# Bachelor of Computer Applications (BCA)

# C Programming Lab

## (OBCACO106P24)

## SLM (SEMESTER 1)

## Jaipur National University
## Directorate of Distance Education

---

**Established by Government of Rajasthan**
**Approved by UGC under Sec 2(f) of UGC ACT 1956**
**&**
**NAAC A+ Accredited**

# TABLE OF CONTENTS

## EXPERT COMMITTEE

Prof. Sunil Gupta
(Computer and Systems Sciences, JNU Jaipur)

Dr. Deepak Shekhawat
(Computer and Systems Sciences, JNU Jaipur)

Dr. Shalini Rajawat
(Computer and Systems Sciences, JNU Jaipur)

## COURSE COORDINATOR

Pawan Jakhar
(Computer and Systems Sciences, JNU Jaipur)

## UNIT PREPARATION

| **Unit Writer(s)** | **Assisting & Proofreading** | **Unit Editor** |
|---|---|---|
| Pawan Jakhar<br>(Computer and Systems Sciences, JNU Jaipur) | Mr. Satender Singh<br>(Computer and Systems Sciences, JNU Jaipur) | Mr. Ramlal Yadav<br>(Computer and Systems Sciences, JNU Jaipur) |

**Secretarial Assistance**

Mr. Mukesh Sharma

# COURSE INTRODUCTION

***"Clean code always looks like it was written by someone who cares."***
***- Robert C. Martin***

Welcome to the C Programming Lab course! This hands-on laboratory experience is designed to deepen your understanding of C programming by moving beyond theoretical concepts to practical application. In this course, you'll explore the foundational elements of C programming through a series of interactive exercises and projects. Our goal is to build your proficiency in coding, debugging, and problem-solving using C, one of the most enduring and influential programming languages.

Throughout the lab sessions, you'll engage with real-world programming challenges that will enhance your skills in areas such as data types, control structures, functions, and pointers. You'll also gain practical experience with memory management and file handling, which are crucial for developing efficient and robust software. Each lab will encourage you to think critically and creatively, applying your knowledge to solve complex problems and optimize your code.

By the end of the course, you will have developed a solid foundation in C programming that will serve as a valuable asset in your future computing endeavors. Whether you're aiming for a career in software development, systems programming, or any field where coding is essential, the skills you acquire here will be instrumental in your success. Get ready to immerse yourself in the world of C programming, where you'll transform theoretical concepts into tangible, real-world applications.

**Course Outcomes:**
**At the completion of the course, a student will be able to:**

1. Identify situations where computational methods and computers would be useful.
2. Summarize the programming tasks using techniques learned and writepseudo-code.
3. Choose the right data representation formats based on the requirements of the problem.
4. Use comparisons and limitations of the various programming constructs and choose the right one for the task in hand.
5. Implement file Operations in C programming for a given application.

---

**Acknowledgements:**

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

---

# C Programming Lab

## 1. Write a program to find the sum of two integers.

```
#include <stdio.h>
int main() {
   int num1, num2, sum;
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);

   sum = num1 + num2;
printf("Sum = %d\n", sum);
return ;
}
```

## 2. Write a C code to find the difference between two integers.

```
#include <stdio.h>
int main() {
   int num1, num2, difference;
printf("Enter first integer: ");
scanf("%d", &num1);
printf("Enter second integer: ");
scanf("%d", &num2);
   difference = num1 - num2;
printf("The difference is: %d\n", difference);

return ;
}
```

## 3. Write a program to multiply two floating-point numbers.

```
#include <stdio.h>

int main() {
   float num1, num2, product;

printf("Enter first float: ");
scanf("%f", &num1);

printf("Enter second float: ");
scanf("%f", &num2);
product = num1 * num2;
printf("The product is: %.2f\n", product);

return ;
}
```

## 4. Write a program to divide two numbers and output the quotient and remainder.

```
#include <stdio.h>

int main() {
   int num1, num2, quotient, remainder;
printf("Enter dividend: ");
```

```
scanf("%d", &num1);

printf("Enter divisor: ");
scanf("%d", &num2);

   quotient = num1 / num2;
   remainder = num1 % num2;

printf("The quotient is: %d\n", quotient);
printf("The remainder is: %d\n", remainder);

return ;
```

5. **Write a program code in C to count the factorial of a provided number using recursion.**

```
#include <stdio.h>
long int factorial(int n) {
   if (n >= 1)
      return n * factorial(n - 1);
   else
      return 1;
}
int main() {
   int num;
printf("Enter a number: ");
scanf("%d", &num);
printf("Factorial of %d is %ld\n", num, factorial(num));
return ;
}
```

**6. Write a 'C' program code to check whether a provided number is even or odd.**

```
#include <stdio.h>

int main() {
   int num;
printf("Enter an integer: ");
scanf("%d", &num);

   if (num % 2 == 0)
printf("%d is even.", num);
   else
printf("%d is odd.", num);
return ;
}
```

**7. Write a program code in 'C' language to check whether a provided number is prime or not.**

```
#include <stdio.h>
int main() {
   int num, i, isPrime = 0;
printf("Enter a positive integer: ");
scanf("%d", &num);

for(i = 2; i<= num/2; ++i) {
if(num%i == 0) {
isPrime = 1;
break;

}
 }
   if (isPrime == 0)
printf("%d is a prime number.", num);
else
printf("%d is not a prime number.", num);
return ;
}
```

**8. Write a program to find the largest number among three given numbers.**

```
#include <stdio.h>
int main() {
   int num1, num2, num3;
printf("Enter three numbers: ");
scanf("%d %d %d", &num1, &num2, &num3);
if(num1 >= num2 && num1 >= num3)
printf("%d is the largest number.", num1);
   else if(num2 >= num1 && num2 >= num3)
printf("%d is the largest number.", num2);
   else
printf("%d is the largest number.", num3);
return ;
}
```

**9. Write a program code in 'C' language to swap the values of two given variables by using pointers.**

```
#include <stdio.h>

void swap(int *num1, int *num2) {
   int temp = *num1;
   *num1 = *num2;
   *num2 = temp;
}

int main() {
   int num1, num2;
printf("Enter two numbers: ");
scanf("%d %d", &num1, &num2);
printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
swap(&num1, &num2);
printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
return ;
}
```

**10. Write a program code in 'C' language to find the length of a string without apply any library function.**

```
#include <stdio.h>
int main() {
   char str[100];
   int length = 0;

printf("Enter a string: ");
   gets(str);
while(str[length] != '\0') {
      ++length;
   }
printf("Length of string: %d", length);
return ;
}
```

**11. Write a program code in 'C' language to copy a string from source to destination without apply any library function.**

```
#include <stdio.h>
void copyString(char *source, char *destination) {
while(*source != '\0') {
*destination = *source;                                    }
*destination = '\0';
}
int main() {
char source[100], destination[100];
printf("Enter a string: ");
gets(source);
copyString(source, destination);
printf("Copied String is: %s\n", destination);
return ;
```

**12. Write a program code in 'C' language to compare two strings without apply any library function.**

```
#include<stdio.h>

int compareStrings(char* str1, char* str2) {
   int i = 0;
   while(str1[i] == str2[i]) {
      if(str1[i] == '\0' || str2[i] == '\0')
break;
i++;
   }
   if(str1[i] == '\0' && str2[i] == '\0')
   return 0;
   else
    return -1;
}
int main() {
   char str1[100], str2[100];
printf("Enter first string: ");
   gets(str1);
printf("Enter second string: ");
   gets(str2);
   int result = compareStrings(str1, str2);
if(result == 0)
printf("Strings are equal.\n");
   else
printf("Strings are not equal.\n");
return ;
}
```

**13. Write a program code in 'C' language to concatenate two strings without applying any library function.**

```
#include <stdio.h>
void concatenateStrings(char* str1, char* str2) {
   int ij= 0, i = 0;

   while(str1[i] != '\0')
i++;

   while(str2[j] != '\0') {
      str1[i] = str2[j];
i++;
j++;
   }

   str1[i] = '\0';
}

int main() {
   char str1[200], str2[100];

printf("Enter first string: ");
   gets(str1);

printf("Enter second string: ");
   gets(str2);

concatenateStrings(str1, str2);

printf("After concatenation: %s\n", str1);

return ;
}
```

**14. Write a program program code in 'C' language to reverse a string without applying any library function.**

```
#include <stdio.h>
void reverseString(char* str) {
    int len = 0, i;
    char temp;
    while (str[len] != '\0')
len++;

    for (i = 0; i<len / 2; i++) {
        temp = str[i];
        str[i] = str[len - i - 1];
str[len - i - 1] = temp;
    }
}
int main() {
    char str[100];

printf("Enter a string: ");
    gets(str);

reverseString(str);

printf("Reversed String is: %s\n", str);
return ;
}
```

