Q1. Design, Develop and Implement a menu driven Program in C/C++ for the following Array operations

a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element at a given valid Position (POS)
d. Deleting an Element at a given valid Position (POS)
e. Exit.

Support the program with functions for each of the above operations.

ANS:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

void createArray(int arr[], int *size);
void displayArray(int arr[], int size);
void insertElement(int arr[], int *size, int element, int pos);
void deleteElement(int arr[], int *size, int pos);

int main () {
  int arr[MAX_SIZE];
  int size = 0;
  int choice, element, pos;

  while (1) {
    printf("\nMenu:\n");
    printf("1. Create Array\n");
    printf("2. Display Array\n");
    printf("3. Insert Element\n");
    printf("4. Delete Element\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        createArray(arr, &size);
        break;
      case 2:
        displayArray(arr, size);
        break;
      case 3:
        printf("Enter the element to insert: ");
        scanf("%d", &element);
        printf("Enter the position (0 to %d): ", size);
        scanf("%d", &pos);
        insertElement(arr, &size, element, pos);
        break;
      case 4:
        printf("Enter the position to delete (0 to %d): ", size - 1);
        scanf("%d", &pos);
        deleteElement(arr, &size, pos);
```

```
            break;
        case 5:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void createArray(int arr[], int *size) {
   printf("Enter the number of elements: ", MAX_SIZE);
   scanf("%d", size);
   if (*size < 1 || *size > MAX_SIZE) {
      printf("Invalid size! Please enter a number between 1 and %d.\n", MAX_SIZE);
      *size = 0; // Reset size if invalid
      return;
   }
   printf("Enter %d elements:\n", *size);
   for (int i = 0; i < *size; i++) {
      scanf("%d", &arr[i]);
   }
}

void displayArray(int arr[], int size) {
   if (size == 0) {
      printf("Array is empty.\n");
      return;
   }
   printf("Array elements are:\n");
   for (int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

void insertElement(int arr[], int *size, int element, int pos) {
   if (pos < 0 || pos > *size || *size >= MAX_SIZE) {
      printf("Invalid position for insertion!\n");
      return;
   }
   for (int i = *size; i > pos; i--) {
      arr[i] = arr[i - 1];
   }
   arr[pos] = element;
   (*size)++;
   printf("Element %d inserted at position %d.\n", element, pos);
}

void deleteElement(int arr[], int *size, int pos) {
   if (pos < 0 || pos >= *size) {
```

```
    printf("Invalid position for deletion!\n");
    return;
  }
  for (int i = pos; i < *size - 1; i++) {
    arr[i] = arr[i + 1];
  }
  (*size)--;
  printf("Element at position %d deleted.\n", pos);
}
```

****

Q2. Design, Develop and Implement the following menu driven Programs in C/C++ using Array operations

a. Write a program for Bubble Sort algorithm
b. Write a program for Merge Sort algorithm
c. Write a program for Radix Sort algorithm
d. Write a program for Insertion Sort algorithm
e. Write a program for Selection Sort algorithm

ANS:

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

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

void merge(int arr[], int left, int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - mid;
  int L[n1], R[n2];
  for (int i = 0; i < n1; i++) L[i] = arr[left + i];
  for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

  int i = 0, j = 0, k = left;
  while (i < n1 && j < n2) arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
  while (i < n1) arr[k++] = L[i++];
  while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
  }
}
```

```
int getMax(int arr[], int n) {
  int max = arr[0];
  for (int i = 1; i < n; i++) if (arr[i] > max) max = arr[i];
  return max;
}

void countingSort(int arr[], int n, int exp) {
  int output[n], count[10] = {0};
  for (int i = 0; i < n; i++) count[(arr[i] / exp) % 10]++;
  for (int i = 1; i < 10; i++) count[i] += count[i - 1];
  for (int i = n - 1; i >= 0; i--) output[--count[(arr[i] / exp) % 10]] = arr[i];
  for (int i = 0; i < n; i++) arr[i] = output[i];
}

void radixSort(int arr[], int n) {
  int max = getMax(arr, n);
  for (int exp = 1; max / exp > 0; exp *= 10) countingSort(arr, n, exp);
}

void insertionSort(int arr[], int n) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > key) arr[j + 1] = arr[j--];
    arr[j + 1] = key;
  }
}

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

void printArray(int arr[], int n) {
  for (int i = 0; i < n; i++) printf("%d ", arr[i]);
  printf("\n");
}

int main() {
  int choice, n;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter elements: ");
  for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

  do {
```

```
    printf("\nMenu:\n");
    printf("1. Bubble Sort\n2. Merge Sort\n3. Radix Sort\n4. Insertion Sort\n5. Selection Sort\n6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        bubbleSort(arr, n);
        printf("Sorted array (Bubble Sort): ");
        printArray(arr, n);
        break;
      case 2:
        mergeSort(arr, 0, n - 1);
        printf("Sorted array (Merge Sort): ");
        printArray(arr, n);
        break;
      case 3:
        radixSort(arr, n);
        printf("Sorted array (Radix Sort): ");
        printArray(arr, n);
        break;
      case 4:
        insertionSort(arr, n);
        printf("Sorted array (Insertion Sort): ");
        printArray(arr, n);
        break;
      case 5:
        selectionSort(arr, n);
        printf("Sorted array (Selection Sort): ");
        printArray(arr, n);
        break;
      case 6:
        printf("Exiting program.\n");
        break;
      default:
        printf("Invalid choice. Try again.\n");
    }
  } while (choice != 6);

  return 0;
}
```

