//To write a C program to read two matrices and perform matrix multiplication.

```
#include<stdio.h>
int main() {
    int a[10][10], b[10][10], c[10][10], n, i, j, k;
    printf("Enter the value of N (N <= 10): ");
    scanf("%d", & n);
    printf("Enter the elements of Matrix-A: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", & a[i][j]);
        }
    }
    printf("Enter the elements of Matrix-B: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", & b[i][j]);
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            c[i][j] = 0;
            for (k = 0; k < n; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    printf("The product of the two matrices is: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%d\t", c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
Enter the value of N (N <= 10): 2
Enter the elements of Matrix-A:
2 2
2 2
Enter the elements of Matrix-B:
2 2
2 2
The product of the two matrices is:
8       8
8       8


...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to read a set of numbers from the user as input and find whether they are odd or even numbers.

```
#include <stdio.h>
int main() {
   int num[10],i;
   printf("Enter 10 numbers: ");
   for(i=0;i<10;i++)
      scanf("%d", &num[i]);
   printf("\nEven numbers are:\n");
   for(i=0;i<10;i++)
   {
      if(num[i] % 2 == 0)
      printf("%d ", num[i]);
   }
   printf("\nOdd numbers are:\n");
   for(i=0;i<10;i++)
   {
      if(num[i] % 2 != 0)
      printf("%d ", num[i]);
   }

   return 0;
}
```

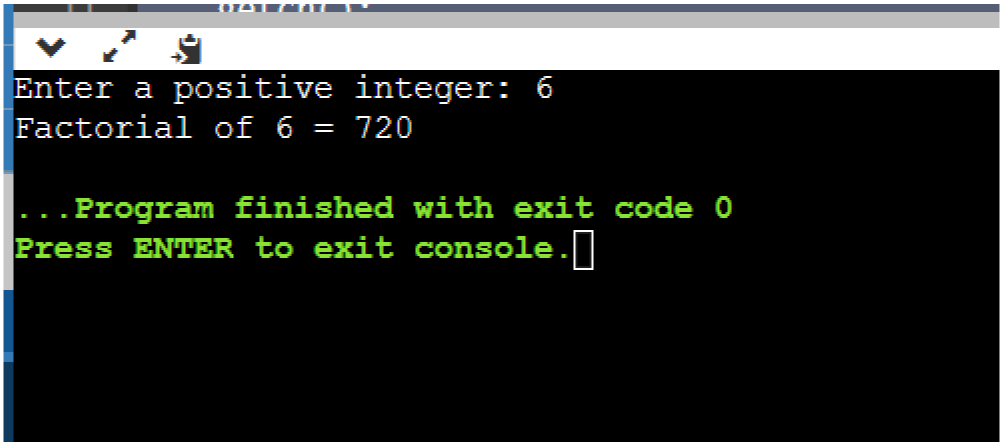
OUTPUT:

```
Enter 10 numbers:
1
2
3
4
5
6
7
8
9
10

Even numbers are:
2 4 6 8 10
Odd numbers are:
1 3 5 7 9

...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to read a number from user and calculate factorial of a given number without using Recursion

```
#include <stdio.h>
#include<conio.h>
int factorial(int);
void main()
{
   int n,fact;
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   fact= factorial(n);
   printf("Factorial of %d = %d",n, fact) ;
   getch();
}
int factorial(int num)
{
   int i=1,f=1;
   while(i<=num)
   {
      f=f*i;
      i++;
   }
   return f;
}
```

OUTPUT:


Enter a positive integer: 6
Factorial of 6 = 720
...Program finished with exit code 0
Press ENTER to exit console.

//Write a C program to read a number from user and print the fibonacci series without using Recursion

```
#include<stdio.h>
int main()
{
   int n1=0,n2=1,n3,i,num;
   printf("Number of elements:");
   scanf("%d",&num);
   //To print first 0, and 1.
   printf("\n%d %d",n1,n2);
   for(i=2; i < num; ++i)
   {
      n3=n1+n2;
      printf(" %d",n3);
      n1=n2;
      n2=n3;
   }
   return 0;
}
```

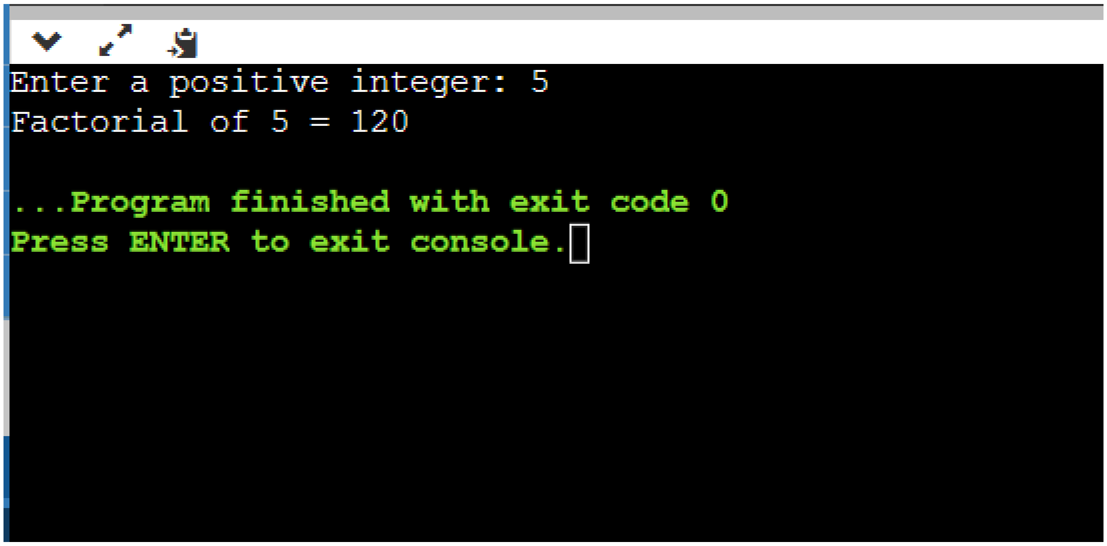
OUTPUT:

```
Number of elements:7

0 1 1 2 3 5 8

...Program finished with exit code 0
Press ENTER to exit console.
```