## 15. Write a program to sort an array in ascending order.

```
#include <stdio.h>
void sortArrayInAscendingOrder(int array[], int n) {
   int i, j, temp;
for(i = 0; i< n - 1; i++) {
for(j = 0; j < n - i - 1; j++) {
          if(array[j] >array[j + 1]) {
             temp = array[j];
             array[j] = array[j + 1];
array[j + 1] = temp;
          }
       }
   }
}

int main() {
   int array[100], n, i;
printf("Enter number of elements in array: ");
scanf("%d", &n);
printf("Enter %d integers: ", n);
for(i = 0; i< n; i++)
scanf("%d", &array[i]);
sortArrayInAscendingOrder(array, n);
printf("Sorted list in ascending order: ");

for(i = 0; i< n; i++)
printf("%d ", array[i]);
return ;
}
```

## 16. Write a program to discover the maximum and minimum element in an array.

```
#include <stdio.h>
void findMaxMin(int array[], int n) {
   int max = array[0], min = array[0], i;
for(i = 1; i< n; i++) {
      if(array[i] > max)
         max = array[i];
      if(array[i] < min)
         min = array[i];
   }
printf("Maximum element = %d\n", max);
printf("Minimum element = %d\n", min);
}
int main() {
   int array[100], n, i;
printf("Enter number of elements in array: ");
scanf("%d", &n);
printf("Enter %d integers: ", n);
for(i = 0; i< n; i++)
scanf("%d", &array[i]);

findMaxMin(array, n);

return ;
}
```

## 17. Write a program to implement linear search in an Array.

#include <stdio.h>

int linear_search(int array[], int size, int target){

```
#include <stdio.h>
int linear_search(int array[], int size, int target){
for(int i = 0; i< size; i++){
    if(array[i] == target){
      return i;
    }
  }
  return -1;
}
int main(){
  int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
  int size = sizeof(array)/sizeof(array[0]);
  int target = 5;
  int result = linear_search(array, size, target);
(result != -1)? printf("Element found at index: %d\n", result)
            : printf("Element not found\n");
```

return ;

}

## 18. Write a program to implement binary search in an array.

```
int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
  int size = sizeof(array) / sizeof(array[0]);
  int target = 5;


  int result = binary_search(array, 0, size - 1, target);


  (result != -1)? printf("Element found at index: %d\n", result)
              : printf("Element not found\n");


return ;
}
```

## 19. Write a program to implement selection sort in an array.

```
#include <stdio.h>
void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void selection_sort(int array[], int size){
   int i, j, min_idx;
   for (i = 0; i< size-1; i++){
min_idx = i;
      for (j = i+1; j < size; j++){
        if (array[j] < array[min_idx]){
min_idx = j;
        }

int main(){
   int array[] = {64, 25, 12, 22, 11};
   int size = sizeof(array)/sizeof(array[0]);
selection_sort(array, size);
printf("Sorted array: \n");
   for (int i=0; i< size; i++){
printf("%d ", array[i]);
   }
return ;
}
```

## 20. Write a program to implement bubble sort in an array.

```
#include <stdio.h>
void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void bubble_sort(int array[], int size){
for(int i = 0; i< size-1; i++){
      for (int j = 0; j < size-i-1; j++){
         if (array[j] > array[j+1]){
            swap(&array[j], &array[j+1]);
         }
      }
   }
}
int main(){
   int array[] = {64, 34, 25, 12, 22, 11, 90};
   int size = sizeof(array)/sizeof(array[0]);
bubble_sort(array, size);
printf("Sorted array: \n");
for(int i = 0; i< size; i++){
printf("%d ", array[i]);
   }
return ;
}
   printf("Sorted array: \n");
     for (int i = 0; i< size; i++){
   printf("%d ", array[i]);
     }
   return ;
   }
```

## 21. Write a program to implement quick sort in an array.

```
#include <stdio.h>
void swap(int* a, int* b) {
int t = *a;
*a = *b;
*b = t;
}
int partition (int arr[], int low, int high) {
int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high- 1; j++) {
if (arr[j] < pivot) {
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}
void quickSort(int arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main() {
int arr[] = {10, 7, 8, 9, 1, 5};
int n = sizeof(arr)/sizeof(arr[0]);
quickSort(arr, 0, n-1);
printf("Sorted array: \n");
for (int i=0; i< n; i++)
printf("%d ", arr[i]);
return ;
}
```

## 22. Write a program to implement merge sort in an array.

```
#include<stdio.h>

void merge(int arr[], int l, int m, int r) {

   int i, j, k;

   int n1 = m - l + 1;

   int n2 = r - m;

   int L[n1], R[n2];

   for (i = 0; i< n1; i++)

      L[i] = arr[l + i];

   for (j = 0; j < n2; j++)

      R[j] = arr[m + 1 + j];

i = 0;

   j = 0;

   k = l;

   while (i< n1 && j < n2) {

      if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

      } else {

arr[k] = R[j];

j++;

      }

      k++;

   }

   while (i< n1) {

arr[k] = L[i];
```

```
i++;
        k++;
    }
    while (j < n2) {
arr[k] = R[j];
j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
mergeSort(arr, l, m);
mergeSort(arr, m + 1, r);
merge(arr, l, m, r);
    }
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
mergeSort(arr, 0, arr_size - 1);
printf("Sorted array is \n");
    for (int i = 0; i<arr_size; i++)
printf("%d ", arr[i]);
return ;
}
```

## 23. Write a program to implement heap sort in an array.

```
#include <stdio.h>
void swap(int *a, int *b) {
   int t = *a;
   *a = *b;
   *b = t;
}
void heapify(int arr[], int n, int i) {
   int largest = i;
   int left = 2*i + 1;
   int right = 2*i + 2;
   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);
   }
}
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[] = {12, 11, 13, 5, 6, 7};
   int n = sizeof(arr)/sizeof(arr[0]);
heapSort(arr, n);
printf("Sorted array is \n");
   for (int i=0; i<n; ++i)
printf("%d ", arr[i]);
printf("\n");
return ;
}
```

## 24. Write a program to calculate the average of numbers in an array.

```
#include <stdio.h>

int main() {
   int arr[] = {10, 20, 30, 40, 50};
   int n = sizeof(arr) / sizeof(arr[0]);
   int sum = 0;

for(int i = 0; i< n; i++) {
      sum += arr[i];
   }

   float average = (float)sum / n;
printf("The average of the array is %.2f\n", average);

return ;
}
```

## 25. Write a program to implement stack operations using arrays.

```
#include<stdio.h>
#define MAX_SIZE 101
int A[MAX_SIZE];
int top = -1;

void Push(int x) {
if(top == MAX_SIZE -1) {
printf("Error: stack overflow\n");
return;
 }
 A[++top] = x;
}

void Pop() {
if(top == -1) {
```

```
printf("Error: No element to pop\n");
return;
  }
  top--;
}
int Top() {
  return A[top];
}
int IsEmpty() {
if(top == -1) {
    return 1;
  }
  return 0;
}
int main() {
Push(2); Push(5); Push(10);
printf("Top element is %d\n",Top());
Pop();
printf("Top element is %d\n",Top());
}
```

## 26. Write a program to implement queue operations using arrays.

```
#include <stdio.h>
 #define SIZE 5

 int items[SIZE], front = -1, rear = -1;

 void enQueue(int value){
 if(rear == SIZE-1)
 printf("\nQueue is Full!!");
   else {
 if(front == -1)
       front = 0;
     rear++;
     items[rear] = value;
 printf("\nInserted -> %d", value);
   }
 }
```

```
void deQueue(){
if(front == -1)
printf("\nQueue is Empty!!");
else{
printf("\nDeleted : %d", items[front]);
    front++;
if(front > rear)
       front = rear = -1;
  }
}

int main(){
  //deQueue is not possible on empty queue
deQueue();
enQueue(1);
enQueue(2);
enQueue(3);
enQueue(4);
enQueue(5);

enQueue(6);
deQueue();
enQueue(7);
return ;
}
```

## 27. Write a program to implement a circular queue using an array.

```
#include <stdio.h>

#define SIZE 5


int items[SIZE], front = -1, rear =-1;


int isFull() {
   if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
   return 0;
}


int isEmpty() {
   if (front == -1) return 1;
   return 0;
}


void enQueue(int element) {
   if (isFull()) printf("\n Queue is full!! \n");
int deQueue() {
   int element;
   if (isEmpty()) {
printf("\n Queue is empty !! \n");
return(-1);
   } else {
      element = items[front];
      if (front == rear){
         front = -1;
         rear = -1;
      } else {
         front = (front + 1) % SIZE;
      }
```

```
printf("\n Deleted element -> %d \n", element);
        return(element);
    }
}

int main() {
enQueue(1);
enQueue(2);
enQueue(3);
enQueue(4);
enQueue(5);
deQueue();
deQueue();
    return 0;
}
```

## 28. Write a program code in ‘C’ to applying a priority queue.

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

// Define Node structure
typedef struct node {
    int data;
    int priority;
    struct node* next;
} Node;
```

```
// Function to create a new node
Node* newNode(int d, int p) {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = d;
    temp->priority = p;
    temp->next = NULL;
    return temp;
}

// Function to check the queue is empty
int isEmpty(Node* head) {
    return (head == NULL);
}

