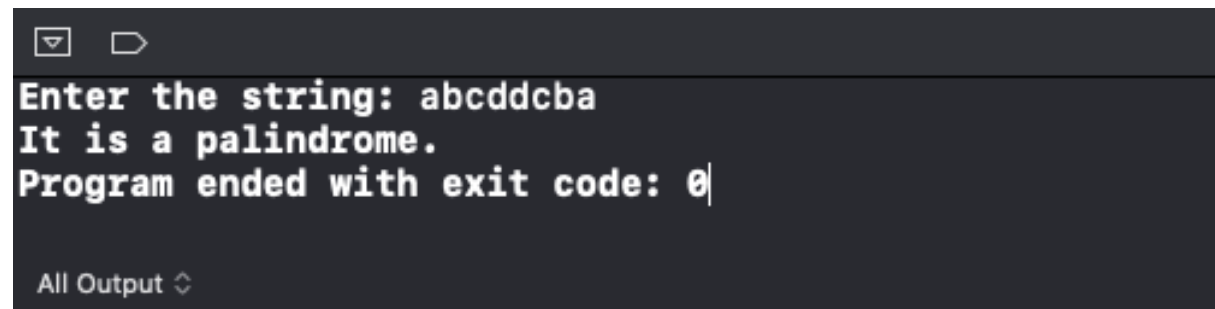
for(int i=0; string[i]!='\\0'; i++)
    enqueue(queue,string[i]);

int strlen=rear;

traverse(queue,strlen);
printf("\\n");

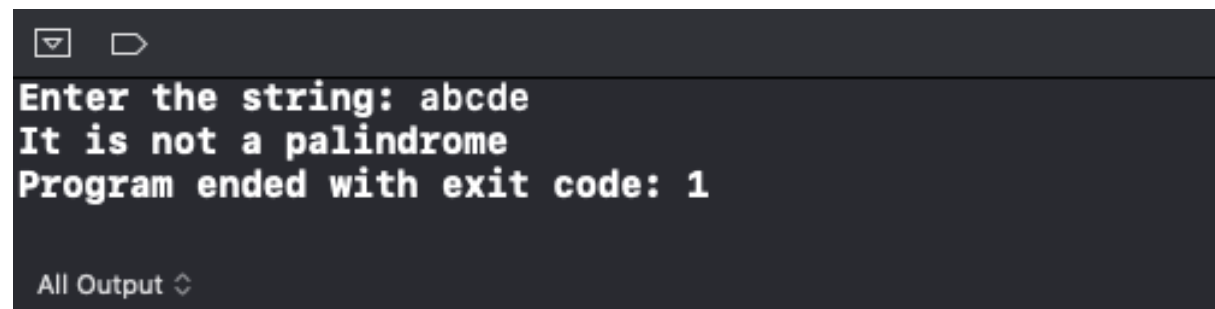
return 0;
}
```

Output:



```
Enter the string: abcddcba
It is a palindrome.
Program ended with exit code: 0
```

All Output ↕



```
Enter the string: abcde
It is not a palindrome
Program ended with exit code: 1
```

All Output ↕

Q23. WAP to implement Singly Linked List.

Code:

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

int length(void);
struct node{
    int data;
    struct node *link;
}*root;

void add_end(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    right=root;
    while(right->link!=NULL)
        right=right->link;
    right->link=temp;
    right=temp;
    right->link=NULL;
}

void add_begin( int num ){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    if (root== NULL)
    {
        root=temp;
        root->link=NULL;
    }
    else
    {
        temp->link=root;
        root=temp;
    }
}

void addafter(int num, int loc){
    int i;
    struct node *temp,*left=NULL,*right;
    right=root;
    for(i=0;i<loc;i++)
    {
        left=right;
        right=right->link;
    }
}
```

```

temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
left->link=temp;
left=temp;
left->link=right;
}

```

```

void insert(int num){
    int c=0;
    struct node *temp;
    temp=root;
    if(temp==NULL)
    {
        add_begin(num);
    }
    else
    {
        while(temp!=NULL)
        {
            if(temp->data<num)
                c++;
            temp=temp->link;
        }
        if(c==0)
            add_begin(num);
        else if(c<length())
            addafter(num,c);
        else
            add_end(num);
    }
}

int delete_index(int num){
    struct node *temp, *prev = NULL;
    int i=0;
    temp=root;
    while(temp!=NULL)
    {
        if(i==num)
        {
            if(temp==root)
            {
                root=temp->link;
                free(temp);
                return 1;
            }
            else
            {
                prev->link=temp->link;
                free(temp);
                return 1;
            }
        }
        prev=temp;
        temp=temp->link;
        i++;
    }
}

```

```

    }
}
else
{
    i++;
    prev=temp;
    temp=temp->link;
}
}
return 0;
}

```

```

int delete_data(int num){
    struct node *temp, *prev = NULL;
    temp=root;
    while(temp!=NULL)
    {
        if(temp->data==num)
        {
            if(temp==root)
            {
                root=temp->link;
                free(temp);
                return 1;
            }
            else
            {
                prev->link=temp->link;
                free(temp);
                return 1;
            }
        }
        else
        {
            prev=temp;
            temp=temp->link;
        }
    }
    return 0;
}

```

```

int delete_after(int num){
    struct node *temp, *prev=NULL;
    temp=(struct node *)malloc(sizeof(struct node));
    temp=root;
    while(temp!=NULL){
        if(temp->data==num){
            prev=temp;
            temp=temp->link;
            if(temp==NULL)

```

```

        return 2;

    else{
        prev->link=temp->link;
        free(temp);
        return 1;
    }
}
else
    temp=temp->link;
}
return 0;
}

void display(){
    struct node *r;
    r=root;
    if(r==NULL)
    {
        return;
    }
    while(r!=NULL)
    {
        printf("%d ",r->data);
        r=r->link;
    }
    printf("\n");
}

int length(){
    struct node *n;
    int c=0;
    n=root;
    while(n!=NULL)
    {
        n=n->link;
        c++;
    }
    return c;
}

void first_last(){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp=root;
    add_end(temp->data);
    root=temp->link;
    free(temp);
    printf("Successfully shifted.\n");
}

```

```

int main(){
    int i,num;
    root=NULL;
    do{
        printf("\nList Operations\n");
        printf("=====\n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Size\n");
        printf("4.Delete using index\n");
        printf("5.Delete using data\n");
        printf("6.Delete after\n");
        printf("7.Addition of first element to last\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            printf("Enter only an Integer\n");
            exit(1);
        } else {
            switch(i)
            {
                case 1:printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;
                case 2:if(root==NULL)
                {
                    printf("List is Empty\n");
                }
                else
                {
                    printf("Element(s) in the list are : ");
                }
                display();
                break;
                case 3:printf("Size of the list is: %d\n",length());
                    break;

                case 4:
                    if(root==NULL)
                        printf("List is Empty. UNDERFLOW!\n");
                    else{
                        printf("Enter the index to be delete : ");
                        scanf("%d",&num);
                        if(delete_index(num-1))
                            printf("Deleted Successfully\n");
                        else
                            printf("%d not found in the list\n",num);
                    }
            }
        }
    } while (i!=7);
}

```

```

    }
    break;

case 5:
    if(root==NULL)
        printf("List is Empty. UNDERFLOW!\n");
    else{
        printf("Enter the number to be delete : ");
        scanf("%d",&num);
        if(delete_data(num))
            printf("Deleted Successfully\n");
        else
            printf("%d not found in the list\n",num);
    }
    break;

case 6:
    if(root==NULL)
        printf("List is Empty. UNDERFLOW!\n");
    else{
        printf("Enter the number to delete after : ");
        scanf("%d",&num);
        if(delete_after(num))
            printf("Deleted Successfully\n");
        else if(delete_after(num)==2)
            printf("There is no element after %d",num);
        else
            printf("%d not found in the list\n",num);
    }
    break;

case 7:
    if(root==NULL)
        printf("List is Empty. UNDERFLOW!\n");
    else
        first_last();
    break;

default: printf("Invalid option\n");
}
}
printf("Press 1 to continue...");
scanf("%d",&num);
}while(num==1);
return 0;
}

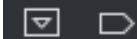
```

Output:

```

List Operations
=====
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 1
Enter the number to insert : 4
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 2
Element(s) in the list are : 1 2 3 4
Press 1 to continue...1
```

List Operations

=====



List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 3
Size of the list is: 4
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 4
Enter the index to be deleted : 3
Deleted Successfully
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 2
Element(s) in the list are : 1 2 4
Press 1 to continue...1
```

List Operations

All Output ↕

☐ ☐

```
List Operations
=====
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 5
Enter the number to be delete : 2
Deleted Successfully
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 2
Element(s) in the list are : 1 4
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 6
Enter the number to delete after : 1
Deleted Successfully
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 2
Element(s) in the list are : 1
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Size
4.Delete using index
5.Delete using data
6.Delete after
7.Addition of first element to last
Enter your choice : 7
Successfully shifted.
Press 1 to continue...1
```

List Operations

=====

1.Insert

2.Display

3.Size

4.Delete using index

5.Delete using data

6.Delete after

7.Addition of first element to last

Enter your choice : 2

Element(s) in the list are : 3 1

Press 1 to continue...0

Program ended with exit code: 0

All Output ↕

Q24. WAP to count the node in a linked list.

Code:

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

int length(void);
struct node{
    int data;
    struct node *link;
}*root;

void insert(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;

    if(root==NULL){
        root=temp;
        root->link=NULL;
    }

    else{
        right=root;

        while(right->link!=NULL)
            right=right->link;

        right->link=temp;
        right=temp;
        right->link=NULL;
    }
}

void count(int element){
    int count=0;
    struct node *r;
    r=root;
    if(r==NULL)
    {
        return;
    }
    while(r!=NULL)
    {
        printf("%d ",r->data);
        if(r->data == element)
            count++;
        r=r->link;
    }
}
```

```

    printf("\n");
    printf("%d occurs %d times in the linked list.\n",element,count);
}

int main(){
    int i,num = 0;
    root=NULL;
    do{
        printf("\nList Operations\n");
        printf("=====\n");
        printf("1.Insert\n");
        printf("2.Count\n");

        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            printf("Enter only an Integer\n");
            exit(1);
        } else {
            switch(i)
            {
                case 1:
                    printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;
                case 2:if(root==NULL)
                {
                    printf("List is Empty\n");
                }
                else
                {
                    printf("Enter the element to be searched for: ");
                    scanf("%d",&num);
                    printf("Element(s) in the list are : ");
                    count(num);
                }
                break;

                default: printf("Invalid option\n");
            }
        }
        printf("Press 1 to continue...");
        scanf("%d",&num);
    }while(num==1);
    return 0;
}

```

Output:

☐ ☐
List Operations
=====

- 1.Insert
- 2.Count

Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====

- 1.Insert
- 2.Count

Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1

List Operations
=====

- 1.Insert
- 2.Count

Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1

List Operations
=====

- 1.Insert
- 2.Count

Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations

=====

1.Insert

2.Count

Enter your choice : 1

Enter the number to insert : 1

Press 1 to continue...1

List Operations

=====

1.Insert

2.Count

Enter your choice : 1|

Enter the number to insert : 4

Press 1 to continue...1

List Operations

=====

1.Insert

2.Count

Enter your choice : 2

Enter the element to be searched for: 1

Element(s) in the list are : 1 3 2 1 1 4

1 occurs 3 times in the linked list.

Press 1 to continue...0

Program ended with exit code: 0

All Output ↕

Q25. WAP to reverse elements of a Linked list.