// Write a C program to read a number from user and calculate factorial of a given number using Recursion

```
#include <stdio.h>
#include<conio.h>
int factorial(int);
void main()
{
   int n,fact;
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   fact= factorial(n);
   printf("Factorial of %d = %d",n, fact) ;
}
int factorial(int n)
{
   if (n==1)    //Base case
      return 1;
   else
      return n*factorial(n-1);   // Inductive step

}
```

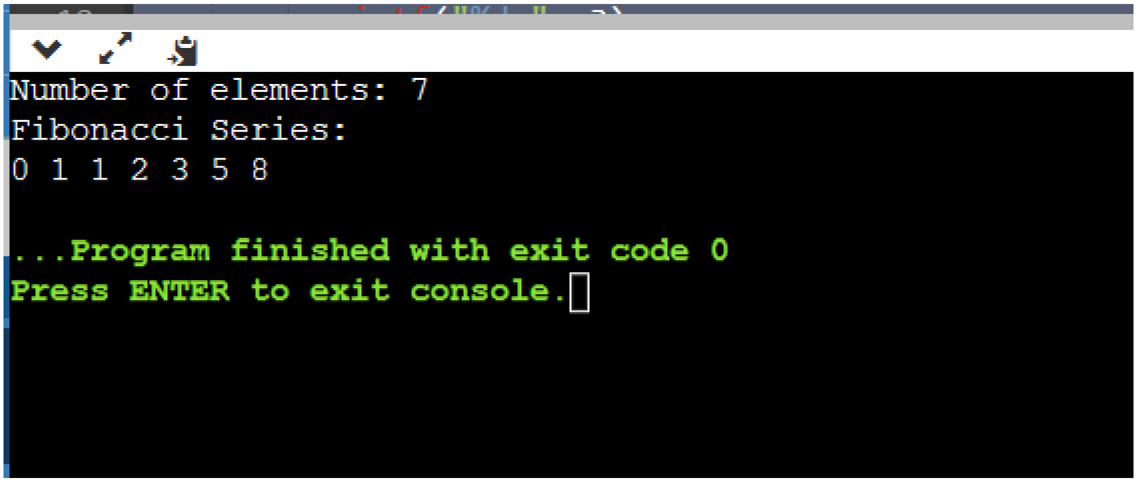
OUTPUT:


Enter a positive integer: 5
Factorial of 5 = 120
...Program finished with exit code 0
Press ENTER to exit console.

// Write a C program to read a number from the user and print the fibonacci series using Recursion.

```
#include<stdio.h>
 //Function Definition
void my_fibonacci(int n)
{
   static int n1=0,n2=1,n3;
   if(n>0)
   {
       n3 = n1 + n2;
       n1 = n2;
       n2 = n3;
       printf("%d ",n3);
       my_fibonacci(n-1);
   }
}

int main(){
   int n;
   printf("Number of elements: ");
   scanf("%d",&n);
   printf("Fibonacci Series: \n");
   printf("%d %d ",0,1);
   my_fibonacci(n-2);
   return 0;
 }
```

OUTPUT:



```
Number of elements: 7
Fibonacci Series:
0 1 1 2 3 5 8

...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to implement Array operations such as Insert, Delete and Display.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define MAX 10
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;
void main()
{
   int ch;
   char g='y';
   do
   {
      printf("\n main Menu");
      printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
      printf("\n Enter your Choice:");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1:
            create();
         break;
         case 2:
            deletion();
         break;
         case 3:
            search();
         break;
         case 4:
            insert();
         break;
         case 5:
            display();
         break;
         case 6:
            exit(0);
```

```
            break;
            default:
                printf("\n Enter the correct choice:");
        }
        printf("\n Do u want to continue:::");
        scanf("\n%c", &g);
    }
    while(g=='y'||g=='Y');
    getch();
}
void create()
{
    printf("\n Enter the number of nodes:");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter the Element:%d:",(i+1));
        scanf("%d", &b[i]);
    }
}
void deletion()
{
    printf("\n Enter the position u want to delete::");
    scanf("%d", &pos);
    if(pos>=n)
    {
        printf("\n Invalid Location::");
    }
    else
    {
        for(i=pos+1;i<n;i++)
        {
            b[i-1]=b[i];
        }
        n--;
    }
    printf("\n The Elements after deletion:");
    for(i=0;i<n;i++)
    {
        printf("\t%d", b[i]);
```

```
    }
}
void search()
{
    printf("\n Enter the Element to be searched:");
    scanf("%d", &e);
    int pos,flag=0;
    for(i=0;i<n;i++)
    {
        if(b[i]==e)
        {
            flag=1;
            pos=i;
            break;

        }

    }
    if(flag==1)
    {
        printf("Value is in the %d Position", pos+1);
    }
    else
    {
        printf("Value not found in the list");
    }
}
void insert()
{
    printf("\n Enter the position u need to insert::");
    scanf("%d", &pos);
    if(pos>=n)
    {
        printf("\n invalid Location::");
    }
    else
    {
        for(i=MAX-1;i>=pos-1;i--)
        {
            b[i+1]=b[i];
```

```
        }
        printf("\n Enter the element to insert::\n");
        scanf("%d",&p);
        b[pos]=p;
        n++;
    }
    printf("\n The list after insertion::\n");
    display();
}
void display()
{
    printf("\n The Elements of The list ADT are:");
    for(i=0;i<n;i++)
    {
        printf("\n\n%d", b[i]);
    }
}
```

OUTPUT:

```
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice:1

Enter the number of nodes:5

Enter the Element:1:10

Enter the Element:2:20

Enter the Element:3:30

Enter the Element:4:40

Enter the Element:5:50

Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice:2

Enter the position u want to delete::2

The Elements after deletion:   10      20      40      50
Do u want to continue:::y
```

```
main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice:3

Enter the Element to be searched:50
Value is in the 4 Position
Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice:3

Enter the Element to be searched:100
Value not found in the list
Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice:4

Enter the position u need to insert::1
```

```
Enter the element to insert::
30

The list after insertion::

The Elements of The list ADT are:

10

30

20

40

50
Do u want to continue:::5


...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to read a number from the user and search that number in a set of numbers using the Linear Search method.