// Function to insert an element in queue with priority
void insert(Node** head, int d, int p) {
    Node* start = (*head);
    Node* temp = newNode(d, p);

    if ((*head)->priority > p) {
        temp->next = *head;
        (*head) = temp;
    } else {
        while (start->next != NULL && start->next->priority < p) {
            start = start->next;
        }
        temp->next = start->next;
        start->next = temp;
    }
}
```

```
// Function to remove element with the highest priority
void pop(Node** head) {
    Node* temp = *head;
    (*head) = (*head)->next;
    free(temp);
}

// Function to show the element with the highest priority
int peek(Node** head) {
    return (*head)->data;
}

int main() {
    Node* pq = newNode(4, 1);
insert(&pq, 5, 2);
insert(&pq, 6, 3);
insert(&pq, 7, 0);

    while (!isEmpty(pq)) {
printf("%d ", peek(&pq));
        pop(&pq);
    }

return ;
}
```

## 29. Write a program to implement linked list operations.

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

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

struct Node* head;

void Insert(int x) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = x;
   temp->next = head;
   head = temp;
}

void Print() {
   struct Node* temp = head;
printf("List is: ");
while(temp != NULL) {
printf("%d ", temp->data);
      temp = temp->next;
   }
printf("\n");
}

int main() {
   head = NULL;
printf("How many numbers?\n");
   int n, i, x;
scanf("%d", &n);
for(i = 0; i< n; i++) {
printf("Enter the number \n");
scanf("%d", &x);
      Insert(x);
Print();
   }
}
```

## 30. Write a program to implement a stack using linked lists.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
void push(struct Node** top_ref, int new_data) {
struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
if (new_node == NULL) {
printf("Stack overflow \n");
exit(0);
}
new_node->data = new_data;
new_node->next = (*top_ref);
(*top_ref) = new_node;
}
int pop(struct Node** top_ref) {
char res;
struct Node* top;

if (*top_ref == NULL) {
printf("Stack underflow \n");
exit(0);
} else {
top = *top_ref;
res = top->data;
*top_ref = top->next;
free(top);
return res;
```

```
}
}
int main() {
struct Node* top = NULL;
push(&top, 10);
push(&top, 20);
push(&top, 30);
printf("%d popped from stack\n", pop(&top));
return 0;
}
```

## 31. Write a program to implement a queue using linked lists.

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

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

struct Node *front = NULL;
struct Node *rear = NULL;

void enqueue(int x) {
   struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
  temp->data = x;
  temp->next = NULL;

if(front == NULL && rear == NULL){
      front = rear = temp;
```

```
return;
    }

    rear->next = temp;
    rear = temp;
}

void dequeue() {
    struct Node *temp = front;
if(front == NULL) {
printf("Queue is Empty\n");
return;
    }
if(front == rear) {
        front = rear = NULL;
    }
    else {
        front = front->next;
    }
    free(temp);
}

    void print() {
       struct Node *temp = front;
    while(temp != NULL) {
    printf("%d ",temp->data);
           temp = temp->next;
       }
    printf("\n");
    }
```

```
int main() {
enqueue(1); print();
enqueue(2); print();
enqueue(3); print();
dequeue(); print();
return ;
}
```

## 32. Write a program to reverse a type of data structure known as linked list.

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

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

struct Node* reverse(struct Node* head) {
   struct Node *current, *prev, *next;
   current = head;
prev = NULL;
while(current != NULL) {
      next = current->next;
      current->next = prev;
prev = current;
      current = next;
   }
   head = prev;
return head;
```

```
}

void print(struct Node* head) {
while(head != NULL) {
printf("%d ", head->data);
        head = head->next;
    }
printf("\n");
}

int main() {
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 1;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = malloc(sizeof(struct Node));
    head->next->next->data = 3;
    head->next->next->next = NULL;

    head = reverse(head);
    print(head);

return ;
}
```

## 33. Write a program to create and traverse a doubly linked list.

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

struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
};


   void append(struct Node** head_ref, int new_data) {
      struct Node* new_node = (struct Node*) malloc(sizeof(struct  Node));
      struct Node* last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;

      if (*head_ref == NULL) {
   new_node->prev = NULL;
         *head_ref = new_node;
   return;
      }

      while (last->next != NULL) {
         last = last->next;
      }

      last->next = new_node;
   new_node->prev = last;

return;
}
```

```
void printList(struct Node* node) {
   while (node != NULL) {
printf("%d ", node->data);
      node = node->next;
   }
}

int main() {

   struct Node* head = NULL;
append(&head, 1);
append(&head, 2);
append(&head, 3);

printList(head);

return ;
}
```

## 34. Write a program to create and traverse a circular linked list.

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

// Node structure
struct Node {
   int data;
   struct Node *next;
};

struct Node *createNode(int data) {
   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
newNode->data = data;
newNode->next = newNode;
   return newNode;
}

void insertNode(struct Node **head, int data) {
   struct Node *newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
   } else {
      struct Node *temp = *head;
      while (temp->next != *head) {
         temp = temp->next;
      }
      temp->next = newNode;
newNode->next = *head;
   }
}

void traverseList(struct Node *head) {
   if (head == NULL) {
printf("List is empty.\n");
return;
   }
   struct Node *temp = head;
   do {
printf("%d -> ", temp->data);
      temp = temp->next;
   } while (temp != head);
printf("HEAD\n");
```

```
}

int main() {
    struct Node *head = NULL;

insertNode(&head, 1);
insertNode(&head, 2);
insertNode(&head, 3);
insertNode(&head, 4);
insertNode(&head, 5);

traverseList(head);
return ;
}
```

**35. Write a program to implement operations on an algoritham known as binary search tree.**

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

typedef struct node {
   int key;
   struct node *left, *right;
} Node;

// Function to create a new Node
Node* newNode(int item) {
   Node* temp = (Node*)malloc(sizeof(Node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}

// Function to perform in-order traversal of the BST
void inorder(Node* root) {
   if (root != NULL) {
inorder(root->left);
printf("%d \n", root->key);
inorder(root->right);
   }
}
```

```
// Function to insert a new node in the BST
Node* insert(Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}
```

```
// Function to find the node with the minimum key value, this node will be the left most leaf
Node* minValueNode(Node* node) {
    Node* current = node;

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

    return current;
}

// Function to delete a node in the BST
Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
```

```
    else {
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }

        Node* temp = minValueNode(root->right);

        root->key = temp->key;

        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}

int main() {
    Node* root = NULL;
    root = insert(root, 50);
insert(root, 30);
insert(root, 20);
insert(root, 40);
insert(root, 70);
insert(root, 60);
insert(root, 80);
```

```
printf("Inorder traversal of the BST:\n");
inorder(root);

printf("\nDelete 20\n");
   root = deleteNode(root, 20);
printf("Inorder traversal of the BST:\n");
inorder(root);

return ;
}
```

## 36. Write a program to implement depth-first search in a graph.

```
#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 point to an array
   struct node** adjLists;
};

struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
```

```
void printGraph(struct Graph* graph);
void DFS(struct Graph* graph, int vertex);

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;
}

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;
    }
}

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

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;
}
```

```
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");
   }
}
```

## 37. Write a program to implement breadth-first search in a graph.

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

#define SIZE 40
struct queue {
   int items[SIZE];
   int front;
   int rear;
};

struct queue* createQueue();
```

```
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);

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);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);
void bfs(struct Graph* graph, int startVertex);

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

```
addEdge(graph, 3, 3);

bfs(graph, 2);

   return 0;
}

void bfs(struct Graph* graph, int startVertex) {
   struct queue* q = createQueue();

   graph->visited[startVertex] = 1;
enqueue(q, startVertex);

   while (!isEmpty(q)) {
printQueue(q);
      int currentVertex = dequeue(q);
printf("Visited %d\n", currentVertex);

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

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

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

```
}

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

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 queue* createQueue() {
   struct queue* q = malloc(sizeof(struct queue));
   q->front = -1;
   q->rear = -1;
   return q;
}

int isEmpty(struct queue* q) {
   if (q->rear == -1)
      return 1;
   else
      return 0;
}

void enqueue(struct queue* q, int value) {
   if (q->rear == SIZE - 1)
printf("\nQueue is Full!!");
   else {
      if (q->front == -1)
         q->front = 0;
      q->rear++;
      q->items[q->rear] = value;
   }
}
```

```
int dequeue(struct queue* q) {
   int item;
   if (isEmpty(q)) {
printf("Queue is empty");
      item = -1;
   } else {
      item = q->items[q->front];
      q->front++;
      if (q->front > q->rear) {
printf("Resetting queue ");
         q->front = q->rear = -1;
      }
   }
   return item;
}

void printQueue(struct queue* q) {
   int i = q->front;

   if (isEmpty(q)) {
printf("Queue is empty");
   } else {
printf("\nQueue contains \n");
      for (i = q->front; i< q->rear + 1; i++) {
printf("%d ", q->items[i]);
      }
   }
}
```

## 38. Write a program code in ‘C’ to applying Dijkstra's algorithm.