****

Q3. Design, Develop and Implement the following menu driven Programs in C/C++ for implementing

a. Write a program for Heap Sort algorithm
b. Write a program for Quick Sort algorithm
c. Write a program for linear search algorithm
d. Write a program for displaying a sparse matrix
e. Fibonacci numbers
f. Factorial of a number

ANS:

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

// Heap Sort Functions
void heapify(int arr[], int n, int i) {
  int largest = i, left = 2 * i + 1, 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) {
    int temp = arr[i];
    arr[i] = arr[largest];
    arr[largest] = temp;
    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--) {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    heapify(arr, i, 0);
  }
}

// Quick Sort Functions
void quickSort(int arr[], int low, int high) {
  if (low < high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++) {
      if (arr[j] < pivot) {
        int temp = arr[++i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    int pi = i + 1;
    quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
  }
}

// Linear Search
int linearSearch(int arr[], int n, int key) {
  for (int i = 0; i < n; i++) if (arr[i] == key) return i;
  return -1;
}
```

```
// Sparse Matrix Display
void displaySparseMatrix(int rows, int cols, int sparse[10][10]) {
   printf("Sparse Matrix Representation:\nRow\tColumn\tValue\n");
   for (int i = 0; i < rows; i++)
      for (int j = 0; j < cols; j++)
         if (sparse[i][j] != 0)
            printf("%d\t%d\t%d\n", i, j, sparse[i][j]);
}

// Fibonacci Sequence
int fibonacci(int n) {
   if (n <= 1) return n;
   return fibonacci(n - 1) + fibonacci(n - 2);
}

// Factorial Function
int factorial(int n) {
   if (n <= 1) return 1;
   return n * factorial(n - 1);
}

// Print Array
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++) printf("%d ", arr[i]);
   printf("\n");
}

int main() {
   int choice, n, key, index;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter elements: ");
   for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

   int sparse[10][10] = { {0, 0, 0, 5}, {0, 1, 0, 0}, {2, 0, 9, 0}, {0, 3, 0, 0} };
   int sparseRows = 4, sparseCols = 4;

   do {
      printf("\nMenu:\n");
      printf("1. Heap Sort\n2. Quick Sort\n3. Linear Search\n4. Display Sparse Matrix\n");
      printf("5. Fibonacci Number\n6. Factorial\n7. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            heapSort(arr, n);
            printf("Sorted array (Heap Sort): ");
            printArray(arr, n);
            break;
         case 2:
            quickSort(arr, 0, n - 1);
```