```
#include<stdio.h>
int linear(int [],int );
int main(){
	int keyElement,i;
	int arr[10];
	printf("Enter 10 elements:");
	for(i=0;i<10;i++)
	{
	   scanf("%d",&arr[i]);
	}
	printf("Enter the element to search: ");
	scanf("%d", &keyElement);
	if(linear(arr,keyElement))
		printf("The element is found\n");
	else
		printf("The element is not found\n");
}
int linear(int arr[],int keyElement){

	for(int i=0;i<7;i++){
		if(arr[i]==keyElement){
			return 1;
		}
	}
	return 0;
}
```

OUTPUT:

```
Enter 10 elements:
67 45 23 90 11 83 72 17 32 50
Enter the element to search: 11
The element is found


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter 10 elements:
67 45 23 90 11 83 72 17 32 50
Enter the element to search: 100
The element is not found


...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to read a number from the user and search that number in a set of numbers using the Binary Search method.

```
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {       mid = (beg + end)/2;
         if(a[mid] == val)
       {
          return mid+1;
       }
          /* if the item to be searched is smaller than middle, then it can only be in left subarray */
       else if(a[mid] < val)
       {
          return binarySearch(a, mid+1, end, val);
       }
          /* if the item to be searched is greater than middle, then it can only be in right subarray */
       else
       {
          return binarySearch(a, beg, mid-1, val);
       }
   }
   return -1;
}
int main() {
  int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
  int val = 40; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = binarySearch(a, 0, n-1, val); // Store result
  printf("The elements of the array are - ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d", val);
  if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array", res);
  return 0;
}
```

OUTPUT:

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array

...Program finished with exit code 0
Press ENTER to exit console.
```

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 100
Element is not present in the array

...Program finished with exit code 0
Press ENTER to exit console.
```

//To define a singly linked list node and perform operations such as insertions and deletions dynamically

```
#include <stdio.h>
#include <conio.h>
//#include <process.h>
//#include <alloc.h>
#include <string.h>
struct node
{
   int label;
   struct node *next;
};
main()
{
   int ch, fou=0;
   int k;
   struct node *h, *temp, *head, *h1;
   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->label = -1;
   head->next = NULL;
    while(-1)
   {
      clrscr();
      printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
      printf("1->Add ");
      printf("2->Delete ");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);
      switch(ch)
      {
         /* Add a node at any intermediate location */
         case 1:
            printf("\n Enter label after which to add : ");
            scanf("%d", &k);
            h = head;
            fou = 0;
```

```
    if (h->label == k)
        fou = 1;
    while(h->next != NULL)
    {
        if (h->label == k)
        {
            fou=1;
            break;
        }
        h = h->next;
    }
    if (h->label == k)
         fou = 1;
    if (fou != 1)
         printf("Node not found\n");
    else
    {
        temp=(struct node *)(malloc(sizeof(struct node)));
        printf("Enter label for new node : ");
        scanf("%d" , &temp->label);
        temp->next = h->next;
        h->next = temp;
    }
break;
/* Delete any intermediate node */
case 2:
    printf("Enter label of node to be deleted\n");
    scanf("%d", &k);
    fou = 0;
    h = h1 = head;
    while (h->next != NULL)
    {
        h = h->next;
        if (h->label == k)
        {
            fou = 1;
            break;
        }
    }
    if (fou == 0)
```

```
            printf("Sorry Node not found\n");
        else
        {
            while (h1->next != h)
                h1 = h1->next;
            h1->next = h->next;
            free(h);
            printf("Node deleted successfully \n");
        }
    break;
    case 3:
        printf("\n\n HEAD -> ");
        h=head;
        while (h->next != NULL)
        {
            h = h->next;
            printf("%d -> ",h->label);
        }
        printf("NULL");
    break;
    case 4:
        exit(0);
}

}
}
```

OUTPUT:

```
 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 1

 Enter label after which to add : -1
Enter label for new node : 23


 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 1

 Enter label after which to add : 23
Enter label for new node : 37


 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 3


 HEAD -> 23 -> 37 -> NULL

 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 2
Enter label of node to be deleted
23
Node deleted successfully


 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 3


 HEAD -> 37 -> NULL

 SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice :
```

//To implement Stack operations such as PUSH, POP and DISPLAY using Linked List.

```
#include <stdio.h>
#include <conio.h>
//#include <process.h>
//#include <alloc.h>
struct node
{
   int label;
   struct node *next;
};
main()
{
   int ch = 0;
   int k;
   struct node *h, *temp, *head;
   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;
   while(1)
   {
      printf("\n Stack using Linked List \n");
      printf("1->Push ");
      printf("2->Pop ");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            h = head;
            temp->next = h->next;
            h->next = temp;
         break;
         case 2:
```

```
        /* Delink the first node */
        h = head->next;
        head->next = h->next;
        printf("Node %d deleted\n", h->label);
        free(h);
      break;
      case 3:
        printf("\n HEAD -> ");
        h = head;
        /* Loop till last node */
        while(h->next != NULL)
        {
          h = h->next;
          printf("%d -> ",h->label);
        }
        printf("NULL \n");
      break;
      case 4:
        exit(0);
    }
  }
}
```

OUTPUT:

```
 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 10

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 20

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node :
30

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 40

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 50

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 3

 HEAD -> 50 -> 40 -> 30 -> 20 -> 10 -> NULL

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 2
Node 50 deleted

 Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice :
```