```
#include <stdio.h>
#include <limits.h>
#define V 9
int minDistance(int dist[], int sptSet[]) {
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (sptSet[v] == 0 &&dist[v] <= min)
min = dist[v], min_index = v;
return min_index;
}
void printSolution(int dist[]) {
printf("Vertex \t\t Distance from Source\n");
for (int i = 0; i< V; i++)
printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src) {
int dist[V];
int sptSet[V];
for (int i = 0; i< V; i++)
dist[i] = INT_MAX, sptSet[i] = 0;

dist[src] = 0;
for (int count = 0; count < V - 1; count++) {
int u = minDistance(dist, sptSet);
sptSet[u] = 1;
for (int v = 0; v < V; v++)
if (!sptSet[v] && graph[u][v] &&dist[u] != INT_MAX &&dist[u] + graph[u][v]
<dist[v])
dist[v] = dist[u] + graph[u][v];
}
```

```
printSolution(dist);
}
int main() {
int graph[V][V] = { {0, 4, 0, 0, 0, 0, 0, 8, 0},
{4, 0, 8, 0, 0, 0, 0, 11, 0},
{0, 8, 0, 7, 0, 4, 0, 0, 2},
{0, 0, 7, 0, 9, 14, 0, 0, 0},
{0, 0, 0, 9, 0, 10, 0, 0, 0},
{0, 0, 4, 14, 10, 0, 2, 0, 0},
{0, 0, 0, 0, 0, 2, 0, 1, 6},
{8, 11, 0, 0, 0, 0, 1, 0, 7},
{0, 0, 2, 0, 0, 0, 6, 7, 0}
};

dijkstra(graph, 0);

return 0;
}
```

## 39. Write a program code in 'C' programming language to implement matrix multiplication.

```
#include<stdio.h>

#define SIZE 3

void main() {
   int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];
   int i, j, k;

printf("Enter elements for matrix A:\n");
```

```
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
scanf("%d", &A[i][j]);
        }
    }

printf("Enter elements for matrix B:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
scanf("%d", &B[i][j]);
        }
    }

for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
            C[i][j] = 0;
for(k=0; k<SIZE; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

printf("Resultant Matrix:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
printf("%d ", C[i][j]);
        }
printf("\n");
    }
}
```

## 40. Write a program code in ‘C’ programming language to find the transpose of a matrix.

```
#include<stdio.h>

#define SIZE 3

void main() {
   int matrix[SIZE][SIZE], transpose[SIZE][SIZE];
   int i, j;

printf("Enter elements for matrix:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
scanf("%d", &matrix[i][j]);
      }
   }

for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
         transpose[j][i] = matrix[i][j];
      }
   }
printf("Transpose of Matrix:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
printf("%d ", transpose[i][j]);
      }
printf("\n");
   }
}
```

## 41. Write a program to find the determinant of a matrix.

```
#include<stdio.h>


void main() {
   int matrix[3][3];
   int i, j;
   int determinant;


printf("Enter elements for 3x3 matrix:\n");
for(i=0; i<3; i++) {
for(j=0; j<3; j++) {
scanf("%d", &matrix[i][j]);
      }
   }


   determinant = matrix[0][0] * ((matrix[1][1]*matrix[2][2]) -
(matrix[2][1]*matrix[1][2])) - matrix[0][1] * (matrix[1][0]
            * matrix[2][2] - matrix[2][0] * matrix[1][2]) + matrix[0][2] * (matrix[1][0] *
matrix[2][1] - matrix[2][0] * matrix[1][1]);


printf("Determinant of the Matrix : %d", determinant);
}
```

## 42. Write a program to add and subtract two matrices.

```
#include<stdio.h>

#define SIZE 3

void main() {
   int A[SIZE][SIZE], B[SIZE][SIZE], sum[SIZE][SIZE], diff[SIZE][SIZE];
   int i, j;

printf("Enter elements for matrix A:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
scanf("%d", &A[i][j]);
      }
   }

printf("Enter elements for matrix B:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
scanf("%d", &B[i][j]);
      }
   }

for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
         sum[i][j] = A[i][j] + B[i][j];
         diff[i][j] = A[i][j] - B[i][j];
      }
   }
```

```
printf("Sum of Matrices:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
printf("%d ", sum[i][j]);
        }
printf("\n");
    }

printf("Difference of Matrices:\n");
for(i=0; i<SIZE; i++) {
for(j=0; j<SIZE; j++) {
printf("%d ", diff[i][j]);
        }
printf("\n");
    }
}
```

## 43. Write a program to implement a singly linked list and its operations.

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

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

void insertAtBeginning(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
new_node->data = new_data;
new_node->next = (*head);
```

```
    (*head) = new_node;
}

void printList(struct Node *node) {
    while (node != NULL) {
printf(" %d ", node->data);
        node = node->next;
    }
}

int main() {
    struct Node* head = NULL;

insertAtBeginning(&head, 7);
insertAtBeginning(&head, 1);
insertAtBeginning(&head, 3);
insertAtBeginning(&head, 2);

printf("Linked list is: ");
printList(head);

return ;
}
```

## 44. Write a program to convert a decimal number to binary.

```
#include <stdio.h>

void decimalToBinary(int n) {
  if (n == 0)
return;

decimalToBinary(n/2);

printf("%d", n%2);
}

int main() {
  int n;
printf("Enter a decimal number: ");
scanf("%d", &n);
printf("Binary of %d is: ", n);
decimalToBinary(n);

return ;
}
```

## 45. Write a program to convert a binary number to decimal.

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

int binaryToDecimal(int n) {
   int decimal = 0, i = 0, remainder;
   while (n!=0) {
       remainder = n%10;
```

```
        n /= 10;
        decimal += remainder*pow(2,i);
        ++i;
    }
    return decimal;
}

int main() {
    int n;
printf("Enter a binary number: ");
scanf("%d", &n);
printf("Decimal of %d is: %d ", n, binaryToDecimal(n));

return ;
}
```

## 46. Write a program to convert a decimal number to hexadecimal.

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

void decimalToHexadecimal(int n) {
  if (n == 0)
return;

  int temp  = 0;
  temp = n % 16;

decimalToHexadecimal(n/16);

  if (temp < 10)
printf("%d", temp);
```

```
  else
printf("%c", temp + 55);
}

int main() {
  int n;
printf("Enter a decimal number: ");
scanf("%d", &n);
printf("Hexadecimal of %d is: ", n);
decimalToHexadecimal(n);

return ;
}
```

## 47. Write a program code in ‘C’ programming language to calculate the power of a number.

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

int main() {
   double base, exponent, result;
printf("Enter a base number: ");
scanf("%lf", &base);
printf("Enter an exponent: ");
scanf("%lf", &exponent);

   // Calculating power using pow() function
   result = pow(base, exponent);

printf("%.1lf^%.1lf = %.2lf", base, exponent, result);
```

```
return ;
}
```

## 48. Write a program to implement the Fibonacci sequence.

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

int main() {
   int i, n, t1 = 0, t2 = 1, nextTerm;
printf("Enter the number of terms: ");
scanf("%d", &n);

printf("Fibonacci Series: ");

   for (i = 1; i<= n; ++i) {
printf("%d, ", t1);
nextTerm = t1 + t2;
      t1 = t2;
      t2 = nextTerm;
   }
return ;
}
```

## 49. Write a program to check if a string is palindrome or not.

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

void isPalindrome(char str[]) {
   int l = 0;
   int h = strlen(str) - 1;

   while (h > l) {
      if (str[l++] != str[h--]) {
printf("%s is Not a Palindrome\n", str);
return;
      }
   }
printf("%s is a palindrome\n", str);
}

int main() {
   char str[100];
printf("Enter a string to check if it is a palindrome: ");
scanf("%s", str);

isPalindrome(str);
return ;
}
```

**50. Write a program code in 'C' programming language to find the GCD of two given numbers.**

```
#include <stdio.h>

int gcd(int n1, int n2) {
   if (n2 != 0)
      return gcd(n2, n1 % n2);
   else
      return n1;
}

int main() {
   int n1, n2;
printf("Enter two positive integers: ");
scanf("%d %d", &n1, &n2);

printf("GCD of %d and %d is %d", n1, n2, gcd(n1,n2));
return ;
}
```

**51. Write a program to calculate the LCM of two numbers.**

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

int main() {
   int num1, num2, max, lcm=1, i=2;

printf("Enter two numbers: ");
scanf("%d %d", &num1, &num2);

   max = (num1 > num2) ? num1 : num2;
```

```
while(1) {
if(max%num1==0 && max%num2==0) {
        lcm = max;
break;
      }
      max++;
   }

printf("LCM of %d and %d is %d", num1, num2, lcm);

return ;
}
```

**52. Write a program code in 'C' programming language to print the ASCII value of a character.**

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

int main() {
   char ch;

printf("Enter a character: ");
scanf("%c", &ch);

printf("The ASCII value of %c is %d", ch, ch);

return ;
}
```

## 53. Write a program to print all ASCII values and their equivalent characters.

```
#include<stdio.h>

int main() {
   char ch;

for(ch = 0; ch<= 255; ch++) {
printf("ASCII value of character %c = %d\n", ch, ch);
   }

return ;
}
```

## 54. Write a program code in ‘C’ programming language to find and display all prime numbers within a range.