Code:

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

int length(void);
struct node{
    int data;
    struct node *link;
}*root;

void insert(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;

    if(root==NULL){
        root=temp;
        root->link=NULL;
    }

    else{
        right=root;

        while(right->link!=NULL)
            right=right->link;

        right->link=temp;
        right=temp;
        right->link=NULL;
    }
}

void display(){
    struct node *r;
    r=root;
    if(r==NULL)
    {
        return;
    }
    while(r!=NULL)
    {
        printf("%d ",r->data);
        r=r->link;
    }
    printf("\n");
}
```

```

int length(){
    struct node *n;
    int c=0;
    n=root;
    while(n!=NULL)
    {
        n=n->link;
        c++;
    }
    return c;
}

```

```

void reverse(){
    struct node *left, *right;
    int len=length()-1,i=0,temp;
    left=root;
    right=root;
    while(len!=(len/2)){
        while(i!=len){
            right=right->link;
            i++;
        }

        temp=left->data;
        left->data=right->data;
        right->data=temp;

        left=left->link;
        len--;
    }
}

```

```

int main(){
    int i,num;
    root=NULL;
    do{
        printf("\nList Operations\n");
        printf("=====\n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Reverse\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            printf("Enter only an Integer\n");
            exit(1);
        } else {
            switch(i)
            {

```

```

case 1:
    printf("Enter the number to insert : ");
    scanf("%d",&num);
    insert(num);
    break;

case 2:
    if(root==NULL){
        printf("List is Empty\n");
    }
    else{
        printf("Element(s) in the list are : ");
        display();
    }
    break;

case 3:
    if(root==NULL)
        printf("List is Empty\n");

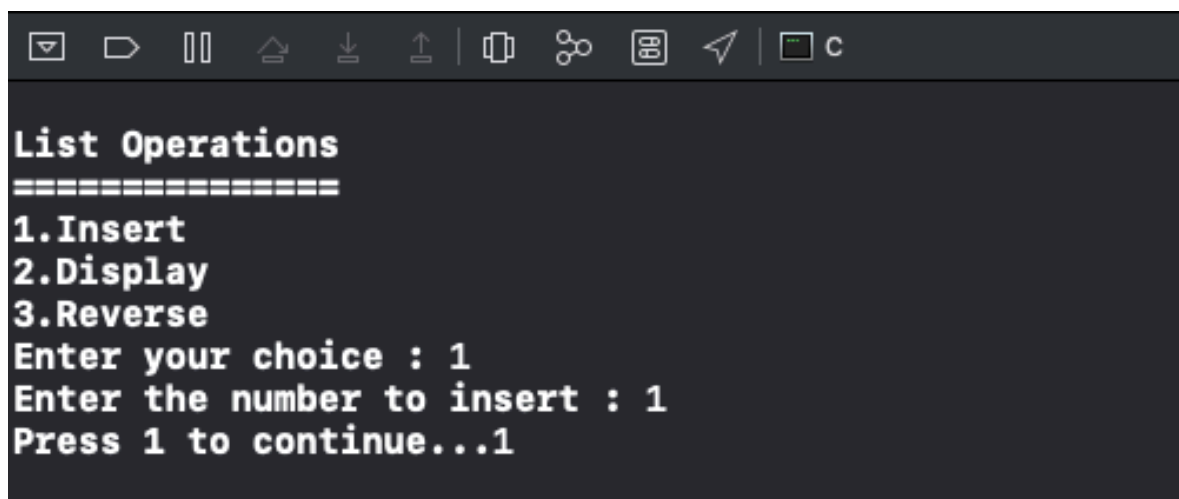
    else
        reverse();

    break;

default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
}while(num==1);
return 0;
}

```

Output:



```

List Operations
=====
1.Insert
2.Display
3.Reverse
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

```

List Operations

=====

1.Insert

2.Display

3.Reverse

Enter your choice : 1

Enter the number to insert : 2

Press 1 to continue...1

List Operations

=====

1.Insert

2.Display

3.Reverse

Enter your choice : 2

Element(s) in the list are : 1 2

Press 1 to continue...1

List Operations

=====

1.Insert

2.Display

3.Reverse

Enter your choice : 3

Press 1 to continue...1

List Operations

=====

1.Insert

2.Display

3.Reverse

Enter your choice : 2

Element(s) in the list are : 2 1

Press 1 to continue...

All Output ↕

Q26. WAP to implement Stack using linked list.

Code:

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

int length(void);
struct node{
    int data;
    struct node *link;
}*root;

void insert(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;

    if(root==NULL){
        root=temp;
        root->link=NULL;
    }

    else{
        right=root;

        while(right->link!=NULL)
            right=right->link;

        right->link=temp;
        right=temp;
        right->link=NULL;
    }
}

int delete(){
    struct node *temp, *prev = NULL;
    temp=root;
    while(temp->link!=NULL)
    {
        prev=temp;
        temp=temp->link;
    }
    free(temp);
    prev->link=NULL;
    return 0;
}

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

```

r=root;
if(r==NULL)
{
    return;
}
while(r!=NULL)
{
    printf("%d ",r->data);
    r=r->link;
}
printf("\n");
}

```

```

int main() {
    int i,num;
    root=NULL;
    do {
        printf("\nList Operations\n");
        printf("===== \n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Delete\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            printf("Enter only an Integer\n");
            exit(1);
        } else {
            switch(i)
            {
                case 1:
                    printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;

                case 2:
                    if(root==NULL){
                        printf("List is Empty\n");
                    }
                    else {
                        printf("Element(s) in the list are : ");
                        display();
                    }

                    break;

                case 3:
                    if(root==NULL)
                        printf("List is Empty. UNDERFLOW!\n");

```

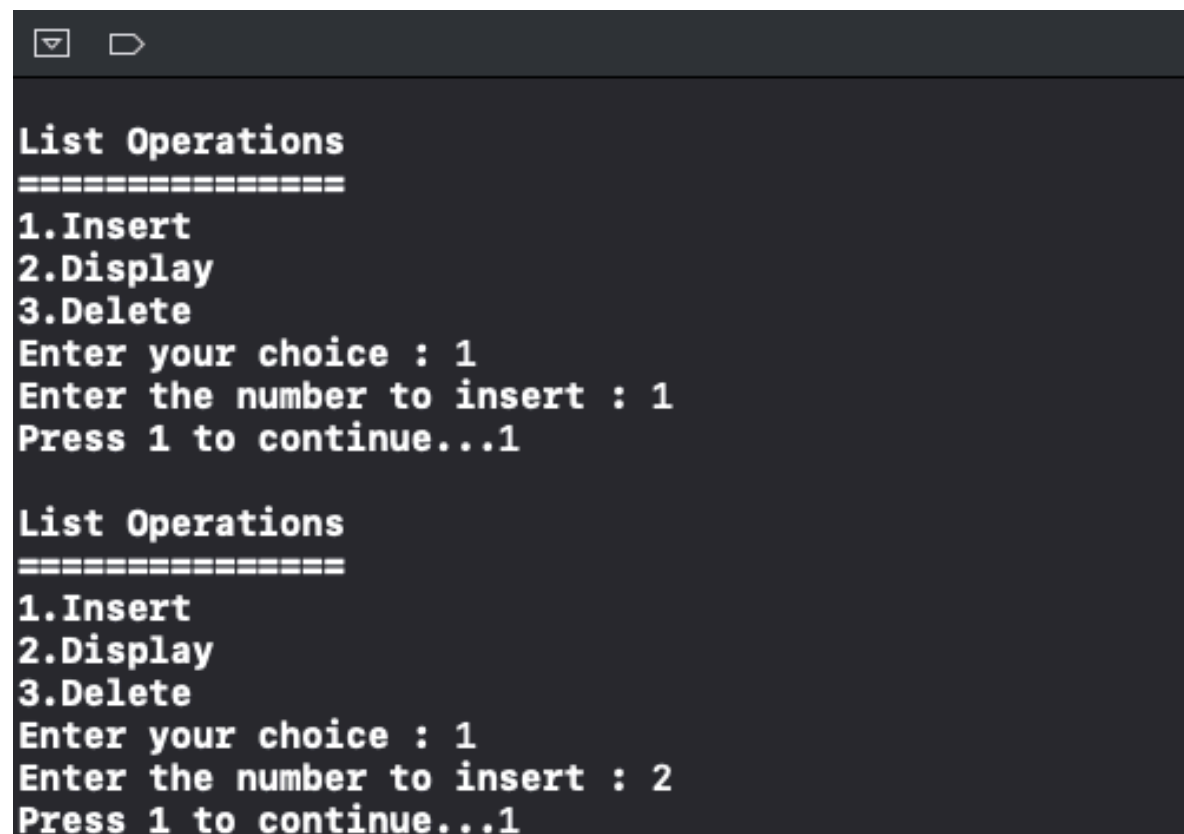
```

        else{
            delete();
        }
        break;

    default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
}while(num==1);
return 0;
}

```

Output:



```

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1

```



```

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 2
Element(s) in the list are : 1 2 3
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 3
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 3
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 2
Element(s) in the list are : 1

```

Q27. WAP to implement Queue using linked list.

Code:

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

int length(void);
struct node{
    int data;
    struct node *link;
}*root;

void insert(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    if (root==NULL)
    {
        root=temp;
        root->link=NULL;
    }
    else{
        right=root;
        while(right->link!=NULL)
            right=right->link;
        right->link=temp;
        right=temp;
        right->link=NULL;
    }
}

int delete(){
    struct node *temp;
    temp=root;
    root=root->link;
    free(temp);

    return 0;
}

void display(){
    struct node *r;
    r=root;
    if(r==NULL)
    {
        return;
    }
    while(r!=NULL)
    {
```

```

        printf("%d ",r->data);
        r=r->link;
    }
    printf("\n");
}

```

```

int main(){
    int i,num;
    root=NULL;
    do{
        printf("\nList Operations\n");
        printf("=====\n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Delete\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            printf("Enter only an Integer\n");
            exit(1);
        } else {
            switch(i)
            {
                case 1:
                    printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;

                case 2:
                    if(root==NULL){
                        printf("List is Empty\n");
                    }
                    else{
                        printf("Element(s) in the list are : ");
                        display();
                    }
                    break;

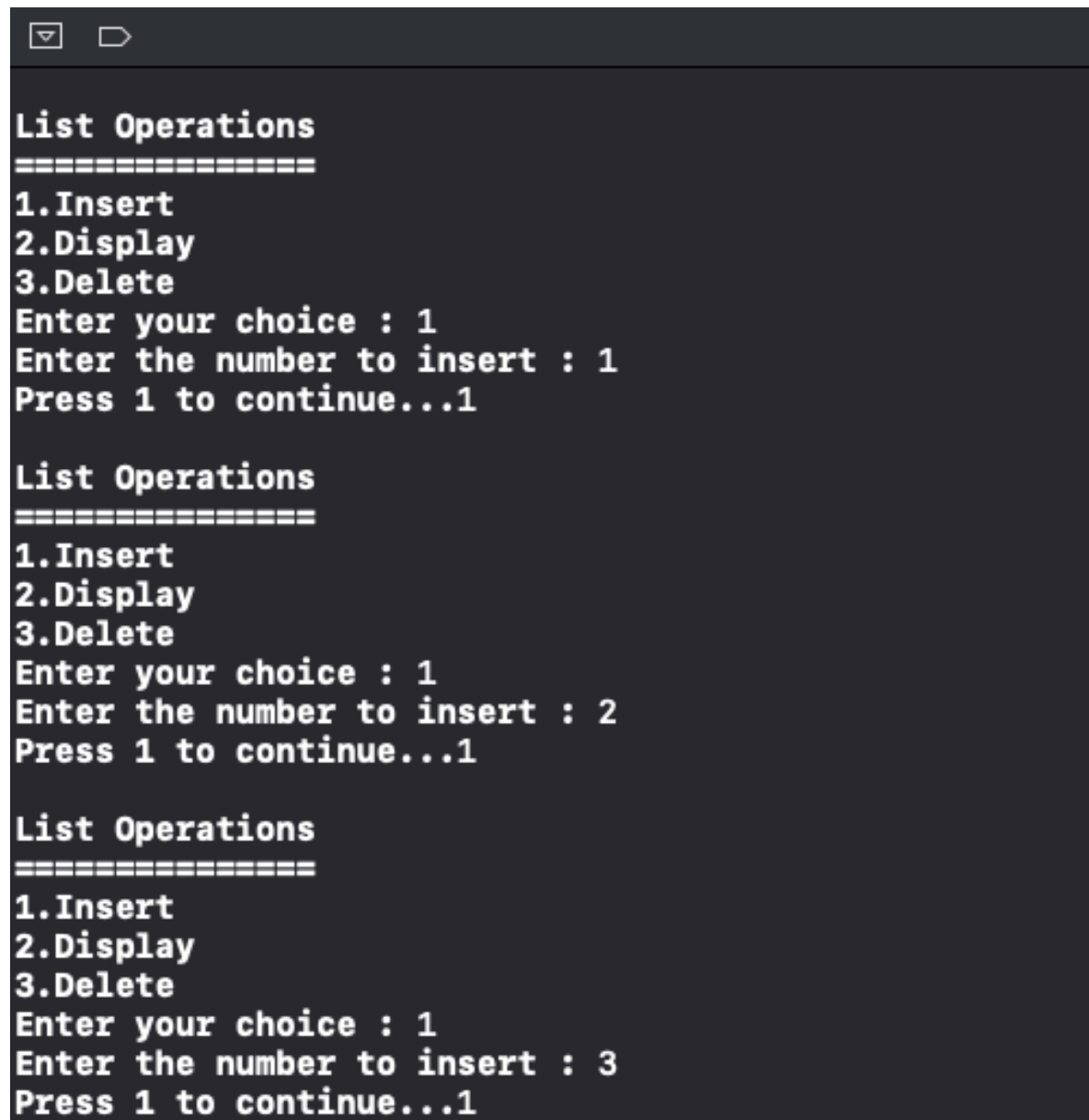
                case 3:
                    if(root==NULL)
                        printf("List is Empty. UNDERFLOW!\n");
                    else{
                        delete();
                    }
                    break;

                default: printf("Invalid option\n");
            }
        }
    }
}