//Write a C program to convert infix expression to its postfix form using Stack.

```
#include<stdio.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
struct Stack {
   int top;
   int maxSize;
   int* array;
};
struct Stack* create(int max)
{
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack -> maxSize = max;
   stack -> top = -1;
   stack -> array = (int*)malloc(stack -> maxSize * sizeof(int));
   return stack;
}
int isFull(struct Stack* stack)
{
   if(stack -> top == stack -> maxSize - 1)
   {
      printf("Will not be able to push maxSize reached\n");
   }
   return stack -> top == stack -> maxSize - 1;
}

int isEmpty(struct Stack* stack)
{
   return stack -> top == -1;
}
void push(struct Stack* stack, int item)
{
   if (isFull(stack))
      return;
   stack -> array[++stack -> top] = item;
}
int pop(struct Stack* stack)
{
   if (isEmpty(stack))
```

```
        return INT_MIN;
    return stack -> array[stack -> top--];
}
int peek(struct Stack* stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top];
}
int checkIfOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}
int precedence(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}
int covertInfixToPostfix(char* expression)
{
    int i, j;

    struct Stack* stack = create(strlen(expression));
    if(!stack)
        return -1 ;

    for (i = 0, j = -1; expression[i]; ++i)
    {
```

```
        if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];

         else if (expression[i] == '(')
            push(stack, expression[i]);

        else if (expression[i] == ')')
        {
            while (!isEmpty(stack) && peek(stack) != '(')
                expression[++j] = pop(stack);
            if (!isEmpty(stack) && peek(stack) != '(')
                return -1;
            else
                pop(stack);
        }
        else
        {
            while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))
                expression[++j] = pop(stack);
            push(stack, expression[i]);
        }

    }

     while (!isEmpty(stack))
        expression[++j] = pop(stack);

    expression[++j] = '\0';
    printf( "%s", expression);
}

int main()
{
char expression[] = "((x+(y*z))-w)";
    covertInfixToPostfix(expression);
    return 0;
}
```

OUTPUT:

```
xyz*+w-

...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and DISPLAY.

```
#include <stdio.h>
#include <conio.h>
struct node
{
   int label;
   struct node *next;
};
main()
{
   int ch=0;
   int k;
   struct node *h, *temp, *head;
   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;
   while(1)
   {
      printf("\n Queue using Linked List \n");
      printf("1->Enqueue ");
      printf("2->Dequeue");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            /* Reorganize the links */
            h = head;
            while (h->next != NULL)
               h = h->next;
            h->next = temp;
            temp->next = NULL;
```

```
        break;
        case 2:
          /* Delink the first node */
          h = head->next;
          head->next = h->next;
          printf("Node deleted \n");
          free(h);
        break;
        case 3:
          printf("\n\nHEAD -> ");
          h=head;
          while (h->next!=NULL)
          {
            h = h->next;
            printf("%d -> ",h->label);
          }
          printf("NULL \n");
        break;
        case 4:
          exit(0);
      }
    }
}
```

OUTPUT:

```
 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 1
Enter label for new node : 10

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 1
Enter label for new node : 20

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 1
Enter label for new node : 30

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 3


HEAD -> 10 -> 20 -> 30 -> NULL

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 2
Node deleted

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 3


HEAD -> 20 -> 30 -> NULL

 Queue using Linked List
1->Enqueue 2->Dequeue3->View 4->Exit
Enter your choice : 4


...Program finished with exit code 0
```

//Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder).

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

struct node {
   int data;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data) {
   struct node tempNode = (struct node) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;

      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
```

```
         }
      }  //go to right of the tree
      else {
         current = current->rightChild;

         //insert to the right
         if(current == NULL) {
            parent->rightChild = tempNode;
            return;
         }
      }
   }
  }
}

struct node* search(int data) {
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);

      //go to left tree
      if(current->data > data) {
         current = current->leftChild;
      }
      //else go to right tree
      else {
         current = current->rightChild;
      }

      //not found
      if(current == NULL) {
         return NULL;
      }
   }

   return current;
}
```

```
void pre_order_traversal(struct node* root) {
  if(root != NULL) {
    printf("%d ",root->data);
    pre_order_traversal(root->leftChild);
    pre_order_traversal(root->rightChild);
  }
}

void inorder_traversal(struct node* root) {
  if(root != NULL) {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
}

void post_order_traversal(struct node* root) {
  if(root != NULL) {
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

int main() {
  int i;
  int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

  for(i = 0; i < 7; i++)
    insert(array[i]);

  i = 31;
  struct node * temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
```

```
  }

  i = 15;
  temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```

OUTPUT:

```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27

...Program finished with exit code 0
Press ENTER to exit console.
```

// To implement a hash table using Linear Probing method.

```
#include <stdio.h>
 #include<stdlib.h>
 #define TABLE_SIZE 10
 int h[TABLE_SIZE]={NULL};
 void insert()
 {
  int key,index,i,flag=0,hkey;
  printf("\nenter a value to insert into hash table\n");
  scanf("%d",&key);
  hkey=key%TABLE_SIZE;
  for(i=0;i<TABLE_SIZE;i++)
   {
    index=(hkey+i)%TABLE_SIZE;
    if(h[index] == NULL)
    {
     h[index]=key;
      break;
    }
   }
   if(i == TABLE_SIZE)
    printf("\nelement cannot be inserted\n");
 }
 void search()
 {
  int key,index,i,flag=0,hkey;
  printf("\nenter search element\n");
  scanf("%d",&key);
  hkey=key%TABLE_SIZE;
  for(i=0;i<TABLE_SIZE; i++)
  {
   index=(hkey+i)%TABLE_SIZE;
   if(h[index]==key)
   {
    printf("value is found at index %d",index);
    break;
   }
  }
  if(i == TABLE_SIZE)
   printf("\n value is not found\n");
```

```
}
void display()
{
 int i;
 printf("\nelements in the hash table are \n");
 for(i=0;i< TABLE_SIZE; i++)
 printf("\nat index %d \t value = %d",i,h[i]);
}
 int main()
{
  int opt,i;
  while(1)
   {
    printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
    scanf("%d",&opt);
    switch(opt)
     {
      case 1:
        insert();
        break;
      case 2:
        display();
        break;
      case 3:
        search();
        break;
      case 4:exit(0);
     }
   }
  return 0;
}
```

OUTPUT:

```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
89

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
34

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
44

Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are

at index 0       value = 0
at index 1       value = 0
at index 2       value = 0
at index 3       value = 0
at index 4       value = 34
at index 5       value = 44
at index 6       value = 0
at index 7       value = 0
at index 8       value = 0
at index 9       value = 89
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
44
value is found at index 5
Press 1. Insert  2. Display     3. Search       4.Exit
```

// To sort an array of N numbers using Insertion sort.