```
#include<stdio.h>
int check_prime(int num) {
   int i;
   if (num == 1) {
      return 0;
   }
   for (i=2; i*i<=num; i++) {
      if (num % i == 0) {
         return 0;
      }
   }
   return 1;
}
int main() {
   int start, end, i;


printf("Enter the range (start end): ");
scanf("%d %d", &start, &end);
```

```
printf("Prime numbers between %d and %d are: ", start, end);

   for (i=start; i<=end; i++) {
      if (check_prime(i)) {
printf("%d ", i);
      }
   }
return ;
}
```

## 55. Write a program code in ‘C’ programming language to count the number of words in a string.

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

int main() {
   char str[100];
   int i, wordCount = 1;

printf("Enter a string: ");
   gets(str);

for(i = 0; str[i] != '\0'; ++i) {
      if(str[i] == ' ' || str[i] == '\n' || str[i] == '\t')
wordCount++;
   }
printf("Number of words in the string: %d", wordCount);
return ;
}
```

## 56. Write a program to convert uppercase letters to lowercase in a string.

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

int main() {
   char str[100];
   int i;

printf("Enter a string: ");
   gets(str);

for(i = 0; str[i]; i++) {
      str[i] = tolower(str[i]);
   }

printf("Lowercase string: %s", str);
return ;
}
```

## 57. Write a program to implement a basic calculator using switch case.

```
#include<stdio.h>

int main() {
   char operator;
   double first, second;

printf("Enter an operator (+, -, *, /): ");
scanf("%c", &operator);
printf("Enter two operands: ");
scanf("%lf %lf", &first, &second);
```

```
   switch(operator) {
       case '+':
printf("%.1lf + %.1lf = %.1lf", first, second, first + second);
break;
       case '-':
printf("%.1lf - %.1lf = %.1lf", first, second, first - second);
break;
       case '*':
printf("%.1lf * %.1lf = %.1lf", first, second, first * second);
break;
       case '/':
if(second != 0)
printf("%.1lf / %.1lf = %.1lf", first, second, first / second);
           else
printf("Error! Division by zero is not allowed.");
break;
       default:
printf("Error! Invalid operator");
   }

return ;
}
```

## 58. Write a program to calculate the number of vowels in a string.

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

int main() {
   char str[100];
   int i, vowelCount = 0;

printf("Enter a string: ");
   gets(str);

for(i = 0; str[i]; i++) {
      if(str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' ||
        str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U') {
vowelCount++;
      }
   }

printf("Number of vowels in the string: %d", vowelCount);
return ;
}
```

## 59. Write a program to count the number of digits in a number.

```
#include<stdio.h>

int main() {
   long longnum;
   int count = 0;

printf("Enter a number: ");
scanf("%lld", &num);

while(num != 0) {
num /= 10;
      ++count;
   }

printf("Number of digits: %d", count);
return ;
}
```

## 60. Write a program to print the multiplication table of a given number.

```
#include <stdio.h>

int main() {
   int num, i;

printf("Enter the number: ");
scanf("%d",&num);

for(i = 1; i<= 10; i++) {
printf("%d * %d = %d\n", num, i, num*i);
```

```
    }

return ;
}
```

## 61. Write a program to implement a simple ticket booking system using structure.

```
#include <stdio.h>
struct Ticket {
    char name[50];
    int seat_number;
};

int main() {
    struct Ticket t1;

printf("Enter your name: ");
scanf("%s", t1.name);

printf("Enter your seat number: ");
scanf("%d", &t1.seat_number);

printf("Ticket booked successfully!\n");
printf("Name: %s\n", t1.name);
printf("Seat Number: %d\n", t1.seat_number);

return ;
}
```

## 62. Write a program to implement a dynamic memory allocation for an integer array.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int *array;
   int size, i;

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

   array = (int*)malloc(size * sizeof(int));
   if (array == NULL) {
printf("Memory not allocated.\n");
      return -1;
   }

printf("Enter elements of array: ");
   for (i = 0; i< size; i++) {
scanf("%d", &array[i]);
   }

printf("Elements of array are: ");
   for (i = 0; i< size; i++) {
printf("%d ", array[i]);
   }

   free(array);
return ;
}
```

## 63. Write a program code in ‘C’ programming language to find the sum of the main diagonal components of a matrix.

```
#include<stdio.h>

int main() {
   int matrix[3][3];
   int i, j, sum = 0;

printf("Enter elements of the matrix:\n");
for(i=0; i<3; i++) {
for(j=0; j<3; j++) {
printf("Enter element [%d][%d]: ", i, j);
scanf("%d", &matrix[i][j]);
      }
   }

for(i=0; i<3; i++) {
for(j=0; j<3; j++) {
if(i == j) {
            sum += matrix[i][j];
         }
      }
   }

printf("The sum of diagonal elements of the matrix is: %d\n", sum);

return ;
}
```

## 64. Write a program code in ‘C’ programming language to find the sum of the off-diagonal components of a matrix.

```
#include<stdio.h>

int main() {
   int matrix[3][3];
   int i, j, sum = 0;

printf("Enter elements of the matrix:\n");
for(i=0; i<3; i++) {
for(j=0; j<3; j++) {
printf("Enter element [%d][%d]: ", i, j);
scanf("%d", &matrix[i][j]);
      }
   }

for(i=0; i<3; i++) {
for(j=0; j<3; j++) {
if(i != j) {
            sum += matrix[i][j];
         }
      }
   }

printf("The sum of off-diagonal elements of the matrix is: %d\n", sum);

return ;
}
```

## 65. Write a program to implement a hashing technique.

```
#include<stdio.h>
#define SIZE 10
void init(int arr[]) {
   int i;
for(i = 0; i< SIZE; i++) {
arr[i] = -1;
   }
}
void insert(int arr[], int value) {
   int key = value % SIZE;

   if(arr[key] == -1) {
arr[key] = value;
printf("%d inserted at index %d\n", value, key);
   } else {
printf("Collision : element %d cannot be inserted at index %d\n", value, key);
   }
}
int main() {
   int arr[SIZE];
init(arr);
insert(arr, 10);
insert(arr, 25);
insert(arr, 15);
insert(arr, 7);
insert(arr, 33);

return ;
}
```

## 66. Write a program to implement a phone directory system using trees.

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

typedef struct Node {
    char name[100];
    char phone[15];
    struct Node* left;
    struct Node* right;
} Node;

Node* newNode(char* name, char* phone) {
    Node* node = (Node*)malloc(sizeof(Node));
strcpy(node->name, name);
strcpy(node->phone, phone);
    node->left = NULL;
    node->right = NULL;

    return node;
}

Node* insert(Node* node, char* name, char* phone) {
if(node == NULL)
        return newNode(name, phone);
if(strcmp(name, node->name) < 0)
        node->left = insert(node->left, name, phone);
    else if(strcmp(name, node->name) > 0)
        node->right = insert(node->right, name, phone);
```

```
    return node;
}

void search(Node* root, char* name) {
if(root != NULL) {
        if(strcmp(root->name, name) == 0)
printf("Found: %s, Phone Number: %s\n", root->name, root->phone);
        else if(strcmp(name, root->name) < 0)
            search(root->left, name);
        else
            search(root->right, name);
    } else {
printf("Contact not found!\n");
    }
}

int main() {
    Node* root = NULL;
    root = insert(root, "shivam", "1234567890");
insert(root, "keni", "2345678901");
insert(root, "krunal", "3456789012");
insert(root, "Bob", "4567890123");

search(root, "Bob");
return ;
}
```

## 67. Write a program to implement the concept of structures in C.

```
#include <stdio.h>

struct Student {
   char name[50];
   int roll_no;
   float marks;
};

int main() {
   struct Student s1;

printf("Enter name: ");
   gets(s1.name);

printf("Enter roll number: ");
scanf("%d", &s1.roll_no);

printf("Enter marks: ");
scanf("%f", &s1.marks);

printf("Displaying Information:\n");
printf("Name: %s\n", s1.name);
printf("Roll: %d\n", s1.roll_no);
printf("Marks: %.1f\n", s1.marks);

return ;
}
```

## 68. Write a program code in C to demonstrate the concept of unions in C.

```
#include <stdio.h>

union Data {
   int i;
   float f;
   char str[20];
};

int main() {
   union Data data;

data.i = 10;
printf("data.i: %d\n", data.i);

data.f = 220.5;
printf("data.f: %.1f\n", data.f);

strcpy(data.str, "C Programming");
printf("data.str: %s\n", data.str);

return ;
}
```

## 69. Write a program to implement the concept of enumerated data type.

```
#include<stdio.h>
enum weekdays {
   Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
};

int main() {
enum weekdays day;
   day = Wednesday;
printf("The day is: %d", day);
return ;
}
```

## 70. Write a program to calculate the sum of n numbers entered by the user.

```
#include<stdio.h>
int main() {
   int n, i, sum = 0;
printf("Enter the number of elements: ");
scanf("%d", &n);
   int numbers[n];

for(i=0; i<n; i++) {
printf("Enter number %d: ", i+1);
scanf("%d", &numbers[i]);
      sum += numbers[i];
   }
printf("The sum is: %d", sum);
return ;
}
```