```

```
        printf("Press 1 to continue...");
        scanf("%d",&num);
    }while(num==1);
    return 0;
}
```

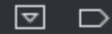
Output:



```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```



```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 2
Element(s) in the list are : 1 2 3
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 3
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 2
Element(s) in the list are : 2 3
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
Enter your choice : 2
Element(s) in the list are : 2 3 1
Press 1 to continue...0
Program ended with exit code: 0
```

All Output ↕

Q28. WAP to implement addition of polynomials using linked list

Code:

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

struct node{
    int coeff;
    int exp;
    struct node *link;
}*poly1,*poly2,*polysum;

void create_node(int c, int p, struct node **temp){
    struct node *r, *z;
    z = *temp;
    if(z == NULL){
        r=(struct node*)malloc(sizeof(struct node));
        r->coeff = c;
        r->exp = p;
        *temp = r;
        r->link = (struct node*)malloc(sizeof(struct node));
        r = r->link;
        r->link = NULL;
    }
    else{
        r->coeff = c;
        r->exp = p;
        r->link = (struct node*)malloc(sizeof(struct node));
        r = r->link;
        r->link = NULL;
    }
}

void polyadd(struct node *poly1, struct node *poly2, struct node *polysum){
    while(poly1->link!=NULL && poly2->link!=NULL)
    {
        if(poly1->exp > poly2->exp)
        {
            polysum->exp = poly1->exp;
            polysum->coeff = poly1->coeff;
            poly1 = poly1->link;
        }
        else if(poly1->exp < poly2->exp)
        {
            polysum->exp = poly2->exp;
            polysum->coeff = poly2->coeff;
            poly2 = poly2->link;
        }
    }
```

```

else
{
    polysum->exp = poly1->exp;
    polysum->coeff = poly1->coeff+poly2->coeff;
    poly1 = poly1->link;
    poly2 = poly2->link;
}

polysum->link = (struct node *)malloc(sizeof(struct node));
polysum = polysum->link;
polysum->link = NULL;
}
while(poly1->link!=NULL || poly2->link!=NULL){
    if(poly1->link)
    {
        polysum->exp = poly1->exp;
        polysum->coeff = poly1->coeff;
        poly1 = poly1->link;
    }
    if(poly2->link)
    {
        polysum->exp = poly2->exp;
        polysum->coeff = poly2->coeff;
        poly2 = poly2->link;
    }
    polysum->link = (struct node*)malloc(sizeof(struct node));
    polysum = polysum->link;
    polysum->link = NULL;
}
}

void display(struct node* ptr){
    while(ptr->link != NULL)
    {
        printf("%dx^%d", ptr->coeff, ptr->exp);
        ptr = ptr->link;
        if(ptr->link != NULL)
            printf(" + ");
    }
}

int main()
{
    create_node(1,3,&poly1);
    create_node(4,2,&poly1);
    create_node(2,1,&poly1);
    create_node(5,0,&poly1);

    create_node(8,2,&poly2);
    create_node(2,1,&poly2);

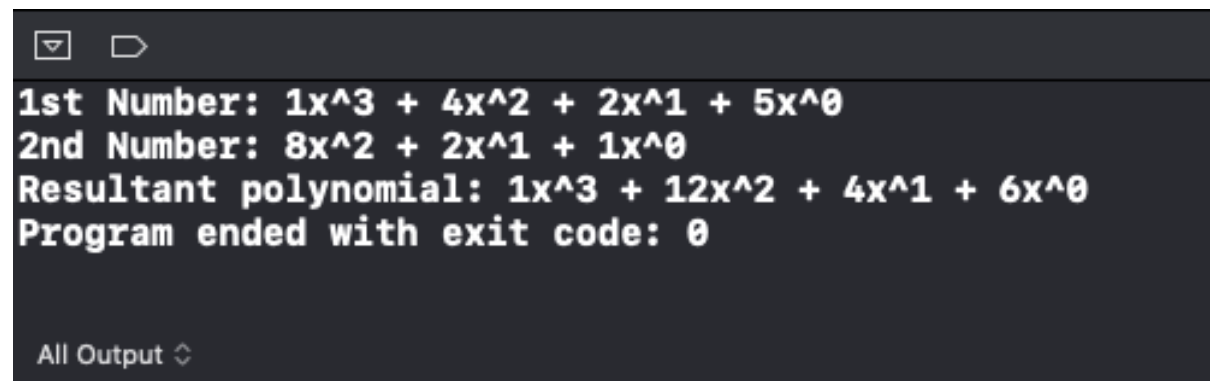
```

```
create_node(1,0,&poly2);

printf("1st Number: ");
display(poly1);
printf("\n2nd Number: ");
display(poly2);
polysum = (struct node *)malloc(sizeof(struct node));
polyadd(poly1, poly2, polysum);
printf("\nResultant polynomial: ");
display(polysum);
printf("\n");

return 0;
}
```

Output:

A screenshot of a terminal window with a dark background and light gray text. The terminal shows the output of a program: "1st Number: 1x^3 + 4x^2 + 2x^1 + 5x^0", "2nd Number: 8x^2 + 2x^1 + 1x^0", "Resultant polynomial: 1x^3 + 12x^2 + 4x^1 + 6x^0", and "Program ended with exit code: 0". At the bottom, there is a label "All Output" with a small upward and downward arrow icon.

```
1st Number: 1x^3 + 4x^2 + 2x^1 + 5x^0
2nd Number: 8x^2 + 2x^1 + 1x^0
Resultant polynomial: 1x^3 + 12x^2 + 4x^1 + 6x^0
Program ended with exit code: 0

All Output ↕
```


Q29. WAP to implement Doubly linked list.

Code:

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

int length(void);
struct node{
    int data;
    struct node *next;
    struct node *prev;
}*root;

void add_end(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    right=root;
    while(right->next!=NULL)
        right=right->next;
    temp->prev=right;
    right->next=temp;
    right=temp;
    right->next=NULL;
}

void add_begin( int num ){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    if (root== NULL)
    {
        root=temp;
        root->next=NULL;
        root->prev=NULL;
    }
    else
    {
        root->prev=temp;
        temp->next=root;
        root=temp;
        root->prev=NULL;
    }
}

void addafter(int num, int loc){
    int i;
    struct node *temp,*left=NULL,*right;
    right=root;
```

```

for(i=0;i<loc;i++)
{
    left=right;
    right=right->next;
}
temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
temp->prev=left;
left->next=temp;
left=temp;
left->next=right;
right->prev=left;
}

```

```

void insert(int num){
    int c=0;
    struct node *temp;
    temp=root;
    if(temp==NULL)
    {
        add_begin(num);
    }
    else
    {
        while(temp!=NULL)
        {
            if(temp->data<num)
                c++;
            temp=temp->next;
        }
        if(c==0)
            add_begin(num);
        else if(c<length())
            addafter(num,c);
        else
            add_end(num);
    }
}

```

```

int delete(int num){
    struct node *temp, *left = NULL;
    temp=root;
    while(temp!=NULL)
    {
        if(temp->data==num)
        {
            if(temp==root)
            {
                root=temp->next;
                root->prev=NULL;
            }
        }
    }
}

```

```

        free(temp);
        return 1;
    }
    else
    {
        if(temp->next==NULL){
            left->next=temp->next;
            free(temp);
            return 1;
        }
        else{
            temp->next->prev=left;
            left->next=temp->next;
            free(temp);
            return 1;
        }
    }
}
else
{
    left=temp;
    temp=temp->next;
}
}
return 0;
}

```

```

void display(){
    struct node *r;
    r=root;
    while(r!=NULL)
    {
        printf("%d ",r->data);
        r=r->next;
    }
// while(r!=NULL)
// {
//     printf("%d ",r->data);
//     r=r->prev;
// }
    printf("\n");
}

```

```

int length(){
    struct node *n;
    int c=0;
    n=root;
    while(n!=NULL)
    {
        n=n->next;
    }
}

```

```

        c++;
    }
    return c;
}

```

```

void reverse(){
    struct node *p=root, *q=NULL, *r=NULL;
    while(p)
    {
        r=q;
        q=p;
        p=p->next;
        q->prev=p;
        q->next=r;
    }
    root=q;
}

```

```

int main(){
    int i,num;
    root=NULL;
    do{
        printf("List Operations\n");
        printf("=====\\n");
        printf("1.Insert\\n");
        printf("2.Display\\n");
        printf("3.Delete\\n");
        printf("4.Reverse\\n");
        printf("Press 0 to exit\\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            exit(1);
        } else {
            switch(i)
            {
                case 1:
                    printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;
                case 2:
                    if(root==NULL)
                    {
                        printf("List is Empty\\n");
                    }
                    else
                    {
                        printf("Element(s) in the list are : ");
                        display();
                    }
                }
            }
        }
    } while (i != 0);
}

```

```

    }
    break;

case 3:
    if(root==NULL)
        printf("List is Empty. UNDERFLOW!\n");
    else{
        printf("Enter the element to be delete : ");
        scanf("%d",&num);
        if(delete(num))
            printf("Deleted Successfully\n");
        else
            printf("%d not found in the list\n",num);
    }
    break;
case 4:
    if(root==NULL)
        printf("List is Empty. UNDERFLOW!\n");
    else{
        reverse();
        printf("Linked list has been reversed\n");
    }
    break;
default: printf("Invalid option\n");
}
}
printf("Press 1 to continue...");
scanf("%d",&num);
printf("\n");
}while(num==1);
return 0;
}

```

Output:

☐ ☐

```
List Operations
=====
1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```

```
List Operations
=====
1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 2
```

All Output ↕

4.Reverse
Press 0 to exit
Enter your choice : 2
Element(s) in the list are : 1 2 3
Press 1 to continue...1

List Operations

=====

1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 3
Enter the element to be delete : 2
Deleted Successfully
Press 1 to continue...1