```
#include<stdio.h>
void main()
{
    int i, j, k, n, temp, a[20], p=0;
    printf("Enter total elements: ");
    scanf("%d",&n);
    printf("Enter array elements: ");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    for(i=1; i<n; i++)
    {
        temp = a[i];
        j = i - 1;
        while((temp<a[j]) && (j>=0))
        {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = temp;
        p++;
        printf("\n After Pass %d: ", p);
        for(k=0; k<n; k++)
            printf(" %d", a[k]);
    }
    printf("\n Sorted List : ");
    for(i=0; i<n; i++)
        printf(" %d", a[i]);
}
```

OUTPUT:

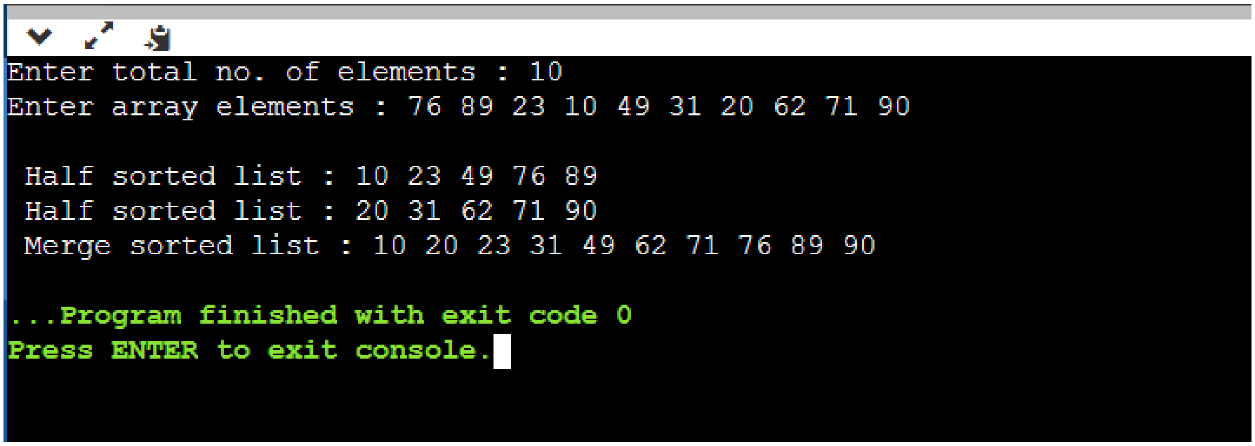
```
Enter total elements: 10
Enter array elements: 23 67 10 87 41 60 53 39 74 8

 After Pass 1:  23 67 10 87 41 60 53 39 74 8
 After Pass 2:  10 23 67 87 41 60 53 39 74 8
 After Pass 3:  10 23 67 87 41 60 53 39 74 8
 After Pass 4:  10 23 41 67 87 60 53 39 74 8
 After Pass 5:  10 23 41 60 67 87 53 39 74 8
 After Pass 6:  10 23 41 53 60 67 87 39 74 8
 After Pass 7:  10 23 39 41 53 60 67 87 74 8
 After Pass 8:  10 23 39 41 53 60 67 74 87 8
 After Pass 9:  8 10 23 39 41 53 60 67 74 87
 Sorted List :  8 10 23 39 41 53 60 67 74 87

...Program finished with exit code 0
Press ENTER to exit console.
```

// **To sort an array of N numbers using Merge sort.**

```
#include <stdio.h>
#include <conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;
int main()
{
   int i, arr[30];
   printf("Enter total no. of elements : ");
   scanf("%d", &size);
   printf("Enter array elements : ");
   for(i=0; i<size; i++)
   scanf("%d", &arr[i]);
   part(arr, 0, size-1);
   printf("\n Merge sorted list : ");
   for(i=0; i<size; i++)
   printf("%d ",arr[i]);
   return 0;
}
void part(int arr[], int min, int max)
{
   int mid,i;
   if(min < max)
   {
      mid = (min + max) / 2;
      part(arr, min, mid);
      part(arr, mid+1, max);
      merge(arr, min, mid, max);
   }
   if (max-min == (size/2)-1)
   {
      printf("\n Half sorted list : ");
      for(i=min; i<=max; i++)
         printf("%d ", arr[i]);
   }
}
void merge(int arr[],int min,int mid,int max)
{
   int tmp[30];
```

```
    int i, j, k, m;
    j = min;
    m = mid + 1;
    for(i=min; j<=mid && m<=max; i++)
    {
        if(arr[j] <= arr[m])
        {
            tmp[i] = arr[j];
            j++;
        }
        else
        {
            tmp[i] = arr[m];
            m++;
        }
    }
    if(j > mid)
    {
        for(k=m; k<=max; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    else
    {
        for(k=j; k<=mid; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    for(k=min; k<=max; k++)
        arr[k] = tmp[k];
}
```

OUTPUT:



```
Enter total no. of elements : 10
Enter array elements : 76 89 23 10 49 31 20 62 71 90

 Half sorted list : 10 23 49 76 89
 Half sorted list : 20 31 62 71 90
 Merge sorted list : 10 20 23 31 49 62 71 76 89 90

...Program finished with exit code 0
Press ENTER to exit console.
```

//To sort an array of N numbers using Quick sort.