## 71. Write a program to read and print elements of an array.

```
#include<stdio.h>

int main() {
   int n, i;
printf("Enter the number of elements: ");
scanf("%d", &n);
   int numbers[n];

for(i=0; i<n; i++) {
printf("Enter element %d: ", i+1);
scanf("%d", &numbers[i]);
   }
printf("Elements of array: ");
for(i=0; i<n; i++) {
printf("%d ", numbers[i]);
   }
return ;
}
```

## 72. Write a program to find the reverse of an array.

```
#include<stdio.h>

int main() {
   int n, i;
printf("Enter the number of elements: ");
scanf("%d", &n);
   int numbers[n];
```

```
for(i=0; i<n; i++) {
printf("Enter element %d: ", i+1);
scanf("%d", &numbers[i]);
   }
printf("Reversed array: ");
for(i=n-1; i>=0; i--) {
printf("%d ", numbers[i]);
   }
return ;
}
```

## 73. Write a program to put even and odd elements of an array in two separate arrays.

```
#include<stdio.h>

int main() {
   int n, i, j=0, k=0;
printf("Enter the number of elements: ");
scanf("%d", &n);
   int numbers[n], even[n], odd[n];

for(i=0; i<n; i++) {
printf("Enter element %d: ", i+1);
scanf("%d", &numbers[i]);
      if(numbers[i]%2 == 0) {
         even[j] = numbers[i];
j++;
      } else {
         odd[k] = numbers[i];
         k++;
      }
```

```
    }
printf("Even elements: ");
for(i=0; i<j; i++) {
printf("%d ", even[i]);
    }
printf("\nOdd elements: ");
for(i=0; i<k; i++) {
printf("%d ", odd[i]);
    }
return ;
}
```

## 74. Write a program to implement a doubly ended queue.

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

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

void insertAtFront(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
new_node->data  =new_data;
new_node->next = (*head_ref);
  (*head_ref) = new_node;
}

void insertAtEnd(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
```

```
new_node->data  =new_data;
new_node->next = NULL;

  if (*head_ref == NULL) {
    *head_ref = new_node;
return;
  }

  struct Node *last = *head_ref;
  while (last->next != NULL)
    last = last->next;

  last->next = new_node;
}

void displayQueue(struct Node* node) {
  while (node != NULL) {
printf(" %d ", node->data);
     node = node->next;
  }
}

int main() {
  struct Node* head = NULL;

insertAtEnd(&head, 7);
insertAtFront(&head, 1);
insertAtEnd(&head, 9);
insertAtFront(&head, 2);
```

```
printf("Created Doubly Ended Queue is: ");
displayQueue(head);

return ;
}
```

## 75. Write a program to implement binary search tree traversal in Preorder, Inorder, and Postorder.

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

struct Node {
  int data;
  struct Node* left, *right;
};

struct Node* newNode(int item) {
  struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
  temp->data = item;
  temp->left = temp->right = NULL;
  return temp;
}

void preorder(struct Node* root) {
  if (root != NULL) {
printf("%d \n", root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
```

```
void inorder(struct Node* root) {
  if (root != NULL) {
inorder(root->left);
printf("%d \n", root->data);
inorder(root->right);
  }
}

void postorder(struct Node* root) {
  if (root != NULL) {
postorder(root->left);
postorder(root->right);
printf("%d \n", root->data);
  }
}

struct Node* insert(struct Node* Node, int data) {
  if (Node == NULL) return newNode(data);
  if (data < Node->data) Node->left  = insert(Node->left, data);
  else if (data > Node->data) Node->right = insert(Node->right, data);
  return Node;
}

int main() {
  struct Node* root = NULL;
  root = insert(root, 8);
insert(root, 3);
insert(root, 10);
insert(root, 1);
```

```
insert(root, 6);
insert(root, 14);
insert(root, 4);
insert(root, 7);
insert(root, 13);

printf("Preorder traversal \n");
  preorder(root);

printf("\nInorder traversal \n");
inorder(root);

printf("\nPostorder traversal \n");
postorder(root);

return ;
}
```

## 76. Write a program to count the number of leaf nodes in an operation known as binary tree.

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

struct Node {
  int data;
  struct Node* left, *right;
};

struct Node* newNode(int data) {
  struct Node* node = (struct Node*)malloc(sizeof(struct Node));
  node->data = data;
```

```
  node->left = NULL;
  node->right = NULL;
  return(node);
}
int getLeafCount(struct Node* node) {
if(node == NULL) return 0;
  if(node->left == NULL && node->right==NULL) return 1;
  else return getLeafCount(node->left) + getLeafCount(node->right);
}
int main() {
  struct Node* root = newNode(1);
  root->left = newNode(2);
  root->right = newNode(3);
  root->left->left = newNode(4);
  root->left->right = newNode(5);
printf("Leaf count of the tree is %d", getLeafCount(root));

return ;
}
```

## 77. Write a program code in C to find the height of an algoritham known as binary tree.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* left, *right;
};
struct Node* newNode(int data) {
  struct Node* node = (struct Node*)malloc(sizeof(struct Node));
  node->data = data;
```

```
  node->left = NULL;
  node->right = NULL;
  return(node);
}
int maxDepth(struct Node* node) {
  if (node==NULL) return 0;
  else {
    int lDepth = maxDepth(node->left);
    int rDepth = maxDepth(node->right);
    if (lDepth>rDepth) return(lDepth+1);
    else return(rDepth+1);
  }
}

int main() {
  struct Node* root = newNode(1);
  root->left = newNode(2);
  root->right = newNode(3);
  root->left->left = newNode(4);
  root->left->right = newNode(5);
printf("Height of tree is %d", maxDepth(root));
return ;
}
```

**78. Write a program to find the number of nodes in an algoritham known as binary tree.**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* left, *right;
};
struct Node* newNode(int data) {
  struct Node* node = (struct Node*)malloc(sizeof(struct Node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;
  return(node);
}

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

int main() {
  struct Node* root = newNode(1);
  root->left = newNode(2);
  root->right = newNode(3);
  root->left->left = newNode(4);
  root->left->right = newNode(5);

printf("The size of the tree is %d", getSize(root));
return ;
}
```

## 79. Write a program to find the sum of all nodes in an algoritham known binary tree.

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};
struct Node* createNode(int data) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return node;
}

int sumOfNodes(struct Node* node) {
if(node == NULL)
      return 0;
   return (node->data + sumOfNodes(node->left) + sumOfNodes(node->right));
}
int main() {
   struct Node* root = createNode(1);
   root->left = createNode(2);
   root->right = createNode(3);
   root->left->left = createNode(4);
   root->left->right = createNode(5);
printf("Sum of all nodes = %d", sumOfNodes(root));
return ;
}
```

## 80. Write a program to implement an AVL tree.

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};
struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return(node);
}
int max(int a, int b) {
    return (a > b)? a : b;
}
int height(struct Node *N) {
    if (N == NULL)
        return 0;
    return N->height;
}
```

```
struct Node *rightRotate(struct Node *y) {
    struct Node *x = y->left;
    struct Node *T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}

struct Node *leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}

int getBalance(struct Node *N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}
```

```
struct Node* insert(struct Node* node, int key) {
    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else // Equal keys are not allowed in BST
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    // Left Right Case
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    // Right Left Case
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
```

```
    // return the (unchanged) node pointer
    return node;
}

// function to print preorder traversal of the tree.
void preOrder(struct Node *root) {
if(root != NULL) {
printf("%d ", root->key);
preOrder(root->left);
preOrder(root->right);
    }
}
int main() {
    struct Node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
printf("Preorder traversal of the constructed AVL tree is \n");
preOrder(root);
return ;
}
```

## 81. Write a program to implement a red-black tree.

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

enumnodeColor {
   RED,
   BLACK
};

struct rbNode {
   int data, color;
   struct rbNode *link[2];
};

struct rbNode *root = NULL;

// Create a red-black tree
struct rbNode *createNode(int data) {
   struct rbNode *newnode;
newnode = (struct rbNode*)malloc(sizeof(struct rbNode));
newnode->data = data;
newnode->color = RED;
newnode->link[0] = newnode->link[1] = NULL;
   return newnode;
}
```

```
// Insert a node
struct rbNode *insertNode(struct rbNode *root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data < root->data) {
        root->link[0] = insertNode(root->link[0], data);
    } else if (data > root->data) {
        root->link[1] = insertNode(root->link[1], data);
    }

    return root;
}

// Display the in-order of the tree
void inorderTraversal(struct rbNode *node) {
    if (node) {
inorderTraversal(node->link[0]);
printf("%d  ", node->data);
inorderTraversal(node->link[1]);
    }
}

int main() {
    int n, i, data;
printf("Enter the number of elements:\n");
scanf("%d", &n);
```

```
printf("Enter tree data:\n");
for(i = 0; i< n; i++) {
scanf("%d", &data);
      root = insertNode(root, data);
   }

printf("Inorder Traversal of Red Black Tree:\n");
inorderTraversal(root);

return ;
}
```

## 82. Write a program to implement a B-tree.

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

#define ORDER 3

typedef struct node {
   int n; // Number of keys in node
   int keys[ORDER - 1]; // Array of keys
   struct node *children[ORDER]; // Array of children
} Node;