List Operations

=====

1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 4
Linked list has been reversed
Press 1 to continue...1

List Operations

=====

1.Insert
2.Display
3.Delete
4.Reverse
Press 0 to exit
Enter your choice : 2
Element(s) in the list are : 3 1
Press 1 to continue...0

Program ended with exit code: 0

Q30. WAP to implement Singly circular linked list

Code:

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

int length(void);
struct node{
    int data;
    struct node *next;
}*root,*last;

void add_end(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    right=root;
    do
        right=right->next;
    while(right->next!=root);
    right->next=temp;
    right=temp;
    right->next=root;
    last=right;
}

void add_begin( int num ){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    if (root==NULL)
    {
        root=temp;
        root->next=root;
        last=root;
    }
    else
    {
        temp->next=root;
        root=temp;
        last->next=root;
    }
}

void addafter(int num, int loc){
    int i;
    struct node *temp,*left=NULL,*right;
    right=root;
    for(i=0;i<loc;i++)
```



```

    {
        left=right;
        right=right->next;
    }
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    left->next=temp;
    left=temp;
    left->next=right;
}

```

```

void insert(int num){
    int c=0;
    struct node *temp;
    temp=root;
    if(temp==NULL)
    {
        add_begin(num);
    }
    else
    {
        do
        {
            if(temp->data<num)
                c++;
            temp=temp->next;
        } while(temp!=root);

        if(c==0)
            add_begin(num);
        else if(c<length())
            addafter(num,c);
        else
            add_end(num);
    }
}

```

```

int delete(int num){
    struct node *temp, *left = NULL;
    temp=root;
    do
    {
        if(temp->data==num)
        {
            if(temp==root)
            {
                root=temp->next;
                printf("%d\n",last->data);
                last->next=root;
                free(temp);
            }
        }
    }
}

```

```

        return 1;
    }
    else
    {
        left->next=temp->next;
        free(temp);
        return 1;
    }
}
else
{
    left=temp;
    temp=temp->next;
}
}while(temp!=root);
return 0;
}

```

```

void display(){
    struct node *r;
    r=root;
    do
    {
        printf("%d ",r->data);
        r=r->next;
    }while(r!=root);
    //printf("%d",r->data);
    printf("\n");
}

```

```

int length(){
    struct node *n;
    int c=0;
    n=root;
    do
    {
        n=n->next;
        c++;
    }while(n!=root);
    return c;
}

```

```

int main(){
    int i,num;
    root=NULL;
    do{
        printf("List Operations\n");
        printf("===== \n");
        printf("1.Insert\n");
        printf("2.Display\n");
    }
}

```

```

printf("3.Delete\n");
printf("Press 0 to exit\n");
printf("Enter your choice : ");
scanf("%d",&i);
if(i<=0){
    exit(1);
} else {
    switch(i)
    {
        case 1:
            printf("Enter the number to insert : ");
            scanf("%d",&num);
            insert(num);
            break;
        case 2:
            if(root==NULL)
            {
                printf("List is Empty\n");
            }
            else
            {
                printf("Element(s) in the list are : ");
                display();
                printf("count: %d\n",length());
            }
            break;

        case 3:
            if(root==NULL)
                printf("List is Empty. UNDERFLOW!\n");
            else{
                printf("Enter the element to be delete : ");
                scanf("%d",&num);
                if(delete(num))
                    printf("Deleted Successfully\n");
                else
                    printf("%d not found in the list\n",num);
            }
            break;

        default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
printf("\n");
}while(num==1);
return 0;
}

```

Output:

```

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 2
All Output ↕
```

```
Element(s) in the list are : 1 2 3  
count: 3  
Press 1 to continue...1
```

```
List Operations
```

```
=====
```

```
1.Insert
```

```
2.Display
```

```
3.Delete
```

```
Press 0 to exit
```

```
Enter your choice : 3
```

```
Enter the element to be delete : 3
```

```
Deleted Successfully
```

```
Press 1 to continue...1
```

```
List Operations
```

```
=====
```

```
1.Insert
```

```
2.Display
```

```
3.Delete
```

```
Press 0 to exit
```

```
Enter your choice : 2
```

```
Element(s) in the list are : 1 2
```

```
count: 2
```

```
Press 1 to continue...0
```

```
Program ended with exit code: 0|
```

All Output ↕

Q31. WAP to implement Doubly circular linked list

Code:

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

int length(void);
struct node{
    int data;
    struct node *next;
    struct node *prev;
}*root,*last;

void add_end(int num){
    struct node *temp,*right;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    right=root;
    do
        right=right->next;
    while(right->next!=root);
    temp->prev=right;
    right->next=temp;
    right=temp;
    right->next=root;
    last=right;
}

void add_begin( int num ){
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    if (root== NULL)
    {
        root=temp;
        root->next=root;
        root->prev=root;
        last=root;
    }
    else
    {
        root->prev=temp;
        temp->next=root;
        root=temp;
        root->prev=last;
        last->next=root;
    }
}
```

```

void addafter(int num, int loc){
    int i;
    struct node *temp,*left=NULL,*right;
    right=root;
    for(i=0;i<loc;i++)
    {
        left=right;
        right=right->next;
    }
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    temp->prev=left;
    left->next=temp;
    left=temp;
    left->next=right;
    right->prev=left;
}

```

```

void insert(int num){
    int c=0;
    struct node *temp;
    temp=root;
    if(temp==NULL)
    {
        add_begin(num);
    }
    else
    {
        do
        {
            if(temp->data<num)
                c++;
            temp=temp->next;
        } while(temp!=root);

        if(c==0)
            add_begin(num);
        else if(c<length())
            addafter(num,c);
        else
            add_end(num);
    }
}

```

```

int delete(int num){
    struct node *temp, *left = NULL;
    temp=root;
    do
    {
        if(temp->data==num)

```

```

{
    if(temp==root)
    {
        root=temp->next;
        root->prev=last;
        last->next=root;
        free(temp);
        return 1;
    }
    else
    {
        if(temp->next==NULL){
            left->next=temp->next;
            free(temp);
            return 1;
        }
        else{
            temp->next->prev=left;
            left->next=temp->next;
            free(temp);
            return 1;
        }
    }
}
else
{
    left=temp;
    temp=temp->next;
}
}while(temp!=root);
return 0;
}

```

```

void display(){
    struct node *r;
    r=root;
    do
    {
        printf("%d ",r->data);
        r=r->next;
    }while(r!=root);
    printf("\n");
}

```

```

int length(){
    struct node *n;
    int c=0;
    n=root;
    do
    {

```



```

    n=n->next;
    c++;
} while(n!=root);
return c;
}

int main(){
    int i,num;
    root=NULL;
    do{
        printf("List Operations\n");
        printf("=====\\n");
        printf("1.Insert\\n");
        printf("2.Display\\n");
        printf("3.Delete\\n");
        printf("Press 0 to exit\\n");
        printf("Enter your choice : ");
        scanf("%d",&i);
        if(i<=0){
            exit(1);
        } else {
            switch(i)
            {
                case 1:
                    printf("Enter the number to insert : ");
                    scanf("%d",&num);
                    insert(num);
                    break;
                case 2:
                    if(root==NULL)
                    {
                        printf("List is Empty\\n");
                    }
                    else
                    {
                        printf("Element(s) in the list are : ");
                        display();
                    }
                    break;
                case 3:
                    if(root==NULL)
                        printf("List is Empty. UNDERFLOW!\\n");
                    else{
                        printf("Enter the element to be delete : ");
                        scanf("%d",&num);
                        if(delete(num))
                            printf("Deleted Successfully\\n");
                        else
                            printf("%d not found in the list\\n",num);
                    }
            }
        }
    } while(i!=0);
}
```

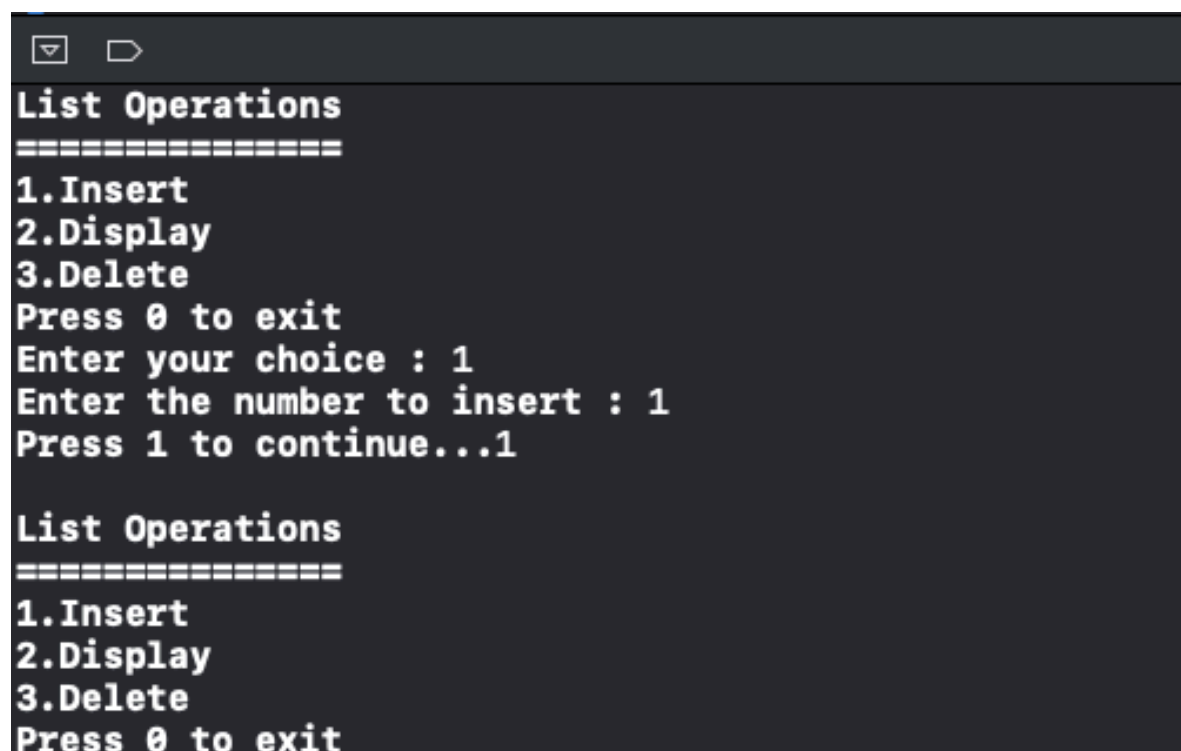
```

        }
        break;

        default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
printf("\n");
}while(num==1);
return 0;
}

```

Output:



```

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Delete
Press 0 to exit

```



```
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 2
Element(s) in the list are : 1 2 3
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 3
Enter the element to be delete : 1
Deleted Successfully
Press 1 to continue...1
```

List Operations

=====

```
1.Insert
2.Display
3.Delete
Press 0 to exit
Enter your choice : 2
Element(s) in the list are : 2 3
Press 1 to continue...0
```

Q32. WAP to implement Binary Search Tree.