```
#include<stdio.h>
#include<conio.h>
void qsort(int arr[20], int fst, int last);
int main()
{
   int arr[30];
   int i, size;
   printf("Enter total no. of the elements : ");
   scanf("%d", &size);
   printf("Enter total %d elements : \n", size);
   for(i=0; i<size; i++)
   scanf("%d", &arr[i]);
   qsort(arr,0,size-1);
   printf("\n Quick sorted elements \n");
   for(i=0; i<size; i++)
   printf("%d\t", arr[i]);
   return 0;
}
void qsort(int arr[20], int fst, int last)
{
   int i, j, pivot, tmp;
   if(fst < last)
   {
      pivot = fst;
      i = fst;
      j = last;
      while(i < j)
      {
         while(arr[i] <=arr[pivot] && i<last)
            i++;
         while(arr[j] > arr[pivot])
            j--;
         if(i <j )
         {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
         }
      }
```

```
        tmp = arr[pivot];
        arr[pivot] = arr[j];
        arr[j] = tmp;
        qsort(arr, fst, j-1);
        qsort(arr, j+1, last);
    }
}
```

OUTPUT:

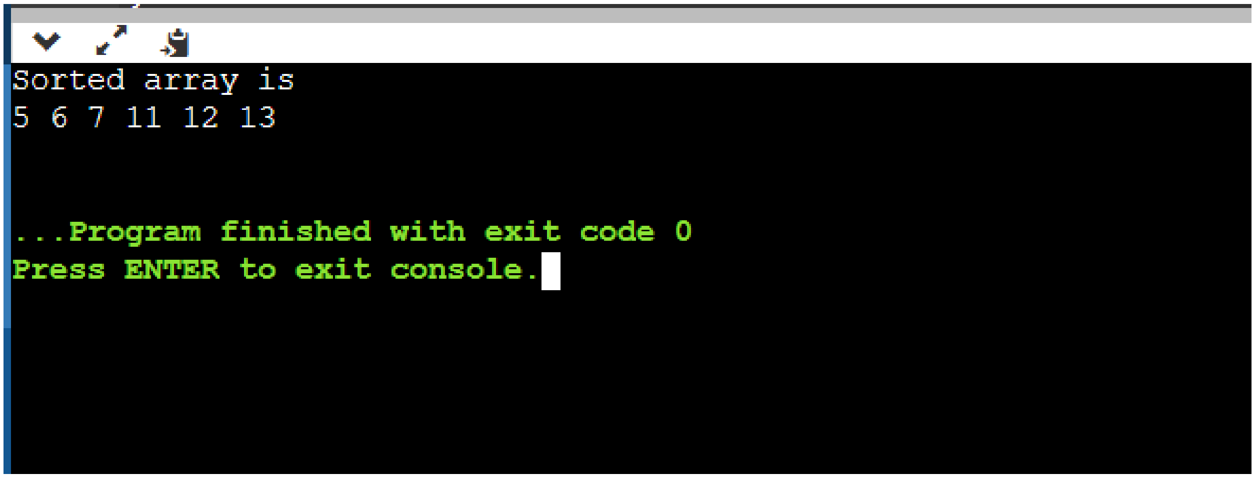
```
Enter total no. of the elements : 10
Enter total 10 elements :
67 98 23 54 29 41 38 17 64 75

 Quick sorted elements
17      23      29      38      41      54      64      67      75      98

...Program finished with exit code 0
Press ENTER to exit console.
```

// Write a C program to implement Heap sort.

```
#include <stdio.h>
void swap(int* a, int* b)
{

    int temp = *a;

    *a = *b;

    *b = temp;
}

void heapify(int arr[], int N, int i)
{
    // Find largest among root, left child and right child
    int largest = i;
    // left = 2*i + 1
    int left = 2 * i + 1;
    // right = 2*i + 2
    int right = 2 * i + 2;
    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])
        largest = left;
    if (right < N && arr[right] > arr[largest])

        largest = right;
    if (largest != i) {

        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}

// Main function to do heap sort
void heapSort(int arr[], int N)
{

    // Build max heap
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);
```

```
    // Heap sort
    for (int i = N - 1; i >= 0; i--)
    {
        swap(&arr[0], &arr[i]);
         // Heapify root element to get highest element at
        heapify(arr, i, 0);
    }
}