// Create a new node
Node* createNode() {
   Node *x = (Node*)malloc(sizeof(Node));
   x->n = 0;
```

```
    x->children[0] = NULL;
    return x;
}

// Insert node into tree
void insert(Node **root, int k) {
    Node *r = *root;
    if (r->n == ORDER - 1) {
        Node *temp = createNode();
        *root = temp;
        temp->children[0] = r;
splitChild(temp, 0);
insertNonFull(temp, k);
    } else {
insertNonFull(r, k);
    }
}

// Insert nonfull node into tree
void insertNonFull(Node *x, int k) {
    int i = x->n;
    if (x->children[0] == NULL) {
        x->keys[i] = k;
        x->n++;
    } else {
        while (i>= 1 && k < x->keys[i - 1]) {
i--;
        }
i++;
        if (x->children[i]->n == ORDER - 1) {
```

```
splitChild(x, i);
          if (k > x->keys[i - 1]) {
i++;
          }
       }
insertNonFull(x->children[i], k);
   }
}

// Split the child of a node
void splitChild(Node *x, int i) {
   Node *t = x->children[i];
   Node *y = createNode();
   x->children[i + 1] = y;
   x->keys[i] = t->keys[ORDER / 2 - 1];
   y->n = ORDER / 2 - 1;
   x->n++;
   int j;
   for (j = 0; j < ORDER / 2 - 1; j++) {
      y->keys[j] = t->keys[j + ORDER / 2];
   }
   if (t->children[0] != NULL) {
      for (j = 0; j < ORDER / 2; j++) {
         y->children[j] = t->children[j + ORDER / 2];
      }
   }
   t->n = ORDER / 2 - 1;
}

// Print the tree
```

```
void printTree(Node *root, int l) {
   int i;
   for (i = 0; i< root->n; i++) {
      if (root->children[i] != NULL) {
printTree(root->children[i], l + 1);
      }
printf("%d ", root->keys[i]);
   }
   if (root->children[i] != NULL) {
printTree(root->children[i], l + 1);
   }
}

int main() {
   Node *root = createNode();
   int num;
printf("Enter numbers to insert into B-Tree (0 to stop): ");
   while (1) {
scanf("%d", &num);
      if (num == 0) {
break;
      }
insert(&root, num);
   }
printf("The B-Tree is: ");
printTree(root, 0);
printf("\n");
return ;
}
```

## 83. Write a program to check if a tree is balanced or not.

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

struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int data) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;

   return node;
}

int max(int a, int b) {
   return (a > b)? a: b;
}

int height(struct Node* node) {
if(node == NULL)
      return 0;
   return 1 + max(height(node->left), height(node->right));
}

int isBalanced(struct Node* node) {
```

```
if(node == NULL)
        return 1;

    int leftHeight = height(node->left);
    int rightHeight = height(node->right);

if(abs(leftHeight - rightHeight) <= 1 &&isBalanced(node->left) &&isBalanced(node->right))
        return 1;

    return 0;
}

int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    if(isBalanced(root))
printf("Tree is balanced");
    else
printf("Tree is not balanced");

return;
}
```

## 84. Write a program to implement Kruskal's algorithm.

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

typedef struct Edge {
    int src, dest, weight;
} Edge;

typedef struct Graph {
    int V, E;
    Edge* edge;
} Graph;

Graph* createGraph(int V, int E) {
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph->V = V;
    graph->E = E;
    graph->edge = (Edge*) malloc(sizeof(Edge) * E);
    return graph;
}

typedef struct subset {
    int parent, rank;
} subset;

int find(subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
```

```
}

void Union(subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank< subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank> subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int compareEdges(const void* a, const void* b) {
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}

void Kruskal(Graph* graph) {
    int V = graph->V;
    Edge result[V];
qsort(graph->edge, graph->E, sizeof(graph->edge[0]), compareEdges);

    subset *subsets = (subset*) malloc(V * sizeof(subset));
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
```

```
        subsets[v].rank = 0;
    }

    int e = 0, i = 0;
    while (e < V - 1 &&i< graph->E) {
        Edge next_edge = graph->edge[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            result[e++] = next_edge;
Union(subsets, x, y);
        }
    }

printf("Following are the edges in the constructed MST\n");
    for (i = 0; i< e; ++i) {
printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);
    }
}

// Driver code
int main() {
    int V = 4;
    int E = 5;
    Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
```

```
    graph->edge[0].weight = 10;

    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    Kruskal(graph);
return ;
}
```

## 85. Write a program to implement Prim's algorithm.

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{
   int min = INT_MAX;
   int 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;
}
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]]);
   }
```

```
}
void primMST(int graph[V][V])
{
   int parent[V];
   int key[V];
   bool mstSet[V];

   for (int i = 0; i< V; i++)
   {
      key[i] = INT_MAX;
mstSet[i] = false;
   }

key[0] = 0;
parent[0] = -1;

   for (int count = 0; count < V - 1; count++)
   {
      int u = minKey(key, mstSet);
mstSet[u] = true;
      for (int v = 0; v < V; v++)
      {
         if (graph[u][v] &&mstSet[v] == false && graph[u][v] < key[v])
         {
            parent[v] = u;
            key[v] = graph[u][v];
```

```
            }
        }
    }

printMST(parent, graph);
}

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 }
    };
primMST(graph);
return ;
}
```

## 86. Write a program to implement Bellman-Ford algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define INFINITY 99999
void BellmanFord(int graph[5][5], int V, int E, int edge[20][2])
{
   int distance[V];
   int i,j;
   for (i = 0; i< V; i++)
      distance[i] = INFINITY;
distance[0] = 0;
for(i = 0; i< V-1; i++)
   {
for(j = 0; j < E; j++)
      {
         int u = edge[j][0];
         int v = edge[j][1];
         int weight = graph[u][v];
         if(distance[u] != INFINITY && distance[u] + weight < distance[v])
            distance[v] = distance[u] + weight;
      }
   }

for(i = 0; i< E; i++)
   {
      int u = edge[i][0];
```

```
        int v = edge[i][1];
        int weight = graph[u][v];
        if(distance[u] != INFINITY && distance[u] + weight < distance[v])
printf("Negative Weight Cycle Exists\n");
    }
printf("Vertex\tDistance from Source\n");
for(i = 0; i< V; i++)
    {
printf("%d\t\t%d\n", i, distance[i]);
    }
}

int main()
{
    int V = 5; // Number of vertices
    int graph[5][5] = {
        {0, -1, 4, 0, 0},
        {0, 0, 3, 2, 2},
        {0, 0, 0, 0, 0},
        {0, 1, 5, 0, 0},
        {0, 0, 0, -3, 0}
    };
    int edge[8][2] = {
        {0, 1},
        {0, 2},
        {1, 2},
```

```
        {1, 3},
        {1, 4},
        {3, 1},
        {3, 2},
        {4, 3}
    };
    int E = sizeof(edge)/sizeof(edge[0]);

BellmanFord(graph, V, E, edge);
return ;
}
```

## 87. Write a program to check if a graph is connected or not.

```
#include<stdio.h>

#define MAX 100

int adj[MAX][MAX];
int visited[MAX];

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

int isConnected(int n) {
```

```
for(int i = 0; i< n; i++)
        if(visited[i] == 0)
            return 0;
    return 1;
}

int main() {
    int nodes, edges, v1, v2;
printf("Enter number of nodes and edges:\n");
scanf("%d%d", &nodes, &edges);
for(int i = 0; i< edges; i++) {
printf("Enter edge (Source and Destination) :\n");
scanf("%d%d", &v1, &v2);
adj[v1][v2] = 1;
adj[v2][v1] = 1;
    }
DFS(0, nodes);
    if(isConnected(nodes))
printf("The graph is connected.\n");
    else
printf("The graph is not connected.\n");
return ;
}
```

## 88. Write a program to find the shortest path between two nodes in a graph.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct node {
   int vertex;
   struct node *next;
} node;

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

node* createNode(int);
Graph* createGraph(int);
void addEdge(Graph*, int, int);
void BFS(Graph*, int, int);

node* createNode(int vertex) {
   node *newNode = malloc(sizeof(node));
newNode->vertex = vertex;
newNode->next = NULL;
   return newNode;
}
Graph* createGraph(int vertices) {
   Graph *graph = malloc(sizeof(Graph));
   graph->numVertices = vertices;
   graph->adjLists = malloc(vertices * sizeof(node*));
```

```
    graph->visited = malloc(vertices * sizeof(int));
    for (int i = 0; i< vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}
void addEdge(Graph *graph, int src, int dest) {
    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;
}
void BFS(Graph *graph, int startVertex, int endVertex) {
    int prev[MAX];
    int dist[MAX];
for(int i = 0; i< graph->numVertices; i++) {
prev[i] = -1;
dist[i] = __INT_MAX__;
    }
    graph->visited[startVertex] = 1;
dist[startVertex] = 0;
    int queue[MAX], front = -1, rear = -1;
    queue[++rear] = startVertex;
    while (front != rear) {
        int currentVertex = queue[++front];
        node *temp = graph->adjLists[currentVertex];
        while (temp) {
```

```
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
               queue[++rear] = adjVertex;
               graph->visited[adjVertex] = 1;
prev[adjVertex] = currentVertex;
dist[adjVertex] = dist[currentVertex] + 1;
            }
            temp = temp->next;
         }
      }
printf("Shortest path length is %d\n", dist[endVertex]);
printf("Path is : ");
   int crawl = endVertex;
   while(prev[crawl] != -1) {
printf("%d ", crawl);
      crawl = prev[crawl];
   }
printf("%d\n", startVertex);
}
int main() {
   Graph *graph = createGraph(5);
addEdge(graph, 0, 1);
addEdge(graph, 0, 2);
addEdge(graph, 1, 2);
addEdge(graph, 1, 3);
addEdge(graph, 2, 3);
addEdge(graph, 3, 4);
BFS(graph, 0, 4);
return ;
}
```