Code:

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

struct node{
    int data;
    struct node *left;
    struct node *right;
}*root=NULL;

struct node *newNode(int num) {
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = num;
    temp->left = temp->right = NULL;
    if(root==NULL){
        root=temp;
    }
    return temp;
}

struct node *insert(int num, struct node *curr){

    if(curr==NULL)
        return newNode(num);

    if(num>curr->data)
        curr->right=insert(num, curr->right);

    else
        curr->left=insert(num, curr->left);

    return curr;
}

void display(struct node *p)           //Inorder
{
    if(p)
    {
        display(p->left);
        printf("%d ",p->data);
        display(p->right);
    }
}

int search(int num, struct node *p){
```

```

    if(p==NULL)
        return 0;
    else if(p->data==num)
        return p->data;
    else if(p->data > num)
        return search(num, p->left);
    else if(p->data < num)
        return search(num, p->right);
    else
        return 0;
}

struct node *minValueNode(struct node *node) {
    struct node *current = node;

    while (current && current->left != NULL)
        current = current->left;

    return current;
}

struct node* delete (int val, struct node* p){
    if(p->data>val)
        p->left = delete(val, p->left);
    else if(p->data<val)
        p->right = delete(val, p->right);
    else{
        if (p->left == NULL){
            struct node *temp = p->right;
            free(p);
            return temp;
        }
        else if (p->right == NULL) {
            struct node *temp = p->left;
            free(p);
            return temp;
        }
        struct node *temp = minValueNode(p->right);

        p->data = temp->data;

        p->right = delete(temp->data,p->right);
    }
    return root;
}

int main(){
    int i,num;
    do{
        printf("List Operations\n");
        printf("=====\n");
    }

```

```

printf("1.Insert\n");
printf("2.Display\n");
printf("3.Search\n");
printf("4.Delete\n");
printf("Press 0 to exit\n");
printf("Enter your choice : ");
scanf("%d",&i);
if(i<=0){
    exit(1);
} else {
    switch(i)
    {
        case 1:
            printf("Enter the number to insert : ");
            scanf("%d",&num);
            insert(num,root);
            break;

        case 2:
            printf("Element(s) in the list are : ");
            display(root);
            printf("\n");
            break;

        case 3:
            printf("Enter the number to be searched : ");
            scanf("%d",&num);
            if(search(num,root))
                printf("%d was found in the binary tree.\n",num);
            else
                printf("%d was not found in the binary tree.\n",num);
            break;

        case 4:
            printf("Enter the element to be delete : ");
            scanf("%d",&num);
            if(delete(num,root))
                printf("Deleted Successfully\n");
            else
                printf("%d not found in the list\n",num);
            break;

        default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
printf("\n");
}while(num==1);
return 0;}

```

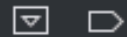
Output:

```

List Operations
=====
1.Insert
2.Display
3.Search
4.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Search
4.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 2
Press 1 to continue...1

List Operations
=====
1.Insert
2.Display
3.Search
4.Delete
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 3
Press 1 to continue...1
```



List Operations

=====

1.Insert

2.Display

3.Search

4.Delete

Press 0 to exit

Enter your choice : 1

Enter the number to insert : 4

Press 1 to continue...1

List Operations

=====

1.Insert

2.Display

3.Search

4.Delete

Press 0 to exit

Enter your choice : 2

Element(s) in the list are : 1 2 3 4

Press 1 to continue...1

List Operations

=====

1.Insert

2.Display

3.Search

4.Delete

Press 0 to exit

Enter your choice : 3

Enter the number to be searched : 3

3 was found in the binary tree.

Press 1 to continue...1

List Operations

=====

- 1.Insert
- 2.Display
- 3.Search
- 4.Delete

Press 0 to exit

Enter your choice : 4

Enter the element to be delete : 2

Deleted Successfully

Press 1 to continue...1

List Operations

=====

- 1.Insert
- 2.Display
- 3.Search
- 4.Delete

Press 0 to exit

Enter your choice : 2

Element(s) in the list are : 1 3 4

Press 1 to continue...0

Program ended with exit code: 0

All Output ↕

Q33. WAP to implement Binary tree traversal and count.

Code:

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

struct node{
    int data;
    struct node *left;
    struct node *right;
}*root=NULL;

struct node *newNode(int num) {
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = num;
    temp->left = temp->right = NULL;
    if(root==NULL){
        root=temp;
    }
    return temp;
}

struct node *insert(int num, struct node *curr){

    if(curr==NULL)
        return newNode(num);

    if(num>curr->data)
        curr->right=insert(num, curr->right);

    else
        curr->left=insert(num, curr->left);

    return curr;
}

void preOrder(struct node *p)          //Preorder
{
    if(p)
    {
        printf("%d ",p->data);
        preOrder(p->left);
        preOrder(p->right);
    }
}

void postOrder(struct node *p)        //Postorder
{

```

```

    if(p)
    {
        postOrder(p->left);
        postOrder(p->right);
        printf("%d ",p->data);
    }
}

void inOrder(struct node *p)          //Inorder
{
    if(p)
    {
        inOrder(p->left);
        printf("%d ",p->data);
        inOrder(p->right);
    }
}

int countNodes(struct node *p){
    int count=1;
    if(p){
        count+=countNodes(p->left);
        count+=countNodes(p->right);
        return count;
    }
    else
        return 0;
}

int countLeaf(struct node *p){
    if(p==NULL)
        return 0;
    if(p->left==NULL && p->right==NULL)
        return 1;
    else
        return countLeaf(p->left)+countLeaf(p->right);
}

int main(){
    int i,num,count;
    do{
        printf("List of Operations\n");
        printf("=====\\n");
        printf("1.Insert\n");
        printf("2.Preorder\n");
        printf("3.Postorder\n");
        printf("4.Inorder\n");
        printf("5.Number of nodes\n");
        printf("6.Number of leaves\n");
        printf("Press 0 to exit\n");
    }
}

```

```

printf("Enter your choice : ");
scanf("%d",&i);
if(i<=0){
    exit(1);
} else {
    switch(i)
    {
        case 1:
            printf("Enter the number to insert : ");
            scanf("%d",&num);
            insert(num,root);
            break;

        case 2:
            printf("Element(s) in the list are : ");
            preOrder(root);
            printf("\n");
            break;

        case 3:
            printf("Element(s) in the list are : ");
            postOrder(root);
            printf("\n");
            break;

        case 4:
            printf("Element(s) in the list are : ");
            inOrder(root);
            printf("\n");
            break;

        case 5:
            count=countNodes(root);
            printf("The number of nodes is %d",count);
            printf("\n");
            break;

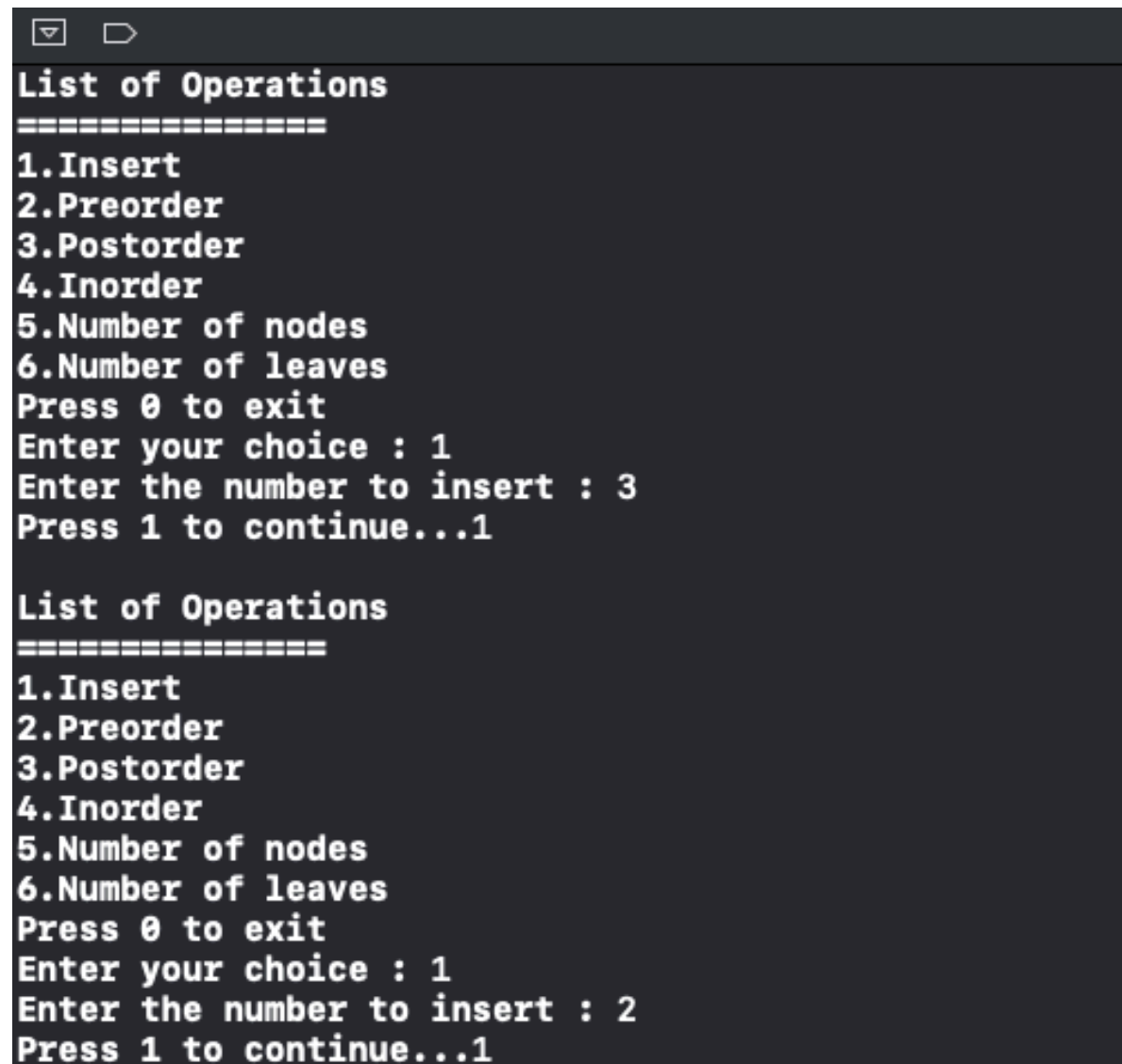
        case 6:
            count=countLeaf(root);
            printf("The number of nodes is %d",count);
            printf("\n");
            break;

        default: printf("Invalid option\n");
    }
}
printf("Press 1 to continue...");
scanf("%d",&num);
printf("\n");
}while(num==1);

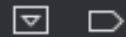
```

```
    return 0;  
}
```

Output:



```
List of Operations  
=====  
1.Insert  
2.Preorder  
3.Postorder  
4.Inorder  
5.Number of nodes  
6.Number of leaves  
Press 0 to exit  
Enter your choice : 1  
Enter the number to insert : 3  
Press 1 to continue...1  
  
List of Operations  
=====  
1.Insert  
2.Preorder  
3.Postorder  
4.Inorder  
5.Number of nodes  
6.Number of leaves  
Press 0 to exit  
Enter your choice : 1  
Enter the number to insert : 2  
Press 1 to continue...1
```



List of Operations

=====

```
1.Insert
2.Preorder
3.Postorder
4.Inorder
5.Number of nodes
6.Number of leaves
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 4
Press 1 to continue...1
```

List of Operations

=====

```
1.Insert
2.Preorder
3.Postorder
4.Inorder
5.Number of nodes
6.Number of leaves
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 1
Press 1 to continue...1
```

List of Operations

=====

```
1.Insert
2.Preorder
3.Postorder
4.Inorder
5.Number of nodes
6.Number of leaves
Press 0 to exit
Enter your choice : 1
Enter the number to insert : 5
Press 1 to continue...1
```

☐ ➤

List of Operations

=====

1.Insert

2.Preorder

3.Postorder

4.Inorder

5.Number of nodes

6.Number of leaves

Press 0 to exit

Enter your choice : 2

Element(s) in the list are : 3 2 1 4 5

Press 1 to continue...1

List of Operations

=====

1.Insert

2.Preorder

3.Postorder

4.Inorder

5.Number of nodes

6.Number of leaves

Press 0 to exit

Enter your choice : 3

Element(s) in the list are : 1 2 5 4 3

Press 1 to continue...1

List of Operations

=====

1.Insert

2.Preorder

3.Postorder

4.Inorder

5.Number of nodes

6.Number of leaves

Press 0 to exit

Enter your choice : 4

Element(s) in the list are : 1 2 3 4 5

Press 1 to continue...1

List of Operations

=====

1.Insert

2.Preorder

3.Postorder

4.Inorder

5.Number of nodes

6.Number of leaves

Press 0 to exit

Enter your choice : 5

The number of nodes is 5

Press 1 to continue...1

List of Operations

=====

1.Insert

2.Preorder

3.Postorder

4.Inorder

5.Number of nodes

6.Number of leaves

Press 0 to exit

Enter your choice : 6

The number of nodes is 2

Press 1 to continue...0

Program ended with exit code: 0

All Output ↕

Q34. WAP to implement Bubble sort.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void bubbleSort(int arr[], int s)
{
    int temp;
    for(int i=0; i<s-1; i++){
        for(int j=0; j<(s-i)-1; j++)
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

int main(){
    int size;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);
    int arr[size];

    create(arr,size);
    bubbleSort(arr, size);
    display(arr, size);

    return 0;
}
```

Output:



Enter the number of elements for the array: 5

Enter the elements of the array.

6

2

3

9

1

The array is : 1 2 3 6 9

Program ended with exit code: 0

All Output

Q35. WAP to implement Selection sort.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void selectionSort(int arr[], int s)
{
    int temp,min;
    for (int i=0; i<s-1; i++){
        min=arr[i];
        for (int j=i+1; j<s; j++)
            if (arr[j]<min)
            {
                min=arr[j];
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

int main(){
    int size;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);
    int arr[size];

    create(arr,size);
    selectionSort(arr, size);
    display(arr, size);

    return 0;
}
```

Output:

```
Enter the number of elements for the array: 5
Enter the elements of the array.
6
3
2
8
7
The array is : 2 3 6 7 8
Program ended with exit code: 0

All Output ↕
```

Q36. WAP to implement Insertion sort.