// A utility function to print array of size n
void printArray(int arr[], int N)
{
    for (int i = 0; i < N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver's code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int N = sizeof(arr) / sizeof(arr[0]);
     // Function call
    heapSort(arr, N);
    printf("Sorted array is\n");
    printArray(arr, N);
}
```

OUTPUT:


Sorted array is
5 6 7 11 12 13
...Program finished with exit code 0
Press ENTER to exit console.

// Write a C program to perform AVL Tree operations such as Insertion, Deletion and search.

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

// structure of the tree node
struct node
{
    int data;
    struct node* left;
    struct node* right;
    int ht;
};

// global initialization of root node
struct node* root = NULL;

// function prototyping
struct node* create(int);
struct node* insert(struct node*, int);
struct node* delete(struct node*, int);
struct node* search(struct node*, int);
struct node* rotate_left(struct node*);
struct node* rotate_right(struct node*);
int balance_factor(struct node*);
int height(struct node*);

int main()
{
    int user_choice, data;
    char user_continue = 'y';
    struct node* result = NULL;

    while (user_continue == 'y' || user_continue == 'Y')
    {
        printf("\n\n------- AVL TREE --------\n");
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Search");
        printf("\n4. EXIT");
```

```
printf("\n\nEnter Your Choice: ");
scanf("%d", &user_choice);

switch(user_choice)
{
   case 1:
      printf("\nEnter data: ");
      scanf("%d", &data);
      root = insert(root, data);
      break;

   case 2:
      printf("\nEnter data: ");
      scanf("%d", &data);
      root = delete(root, data);
      break;

   case 3:
      printf("\nEnter data: ");
      scanf("%d", &data);
      result = search(root, data);
      if (result == NULL)
      {
         printf("\nNode not found!");
      }
      else
      {
         printf("\n Node found");
      }
      break;
   case 4:
      printf("\n\tProgram Terminated\n");
      return 1;

   default:
      printf("\n\tInvalid Choice\n");
}

printf("\n\nDo you want to continue? ");
scanf(" %c", &user_continue);
```

```
    }

    return 0;
}

// creates a new tree node
struct node* create(int data)
{
    struct node* new_node = (struct node*) malloc (sizeof(struct node));

    // if a memory error has occurred
    if (new_node == NULL)
    {
        printf("\nMemory can't be allocated\n");
        return NULL;
    }
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

// rotates to the left
struct node* rotate_left(struct node* root)
{
    struct node* right_child = root->right;
    root->right = right_child->left;
    right_child->left = root;

    // update the heights of the nodes
    root->ht = height(root);
    right_child->ht = height(right_child);

    // return the new node after rotation
    return right_child;
}

// rotates to the right
struct node* rotate_right(struct node* root)
{
```

```
    struct node* left_child = root->left;
    root->left = left_child->right;
    left_child->right = root;

    // update the heights of the nodes
    root->ht = height(root);
    left_child->ht = height(left_child);

    // return the new node after rotation
    return left_child;
}

// calculates the balance factor of a node
int balance_factor(struct node* root)
{
    int lh, rh;
    if (root == NULL)
        return 0;
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;
    return lh - rh;
}

// calculate the height of the node
int height(struct node* root)
{
    int lh, rh;
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL)
        lh = 0;
    else
```

```
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;

    if (lh > rh)
        return (lh);
    return (rh);
}

// inserts a new node in the AVL tree
struct node* insert(struct node* root, int data)
{
    if (root == NULL)
    {
        struct node* new_node = create(data);
        if (new_node == NULL)
        {
            return NULL;
        }
        root = new_node;
    }
    else if (data > root->data)
    {
        // insert the new node to the right
        root->right = insert(root->right, data);

        // tree is unbalanced, then rotate it
        if (balance_factor(root) == -2)
        {
            if (data > root->right->data)
            {
                root = rotate_left(root);
            }
            else
            {
                root->right = rotate_right(root->right);
                root = rotate_left(root);
            }
```

```
        }
    }
    else
    {
        // insert the new node to the left
        root->left = insert(root->left, data);

        // tree is unbalanced, then rotate it
        if (balance_factor(root) == 2)
        {
            if (data < root->left->data)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    // update the heights of the nodes
    root->ht = height(root);
    return root;
}

// deletes a node from the AVL tree
struct node * delete(struct node *root, int x)
{
    struct node * temp = NULL;

    if (root == NULL)
    {
        return NULL;
    }

    if (x > root->data)
    {
        root->right = delete(root->right, x);
        if (balance_factor(root) == 2)
```

```
    {
        if (balance_factor(root->left) >= 0)
        {
            root = rotate_right(root);
        }
        else
        {
            root->left = rotate_left(root->left);
            root = rotate_right(root);
        }
    }
}
else if (x < root->data)
{
    root->left = delete(root->left, x);
    if (balance_factor(root) == -2)
    {
        if (balance_factor(root->right) <= 0)
        {
            root = rotate_left(root);
        }
        else
        {
            root->right = rotate_right(root->right);
            root = rotate_left(root);
        }
    }
}
else
{
    if (root->right != NULL)
    {
        temp = root->right;
        while (temp->left != NULL)
            temp = temp->left;

        root->data = temp->data;
        root->right = delete(root->right, temp->data);
        if (balance_factor(root) == 2)
        {
```

```
            if (balance_factor(root->left) >= 0)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    else
    {
        return (root->left);
    }
}
root->ht = height(root);
return (root);
}

// search a node in the AVL tree
struct node* search(struct node* root, int key)
{
    if (root == NULL)
    {
        return NULL;
    }

    if(root->data == key)
    {
        return root;
    }

    if(key > root->data)
    {
        search(root->right, key);
    }
    else
    {
        search(root->left, key);  }}
```

OUTPUT:

```
------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 1

Enter data: 10


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 1

Enter data: 5


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 1

Enter data: 78
```

```
Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 1

Enter data: 67


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 2

Enter data: 10


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 3
```

```
Enter data: 100

Node not found!

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 3

Enter data: 67

 Node found

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. EXIT

Enter Your Choice: 4
```

//Write a C program to perform the graph traversal using Breadth First Search.

```
#include <stdio.h>
#include <stdlib.h>
 struct node
{
   int vertex;
   struct node *next;
};

struct node *createNode(int);

struct Graph
{
   int numVertices;
   struct node **adjLists;
   int *visited;
};

struct Graph *createGraph(int vertices)
{
   struct Graph *graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;

   graph->adjLists = malloc(vertices * sizeof(struct node *));
   graph->visited = malloc(vertices * sizeof(int));

   int i;
   for (i = 0; i < vertices; i++)
   {
      graph->adjLists[i] = NULL;
      graph->visited[i] = 0;
   }

   return graph;
}

void addEdge(struct Graph *graph, int src, int dest)
{
   struct node *newNode = createNode(dest);
   newNode->next = graph->adjLists[src];
```

```
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

struct node *createNode(int v)
{
    struct node *newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

void printGraph(struct Graph *graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node *temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

void bfs(struct Graph *graph, int startVertex)
{
    struct node *queue = NULL;
    graph->visited[startVertex] = 1;
    enqueue(&queue, startVertex);

    while (!isEmpty(queue))
    {
        printQueue(queue);
```

```
        int currentVertex = dequeue(&queue);
        printf("Visited %d ", currentVertex);

        struct node *temp = graph->adjLists[currentVertex];

        while (temp)
        {
            int adjVertex = temp->vertex;

            if (graph->visited[adjVertex] == 0)
            {
                graph->visited[adjVertex] = 1;
                enqueue(&queue, adjVertex);
            }
            temp = temp->next;
        }
    }
}

int isEmpty(struct node *queue)
{
    return queue == NULL;
}

void enqueue(struct node **queue, int value)
{
    struct node *newNode = createNode(value);
    if (isEmpty(*queue))
    {
        *queue = newNode;
    }
    else
    {
        struct node *temp = *queue;
        while (temp->next)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
```

```
}

int dequeue(struct node **queue)
{
   int nodeData = (*queue)->vertex;
   struct node *temp = *queue;
   *queue = (*queue)->next;
   free(temp);
   return nodeData;
}

void printQueue(struct node *queue)
{
   while (queue)
   {
      printf("%d ", queue->vertex);
      queue = queue->next;
   }
   printf("\n");
}

int main(void)
{
   struct Graph *graph = createGraph(6);
   printf("\nWhat do you want to do?\n");
   printf("1. Add edge\n");
   printf("2. Print graph\n");
   printf("3. BFS\n");
   printf("4. Exit\n");
   int choice;
   scanf("%d", &choice);
   while (choice != 4)
   {
      if (choice == 1)
      {
         int src, dest;
         printf("Enter source and destination: ");
         scanf("%d %d", &src, &dest);
         addEdge(graph, src, dest);
      }
```

```
        else if (choice == 2)
        {
            printGraph(graph);
        }
        else if (choice == 3)
        {
            int startVertex;
            printf("Enter starting vertex: ");
            scanf("%d", &startVertex);
            bfs(graph, startVertex);
        }
        else
        {
            printf("Invalid choice\n");
        }
        printf("What do you want to do?\n");
        printf("1. Add edge\n");
        printf("2. Print graph\n");
        printf("3. BFS\n");
        printf("4. Exit\n");
        scanf("%d", &choice);
    }
    return 0;
}
```