## 89. Write a program to find the longest path between two nodes in a graph.

```
#include <stdio.h>
#define N 10
int adj[N][N];
int visited[N];
int max_d, max_i;
void DFS(int i, int n, int d) {
   visited[i] = 1;
   if (d >max_d) max_d = d, max_i = i;
   for (int j = 0; j < n; ++j)
      if (adj[i][j] && !visited[j])
DFS(j, n, d + 1);
   visited[i] = 0;
}
void longest_path(int n) {
max_d = -1;
DFS(0, n, 0);
max_d = -1;
DFS(max_i, n, 0);
printf("Length of longest path is : %d\n", max_d);
}
int main() {
   int nodes, edges, x, y;
printf("Enter the number of nodes:\n");
scanf("%d", &nodes);
printf("Enter the number of edges:\n");
scanf("%d", &edges);
for(int i = 0; i< edges; i++) {
printf("Enter the edges (Source and Destination) :\n");
scanf("%d %d", &x, &y);
```

```
adj[x][y] = 1;
adj[y][x] = 1;
   }
longest_path(nodes);
return ;
}
```

## 90. Write a program to print the pattern of a right-angle triangle using asterisks.

```
#include <stdio.h>
void printPattern(int n) {
for(int i=0; i<n; i++) {
for(int j=0; j<=i; j++) {
printf("* ");
      }
printf("\n");
   }
}
int main() {
   int n;
printf("Enter the number of rows for the pattern:\n");
scanf("%d",&n);
printPattern(n);
return ;
}
```

## 91. Write a program to print the pattern of a diamond shape using asterisks.

```
#include <stdio.h>
int main() {
   int n, i, j, space = 1;
printf("Enter the number of rows: ");
scanf("%d",&n);
   space = n - 1;

   for (i = 1; i<= n; i++) {
      for (j = 1; j <= space; j++)
printf(" ");
      space--;

      for (j = 1; j <= 2 * i - 1; j++)
printf("*");
printf("\n");
   }

   space = 1;

   for (i = 1; i<= n - 1; i++) {
      for (j = 1; j <= space; j++)
printf(" ");
      space++;
      for (j = 1 ; j <= 2 * (n - i) - 1; j++)
printf("*");
printf("\n");
   }

return ;
}
```

## 92. Write a program to print the pattern of a hollow square using asterisks.

```
#include <stdio.h>
int main() {
   int n, i, j;
printf("Enter the size of the square: ");
scanf("%d",&n);
for(i=1; i<=n; i++) {
for(j=1; j<=n; j++) {
if(i==1 || i==n || j==1 || j==n)
printf("*");
          else
printf(" ");
       }
printf("\n");
   }
return ;
}
```

## 93. Write a program to print the pattern of a pyramid using asterisks.

```
#include <stdio.h>
int main() {
   int n, i, j, space;
printf("Enter the number of rows: ");
scanf("%d",&n);
   space = n - 1;
   for (i = 1; i<= n; i++) {
      for (j = 1; j <= space; j++)
printf(" ");
       space--;
       for (j = 1; j <= 2 * i - 1; j++)
printf("*");
printf("\n");
   }
return ;
}
```

**94. Write a program code in 'C' to print the Fibonacci series up to a given number.**

```
#include<stdio.h>
int main(){
   int num, a = 0, b = 1, c, i = 0;
printf("Enter the number of elements in Fibonacci series: ");
scanf("%d", &num);
while(i<num){
printf("%d ", a);
      c = a + b;
      a = b;
      b = c;
i++;
   }
return ;
}
```

**95. Write a program to generate and display the first N numbers in a series that is a mix of even and odd numbers (odd numbers up to N, followed by even numbers up to N).**

```
#include<stdio.h>
int main(){
   int N, i;
printf("Enter a number N: ");
scanf("%d", &N);
printf("Odd numbers upto %d are: ", N);
for(i = 1; i<= N; i += 2){
printf("%d ", i);
   }
printf("\n");
printf("Even numbers upto %d are: ", N);
for(i = 2; i<= N; i += 2){
printf("%d ", i);
   }
return ;
}
```

## 96. Write a program to generate the first N prime numbers.

```
#include<stdio.h>
int is_prime(int num){
   int i;
if(num<= 1) return 0;
for(i = 2; i * i<= num; i++){
if(num % i == 0) return 0;
   }
   return 1;
}
int main(){
   int N, i = 2, count = 0;
printf("Enter a number N: ");
scanf("%d", &N);
while(count < N){
      if(is_prime(i)){
printf("%d ", i);
         count++;
      }
i++;
   }
return ;
}
```

## 97. Write a program to create a simple ATM machine functionality with options like check balance, deposit money, withdraw money, and quit.

```
#include<stdio.h>
int balance = 1000;
void checkBalance() {
printf("Your current balance is: %d\n", balance);
}
void depositMoney() {
   int deposit;
printf("Enter the amount to deposit: ");
scanf("%d", &deposit);
   balance += deposit;
printf("Amount deposited successfully!\n");
}
void withdrawMoney() {
   int withdrawal;
printf("Enter the amount to withdraw: ");
scanf("%d", &withdrawal);
if(withdrawal > balance){
printf("Insufficient balance!\n");
   } else {
      balance -= withdrawal;
printf("Amount withdrawn successfully!\n");
   }
}
int main() {
   int option;
   do {
printf("\n*** ATM Service ***\n");
```

```
printf("1. Check Balance\n");
printf("2. Deposit Money\n");
printf("3. Withdraw Money\n");
printf("4. Quit\n");
printf("**********\n\n");
printf("Enter your choice: ");
scanf("%d", &option);
        switch (option) {
            case 1:
checkBalance();
break;
            case 2:
depositMoney();
break;
            case 3:
withdrawMoney();
break;
            case 4:
printf("Thank you for using our ATM service!\n");
break;
     default:
printf("Invalid option! Please enter a valid option.\n");
break;
        }
    } while (option != 4);
    return 0;
}
```

**98. Write a program to calculate the perimeter and area of different shapes (rectangle, square, circle, and triangle) using functions.**

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
void rectangle(float length, float breadth) {
printf("Area of Rectangle: %.2f\n", length * breadth);
printf("Perimeter of Rectangle: %.2f\n", 2 * (length + breadth));
}
void square(float side) {
printf("Area of Square: %.2f\n", side * side);
printf("Perimeter of Square: %.2f\n", 4 * side);
}
void circle(float radius) {
printf("Area of Circle: %.2f\n", PI * radius * radius);
printf("Perimeter of Circle: %.2f\n", 2 * PI * radius);
}
void triangle(float side1, float side2, float side3) {
   float s = (side1 + side2 + side3) / 2;
printf("Area of Triangle: %.2f\n", sqrt(s * (s - side1) * (s - side2) * (s - side3)));
printf("Perimeter of Triangle: %.2f\n", side1 + side2 + side3);
}
int main() {
rectangle(5, 6);
square(4);
circle(3);
triangle(3, 4, 5);
return ;
}
```

## 99. Write a program to solve a quadratic equation using the coefficients.

```
#include <stdio.h>
#include <math.h>
int main() {
   double a, b, c, discriminant, root1, root2;
printf("Enter coefficients a, b and c: ");
scanf("%lf %lf %lf", &a, &b, &c);
   discriminant = b*b - 4*a*c;
   if (discriminant > 0) {
      root1 = (-b + sqrt(discriminant)) / (2*a);
      root2 = (-b - sqrt(discriminant)) / (2*a);
printf("Roots are real and different.\n");
printf("Root 1 = %.2lf\n", root1);
printf("Root 2 = %.2lf\n", root2);
   }
   else if (discriminant == 0) {
      root1 = root2 = -b / (2*a);
printf("Roots are real and the same.\n");
printf("Root 1 = Root 2 = %.2lf\n", root1);
   }
   else {
      double realPart = -b/(2*a);
      double imaginaryPart = sqrt(-discriminant)/(2*a);
printf("Roots are complex and different.\n");
printf("Root 1 = %.2lf+%.2lfi\n", realPart, imaginaryPart);
printf("Root 2 = %.2lf-%.2lfi\n", realPart, imaginaryPart);
   }
return ;
}
```