Code:

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

```
void create(int arr[], int s){  
    printf("Enter the elements of the array.\n");  
    for(int i=0; i<s; i++)  
        scanf("%d",&arr[i]);  
}
```

```
void insertionSort(int arr[], int s)  
{  
    int j,temp;  
    for (int i=1; i<s; i++)  
    {  
        temp=arr[i];  
        j=i-1;  
  
        while(arr[j]>temp){  
            arr[j+1]=arr[j];  
            j--;  
        }  
        arr[j+1]=temp;  
    }  
}
```

```
void display(int arr[], int s){  
    printf("The array is :");  
    for(int i=0; i<s; i++)  
        printf(" %d",arr[i]);  
}
```

```
    printf("\n");
}

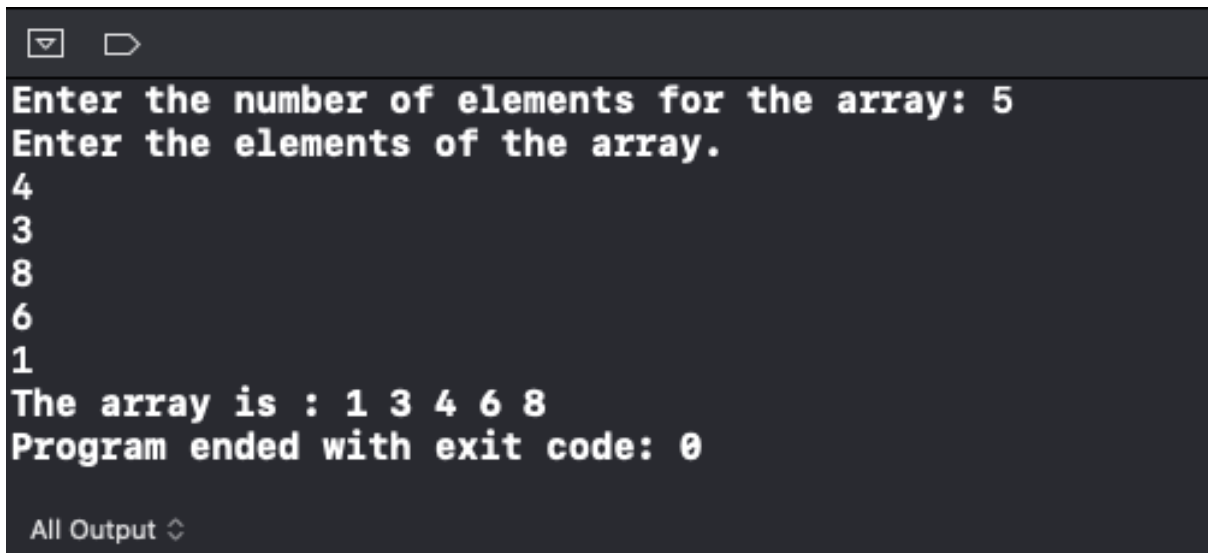
int main(){
    int size;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);
    int arr[size];

    create(arr,size);
    insertionSort(arr, size);
    display(arr, size);

    return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The window has a title bar with a dropdown arrow and a close button. The output text is as follows:

```
Enter the number of elements for the array: 5
Enter the elements of the array.
4
3
8
6
1
The array is : 1 3 4 6 8
Program ended with exit code: 0
```

At the bottom of the terminal, there is a label "All Output" followed by a small up and down arrow icon.

Q37. WAP to implement Shell sort.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void shellSort(int arr[], int s)
{
    int temp,j;
    for (int gap=s/2; gap>0; gap/=2){
        for (int i=gap; i<s; i++){
            temp=arr[i];
            for(j=i; j>=gap && arr[j-gap]>temp; j-=gap)
                arr[j]=arr[j-gap];
            arr[j]=temp;
        }
    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

int main(){
    int size;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);
    int arr[size];

    create(arr,size);
    shellSort(arr, size);
    display(arr, size);

    return 0;
}
```

Output:

```
Enter the number of elements for the array: 5
Enter the elements of the array.
5
4
3
8
0
The array is : 0 3 4 5 8
Program ended with exit code: 0

All Output ↕
```

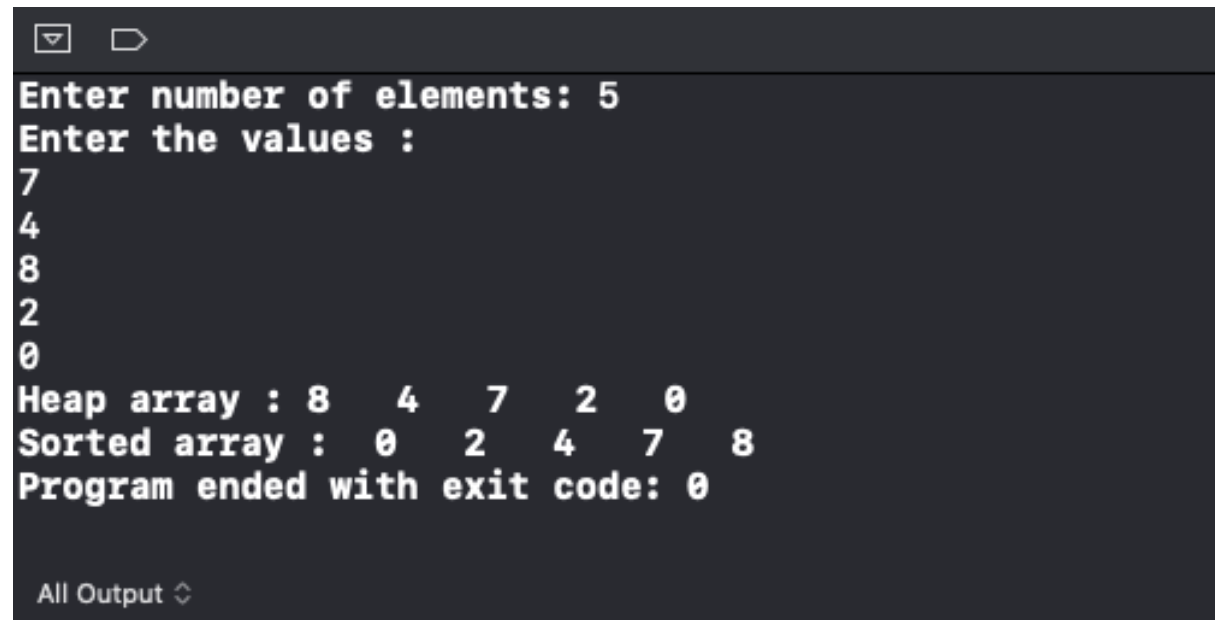

Q38. WAP to implement Max heap.

Code:

```
#include <stdio.h>
int main()
{
    int arr[10], no, i, j, c, heap_root, temp;
    printf("Enter number of elements: ");
    scanf("%d", &no);
    printf("Enter the values : \n");
    for(i=0;i<no;i++)
        scanf("%d", &arr[i]);
    for(i=1;i<no;i++)
    {
        c = i;
        do
        {
            heap_root=(c-1)/2;
            if (arr[heap_root] < arr[c])
            {
                temp = arr[heap_root];
                arr[heap_root] = arr[c];
                arr[c] = temp;
            }
            c = heap_root;
        } while (c != 0);
    }
    printf("Heap array : ");
    for(i=0;i<no;i++)
        printf("%d\t", arr[i]);
    for(j=no-1;j>=0;j--){
        temp = arr[0];
        arr[0] = arr[j];
        arr[j] = temp;
        heap_root = 0;
        do{
            c=2* heap_root+1;
            if ((arr[c] < arr[c + 1]) && c < j-1) c++;
            if (arr[heap_root]<arr[c] && c<j)
            {
                temp = arr[heap_root];
                arr[heap_root] = arr[c];
                arr[c] = temp;
            }
            heap_root = c;
        }while(c<j);
    }
    printf("\nSorted array : ");
    for(i=0;i<no;i++)
```

```
        printf("\t%d", arr[i]);  
    printf("\n");  
    return 0;  
}
```

Output:



The screenshot shows a terminal window with a dark background and light gray text. At the top, there are two small icons: a square with a downward arrow and a rectangle with a rightward arrow. The terminal output is as follows:

```
Enter number of elements: 5  
Enter the values :  
7  
4  
8  
2  
0  
Heap array : 8    4    7    2    0  
Sorted array : 0    2    4    7    8  
Program ended with exit code: 0
```

At the bottom left of the terminal window, there is a label "All Output" followed by a small upward and downward arrow icon.

Q39. WAP to implement Heapify.