OUTPUT:

```
What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
1
Enter source and destination: 0 1
What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
1
Enter source and destination: 0 2
What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
1
Enter source and destination: 1 2
What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
1
Enter source and destination: 2 3
What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
2

 Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
```

```
 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->

 Adjacency list of vertex 4


 Adjacency list of vertex 5

What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
3
Enter starting vertex: 0
0
Visited 0 2 1
Visited 2 1 3
Visited 1 3
Visited 3 What do you want to do?
1. Add edge
2. Print graph
3. BFS
4. Exit
4


...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to perform the graph traversal using Depth First Search.

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

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

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
};

// DFS algo
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

// Create a node
```

```
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Create graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

// Print the graph
void printGraph(struct Graph* graph) {
  int v;
```

```
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

  return 0;
}
```

OUTPUT:

```
Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0


...Program finished with exit code 0
Press ENTER to exit console.
```

// To implement the Shortest Path Algorithms using Dijkstra's Algorithm

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
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];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
```

```
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
```

```
}while(j!=startnode);
}
}
```

OUTPUT:

```
Enter no. of vertices: 5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0

...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to implement Minimum Spanning Tree using Prim's Algorithm.

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

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i,
               graph[i][parent[i]]);
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];
```

```
// Key values used to pick minimum weight edge in cut
int key[V];
// To represent set of vertices included in MST
bool mstSet[V];

// Initialize all keys as INFINITE
for (int i = 0; i < V; i++)
    key[i] = INT_MAX, mstSet[i] = false;

// Always include first 1st vertex in MST.
// Make key 0 so that this vertex is picked as first
// vertex.
key[0] = 0;

// First node is always root of MST
parent[0] = -1;

// The MST will have V vertices
for (int count = 0; count < V - 1; count++) {

    // Pick the minimum key vertex from the
    // set of vertices not yet included in MST
    int u = minKey(key, mstSet);

    // Add the picked vertex to the MST Set
    mstSet[u] = true;

    // Update key value and parent index of
    // the adjacent vertices of the picked vertex.
    // Consider only those vertices which are not
    // yet included in MST
    for (int v = 0; v < V; v++)

        // graph[u][v] is non zero only for adjacent
        // vertices of m mstSet[v] is false for vertices
        // not yet included in MST Update the key only
        // if graph[u][v] is smaller than key[v]
        if (graph[u][v] && mstSet[v] == false
            && graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
```

```
    }

    // print the constructed MST
    printMST(parent, graph);
}

// Driver's code
int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };

    // Print the solution
    primMST(graph);

    return 0;
}
```

OUTPUT:

```
Edge     Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5


...Program finished with exit code 0
Press ENTER to exit console.
```

//Write a C program to implement Minimum Spanning Tree using Kruskal Algorithm.

```
#include <stdio.h>
#include <stdlib.h>
 // Comparator function to use in sorting
int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
}
 // Initialization of parent[] and rank[] arrays
void makeSet(int parent[], int rank[], int n)
{
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}
// Function to find the parent of a node
int findParent(int parent[], int component)
{
    if (parent[component] == component)
        return component;

    return parent[component]
           = findParent(parent, parent[component]);
}
 // Function to unite two sets
void unionSet(int u, int v, int parent[], int rank[], int n)
{
```

```
    // Finding the parents
    u = findParent(parent, u);
    v = findParent(parent, v);

    if (rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if (rank[u] > rank[v]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;

        // Since the rank increases if
        // the ranks of two sets are same
        rank[u]++;
    }
}

// Function to find the MST
void kruskalAlgo(int n, int edge[n][3])
{
    // First we sort the edge array in ascending order
    // so that we can access minimum distances/cost
    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];
    int rank[n];

    // Function to initialize parent[] and rank[]
    makeSet(parent, rank, n);
```

```
    // To store the minimun cost
    int minCost = 0;

    printf(
        "Following are the edges in the constructed MST\n");
    for (int i = 0; i < n; i++) {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];

        // If the parents are different that
        // means they are in different sets so
        // union them
        if (v1 != v2) {
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n", edge[i][0],
                   edge[i][1], wt);
        }
    }

    printf("Minimum Cost Spanning Tree: %d\n", minCost);
}

// Driver code
int main()
{
    int edge[5][3] = { { 0, 1, 10 },
                       { 0, 2, 6 },
                       { 0, 3, 5 },
```

```
                    { 1, 3, 15 },
                    { 2, 3, 4 } };

    kruskalAlgo(5, edge);

    return 0;
}
```

OUTPUT:

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19


...Program finished with exit code 0
Press ENTER to exit console.
```