Code:

```
#include<stdio.h>

void swap(int *a,int *b){
    int tmp=*a;
    *a=*b;
    *b=tmp;
}

void heapify(int arr[],int n,int i){
    int largest=i;
    int l=2*i+1;
    int r=2*i+2;
    if(l<n && arr[l]>arr[largest])
        largest=l;
    if(r<n && arr[r]>arr[largest])
        largest=r;
    if(largest!=i){
        swap(&arr[largest],&arr[i]);
        heapify(arr,n,largest);
    }
}

void heapSort(int arr[],int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i=n-1; i>0; i--){
        swap(&arr[0],&arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
```

```

int arr[100];

int n;

printf("Enter number of elements: ");

scanf("%d",&n);

printf("Input array:\n");

for(int i=0;i<n;i++){

    printf("Enter element %d: ",i+1);

    scanf("%d",&arr[i]);

}

heapSort(arr,n);

printf("\nArray elements after Heap Sort(using Heapify method):\n");

for(int i=0;i<n;i++){

    printf("%d ",arr[i]);

}

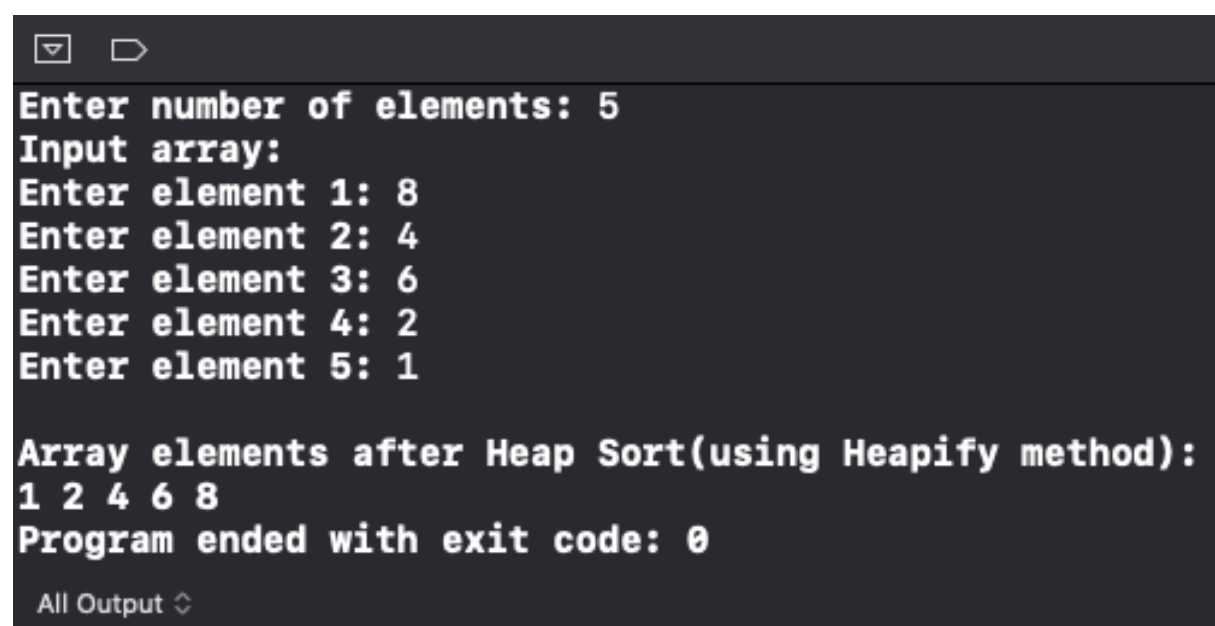
printf("\n");

return 0;

}

```

Output:



```

Enter number of elements: 5
Input array:
Enter element 1: 8
Enter element 2: 4
Enter element 3: 6
Enter element 4: 2
Enter element 5: 1

Array elements after Heap Sort(using Heapify method):
1 2 4 6 8
Program ended with exit code: 0

All Output

```

Q40. WAP to implement Quick sort.

Code:

```
#include<stdio.h>
void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void quicksort(int number[],int first,int last){
    int i, j, pivot, temp;

    if(first<last){
        pivot=first;
        i=first;
        j=last;

        while(i<j){
            while(number[i]<=number[pivot]&&i<last)
                i++;
            while(number[j]>number[pivot])
                j--;
            if(i<j){
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
            }
        }

        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        quicksort(number,first,j-1);
        quicksort(number,j+1,last);
    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

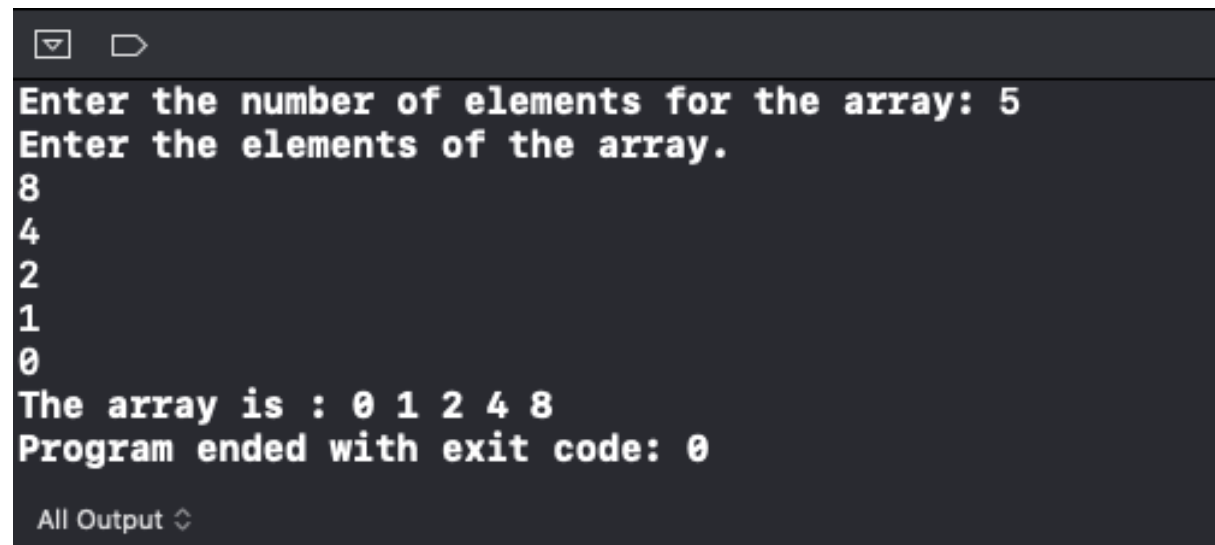
int main(){
    int size;
```

```
printf("Enter the number of elements for the array: ");
scanf("%d", &size);

int arr[size];
create(arr,size);
quicksort(arr,0,size-1);
display(arr, size);

return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The text is displayed in a light-colored, monospaced font. The output shows the program's execution flow: a prompt for the number of elements, input of 5, a prompt for the elements, input of 8 4 2 1 0, a display of the sorted array, and a final exit message.

```
Enter the number of elements for the array: 5
Enter the elements of the array.
8
4
2
1
0
The array is : 0 1 2 4 8
Program ended with exit code: 0

All Output
```

Q41. WAP to implement Merge sort.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void mergeSort(int arr[], int n){
    int temp[n],i,j,k,l1,h1,l2,h2;
    for(int size=1; size < n; size=size*2 )
    {
        l1=0;
        k=0;
        while( l1+size < n)
        {
            h1=l1+size-1;
            l2=h1+1;
            h2=l2+size-1;

            if( h2>=n )
                h2=n-1;

            i=l1;
            j=l2;

            while(i<=h1 && j<=h2 )
            {
                if( arr[i] <= arr[j] )
                    temp[k++]=arr[i++];
                else
                    temp[k++]=arr[j++];
            }

            while(i<=h1)
                temp[k++]=arr[i++];
            while(j<=h2)
                temp[k++]=arr[j++];
            l1=h2+1;
        }

        for(i=l1; k<n; i++)
            temp[k++]=arr[i];

        for(i=0;i<n;i++)
            arr[i]=temp[i];
    }
}
```

```

    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

int main(){
    int size;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);

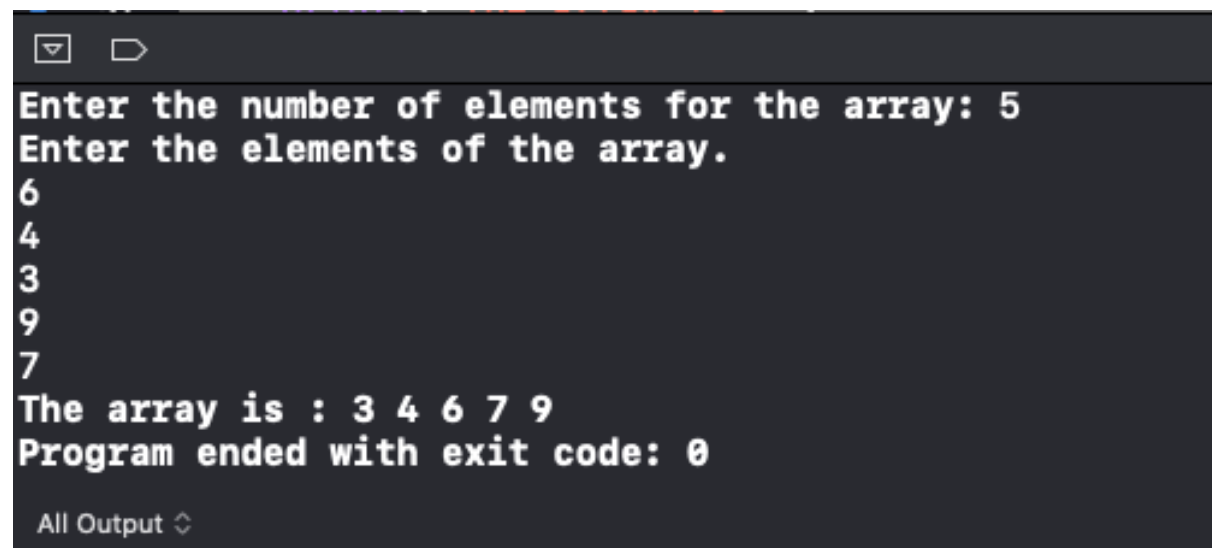
    int arr[size];

    create(arr,size);
    mergeSort(arr, size);
    display(arr, size);

    return 0;
}

```

Output:



```

Enter the number of elements for the array: 5
Enter the elements of the array.
6
4
3
9
7
The array is : 3 4 6 7 9
Program ended with exit code: 0
All Output

```


Q42. WAP to implement Linear search.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void linearSearch(int arr[], int size, int ele){
    int flag=0;
    for(int i=0; i<size; i++)
        if(ele==arr[i]){
            printf("Element %d is found at location %d.\n",ele,i+1);
            flag=1;
        }
    if(flag==0)
        printf("Element %d is not present in the array.\n",ele);
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

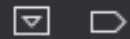
int main(){
    int size,element;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);

    int arr[size];
    create(arr,size);
    display(arr, size);

    printf("Enter the element to be searched: ");
    scanf("%d",&element);
    linearSearch(arr, size, element);
    return 0;
}
```

Output:



Enter the number of elements for the array: 5

Enter the elements of the array.

6

3

2

8

1

The array is : 6 3 2 8 1

Enter the element to be searched: 8

Element 8 is found at location 4.

Program ended with exit code: 0

All Output ↕

Q43. WAP to implement Binary search.

Code:

```
#include<stdio.h>

void create(int arr[], int s){
    printf("Enter the elements of the array.\n");
    for(int i=0; i<s; i++)
        scanf("%d",&arr[i]);
}

void binarySearch(int arr[], int left, int right, int ele){
    int flag=0;
    while(left<=right){
        int mid = left+(right-left)/2;

        if(arr[mid]==ele){
            printf("Element %d has been found at location %d.\n",ele,mid+1);
            flag=1;
        }

        if (arr[mid]<ele)
            left = mid+1;

        else
            right = mid-1;
    }

    if(flag==0){
        printf("Element %d is not present in the array.\n",ele);
    }
}

void display(int arr[], int s){
    printf("The array is :");
    for(int i=0; i<s; i++)
        printf(" %d",arr[i]);
    printf("\n");
}

int main(){
    int size,element;

    printf("Enter the number of elements for the array: ");
    scanf("%d", &size);
    printf("Enter a sorted array for binary search to work.\n");

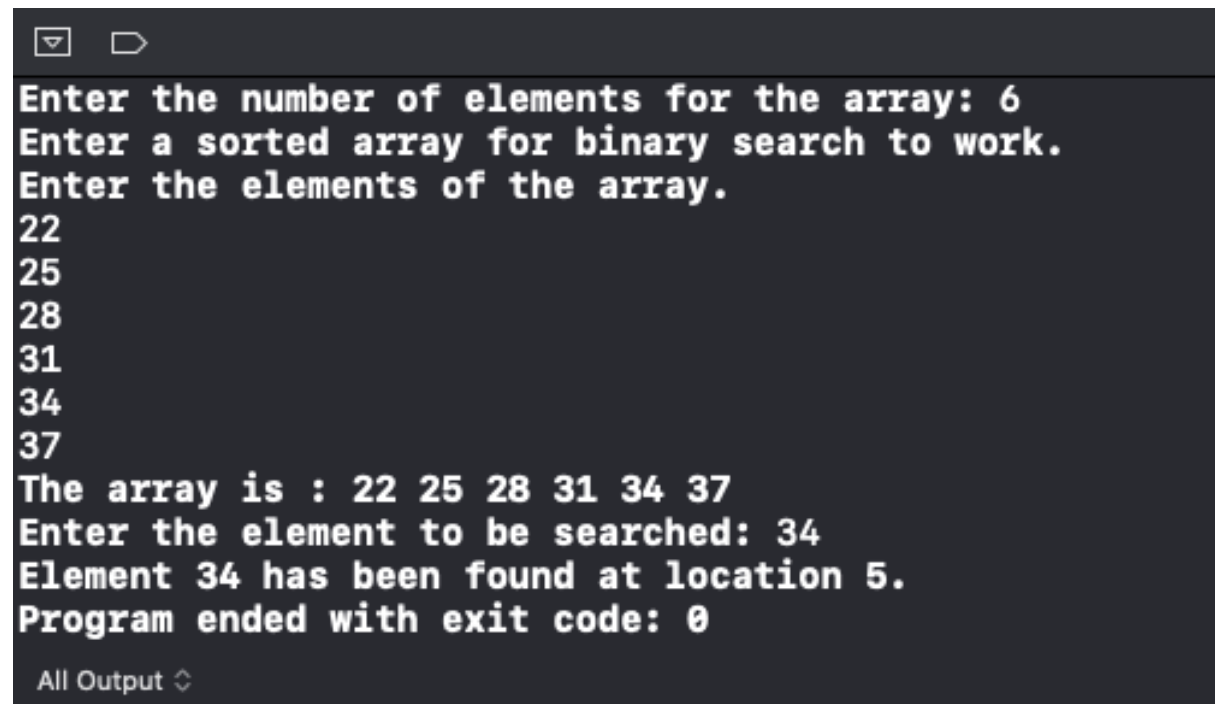
    int arr[size];
```

```
create(arr,size);
display(arr, size);

printf("Enter the element to be searched: ");
scanf("%d",&element);
binarySearch(arr, 0, size, element);

return 0;
}
```

Output:

A screenshot of a terminal window with a dark background and light-colored text. The text shows the execution of a program that prompts for array size and elements, then searches for a specific value. The output is as follows:

```
Enter the number of elements for the array: 6
Enter a sorted array for binary search to work.
Enter the elements of the array.
22
25
28
31
34
37
The array is : 22 25 28 31 34 37
Enter the element to be searched: 34
Element 34 has been found at location 5.
Program ended with exit code: 0
```

At the bottom of the terminal window, there is a label "All Output" followed by a small upward-pointing arrow icon.

Q44. WAP to implement Breadth first traversal.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3
int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph(void);
void BF_Traversal(void);
void BFS(int v);
int queue[MAX], front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue(void);
int isEmpty_queue(void);

int main(){
    create_graph();
    BF_Traversal();
    return 0;
}
void BF_Traversal() {
    int v;
    for(v=0; v<n; v++)
        state[v] = initial;
    printf("Enter Start Vertex for BFS: \n"); scanf("%d", &v);
    BFS(v);
}
void BFS(int v){
    int i;
    insert_queue(v);
    state[v] = waiting;
    while(!isEmpty_queue()){
        v = delete_queue( );
        printf("%d ",v);
        state[v] = visited;
        for(i=0; i<n; i++)
        {
            if(adj[v][i] == 1 && state[i] == initial){
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}
```

```

}
void insert_queue(int vertex){
    if(rear == MAX-1) printf("Queue Overflow\n");
    else{
        if(front == -1) front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}
int isEmpty_queue(){
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}
int delete_queue(){
    int delete_item;
    if(front == -1 || front > rear) {
        printf("Queue Underflow\n");
        exit(1);
    }
    delete_item = queue[front]; front = front+1;
    return delete_item;
}
void create_graph() {
    int count,max_edge,origin,destin;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);
    for(count=1; count<=max_edge; count++){
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d",&origin,&destin);
        if((origin == -1) && (destin == -1))
            break;
        if(origin>=n || destin>=n || origin<0 || destin<0) {
            printf("Invalid edge!\n");
            count--; }
        else {
            adj[origin][destin] = 1;
        }
    }
}
}

```

Output:

```
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0
1
Enter edge 2( -1 -1 to quit ) : 0
3
Enter edge 3( -1 -1 to quit ) : 0
4
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : 3
6
Enter edge 6( -1 -1 to quit ) : 4
7
Enter edge 7( -1 -1 to quit ) : 6
4
Enter edge 8( -1 -1 to quit ) : 6
7
Enter edge 9( -1 -1 to quit ) : 2
5
Enter edge 10( -1 -1 to quit ) : 4
5
Enter edge 11( -1 -1 to quit ) : 7
5
Enter edge 12( -1 -1 to quit ) : 7
8
Enter edge 13( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
Program ended with exit code: 0

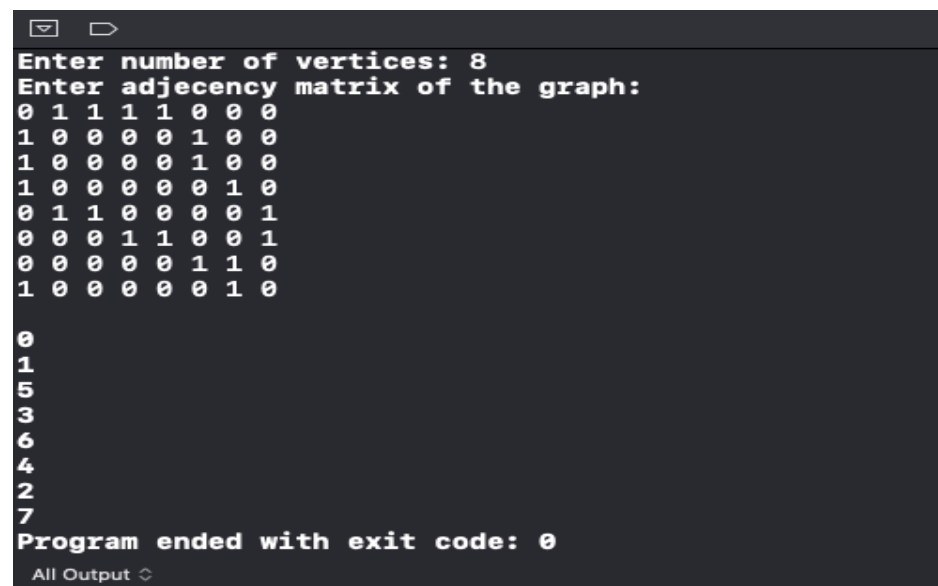
All Output ↕
```

Q45. WAP to implement Depth first traversal.

Code:

```
#include<stdio.h>
void DFS(int);
int G[10][10],visited[10],n;
int main()
{
    int i,j;
    printf("Enter number of vertices: ");
    scanf("%d",&n);
    printf("Enter adjacency matrix of the graph:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
    for(i=0;i<n;i++)
        visited[i]=0;
    DFS(0);
    printf("\n");
}
void DFS(int i)
{
    int j;
    printf("\n%d",i);
    visited[i]=1;
    for(j=0;j<n;j++)
        if(!visited[j]&&G[i][j]==1)
            DFS(j);
}
```

Output:



```
Enter number of vertices: 8
Enter adjacency matrix of the graph:
0 1 1 1 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
0 1 1 0 0 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0
1 0 0 0 0 0 1 0

0
1
5
3
6
4
2
7
Program ended with exit code: 0
All Output
```


Q46. WAP to implement Minimum spanning tree using prims algorithm.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
int prims(void);
int main()
{
    int i,j,total_cost;
    printf("Enter no. of vertices: ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
    total_cost=prims();
    printf("spanning tree matrix:\n");
    for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<n;j++)
            printf("%d\t",spanning[i][j]);
    }
    printf("\nTotal cost of spanning tree=%d",total_cost);
    printf("\n");
    return 0;
}
int prims() {
    int cost[MAX][MAX];
    int u,v,min_distance,distance[MAX],from[MAX]; int
visited[MAX],no_of_edges,i,min_cost,j; //create cost[][] matrix,spanning[][]
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            if(G[i][j]==0)
                cost[i][j]=infinity;
            else
                cost[i][j]=G[i][j];
            spanning[i][j]=0;
        }
    //initialise visited[],distance[] and from[]
    distance[0]=0;
    visited[0]=1;
    for(i=1;i<n;i++)
    {
```

```

    distance[i]=cost[0][i];
    from[i]=0;
    visited[i]=0;
}
min_cost=0; //cost of spanning tree
no_of_edges=n-1; //no. of edges to be added
while(no_of_edges>0)
{
    //find the vertex at minimum distance from the tree
    min_distance=infinity;
    for(i=1;i<n;i++)
        if(visited[i]==0&&distance[i]<min_distance)
        {
            v=i;
            min_distance=distance[i];
        }
    u=from[v];
    //insert the edge in spanning tree

    spanning[u][v]=distance[v];
    spanning[v][u]=distance[v];
    no_of_edges--;
    visited[v]=1;
    //updated the distance[] array
    for(i=1;i<n;i++)
        if(visited[i]==0&&cost[i][v]<distance[i])
        {
            distance[i]=cost[i][v]; from[i]=v;
        }
    min_cost=min_cost+cost[u][v];
}
return(min_cost);
}

```

Output:

```
Enter no. of vertices: 6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
spanning tree matrix:
0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0
Total cost of spanning tree=13
Program ended with exit code: 0
```

All Output ↕

Q47. WAP to implement Minimum spanning tree using kruskals algorithm.

Code:

```
#include<stdio.h>

#define MAX 30

typedef struct edge{
    int u,v,w;
}edge;

typedef struct edgelist {
    edge data[MAX];
    int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal(void);

int find(int belongs[],int vertexno);

void union1(int belongs[],int c1,int c2);

void sort(void);

void print(void);

int main()
{
    int i,j,total_cost;

    printf("Enter number of vertices: ");
    scanf("%d",&n);

    printf("Enter the adjacency matrix: \n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    kruskal();

    print();
}
```

```

    printf("\n");
    return 0;
}

void kruskal()
{
    int belongs[MAX],i,j,cno1,cno2; elist.n=0;
    for(i=1;i<n;i++)
        for(j=0;j<i;j++)
        {
            if(G[i][j]!=0)
            {
                elist.data[elist.n].u=i;
                elist.data[elist.n].v=j;
                elist.data[elist.n].w=G[i][j];
                elist.n++;
            }
        }
    sort();
    for(i=0;i<n;i++)
        belongs[i]=i;
    spanlist.n=0;
    for(i=0;i<elist.n;i++)
    {
        cno1=find(belongs,elist.data[i].u);
        cno2=find(belongs,elist.data[i].v);
        if(cno1!=cno2)
        {
            spanlist.data[spanlist.n]=elist.data[i];
            spanlist.n=spanlist.n+1;
            union1(belongs,cno1,cno2);
        }
    }
}

```

```

    }
}
}
int find(int belongs[],int vertexno)
{
    return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2) {
    int i;
    for(i=0;i<n;i++)
        if(belongs[i]==c2)
            belongs[i]=c1;
}

void sort()
{
    int i,j;
    edge temp;
    for(i=1;i<elist.n;i++) for(j=0;j<elist.n-1;j++) if(elist.data[j].w>elist.data[j+1].w)
    {
        temp=elist.data[j]; elist.data[j]=elist.data[j+1]; elist.data[j+1]=temp;
    }
}

void print()
{
    int i,cost=0;
    for(i=0;i<spanlist.n;i++){
        printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
        cost=cost+spanlist.data[i].w;
    }
    printf("\n\nCost of the spanning tree=%d",cost);
}

```

}

Output:

```
Enter number of vertices: 6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

2 0 1
5 3 2
1 0 3
4 1 3
5 2 4

Cost of the spanning tree=13
Program ended with exit code: 0
All Output ↕
```