```
            printf("Sorted array (Quick Sort): ");
            printArray(arr, n);
            break;
        case 3:
            printf("Enter element to search: ");
            scanf("%d", &key);
            index = linearSearch(arr, n, key);
            if (index != -1)
                printf("Element found at index %d\n", index);
            else
                printf("Element not found\n");
            break;
        case 4:
            displaySparseMatrix(sparseRows, sparseCols, sparse);
            break;
        case 5:
            printf("Enter term position for Fibonacci: ");
            scanf("%d", &key);
            printf("Fibonacci(%d) = %d\n", key, fibonacci(key));
            break;
        case 6:
            printf("Enter number for factorial: ");
            scanf("%d", &key);
            printf("Factorial(%d) = %d\n", key, factorial(key));
            break;
        case 7:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Try again.\n");
    }
  } while (choice != 7);

  return 0;
}
```

****

Q4. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to ch eck Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit

Support the program with appropriate functions for each of the above operations

ANS:

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

```
#define MAX 100

int stack[MAX];
int top = -1;

// Function to check if the stack is empty
int isEmpty() {
  return top == -1;
}

// Function to check if the stack is full
int isFull() {
  return top == MAX - 1;
}

// Function to push an element onto the stack
void push(int element) {
  if (isFull()) {
    printf("Stack Overflow! Cannot push %d\n", element);
  } else {
    stack[++top] = element;
    printf("Pushed %d onto the stack\n", element);
  }
}

// Function to pop an element from the stack
int pop() {
  if (isEmpty()) {
    printf("Stack Underflow! Cannot pop an element\n");
    return -1;
  } else {
    printf("Popped %d from the stack\n", stack[top]);
    return stack[top--];
  }
}

// Function to check if a given string is a palindrome using stack
void checkPalindrome(char str[]) {
  int length = strlen(str);

  // Push all characters of the string onto the stack
  for (int i = 0; i < length; i++) {
    push(str[i]);
  }

  // Check if popping characters gives the original string
  int isPalindrome = 1;
  for (int i = 0; i < length; i++) {
    if (pop() != str[i]) {
      isPalindrome = 0;
      break;
    }
  }
```

```
    if (isPalindrome) {
        printf("The string \"%s\" is a palindrome.\n", str);
    } else {
        printf("The string \"%s\" is not a palindrome.\n", str);
    }

    // Reset top for further operations
    top = -1;
}

// Function to display the stack status
void displayStack() {
    if (isEmpty()) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack status: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, element;
    char str[100];

    do {
        printf("\nMenu:\n");
        printf("1. Push an Element onto Stack\n");
        printf("2. Pop an Element from Stack\n");
        printf("3. Check Palindrome using Stack\n");
        printf("4. Demonstrate Overflow and Underflow situations\n");
        printf("5. Display Stack Status\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to push: ");
                scanf("%d", &element);
                push(element);
                break;
            case 2:
                pop();
                break;
            case 3:
                printf("Enter string to check palindrome: ");
                scanf("%s", str);
                checkPalindrome(str);
                break;
```

```
        case 4:
          printf("Demonstrating Overflow:\n");
          for (int i = 0; i < MAX + 1; i++) {
            push(i);
          }
          printf("\nDemonstrating Underflow:\n");
          for (int i = 0; i < MAX + 1; i++) {
            pop();
          }
          break;
        case 5:
          displayStack();
          break;
        case 6:
          printf("Exiting program.\n");
          break;
        default:
          printf("Invalid choice. Try again.\n");
      }
    } while (choice != 6);

    return 0;
}
```

****

Q5. Design, Develop and Implement a Program in C/C++ for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

ANS:

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

#define MAX 100

// Stack structure
typedef struct Stack {
    char arr[MAX];
    int top;
} Stack;

// Function prototypes
void initStack(Stack *s);
int isFull(Stack *s);
int isEmpty(Stack *s);
void push(Stack *s, char c);
char pop(Stack *s);
char peek(Stack *s);
int precedence(char op);
int isOperator(char c);
```

```
void infixToPostfix(char *infix, char *postfix);

int main() {
   char infix[MAX], postfix[MAX];

   printf("Enter an infix expression: ");
   fgets(infix, sizeof(infix), stdin);
   infix[strcspn(infix, "\n")] = '\0'; // Remove newline character

   infixToPostfix(infix, postfix);
   printf("Postfix expression: %s\n", postfix);

   return 0;
}

void initStack(Stack *s) {
   s->top = -1;
}

int isFull(Stack *s) {
   return s->top == MAX - 1;
}

int isEmpty(Stack *s) {
   return s->top == -1;
}

void push(Stack *s, char c) {
   if (!isFull(s)) {
      s->arr[++s->top] = c;
   }
}

char pop(Stack *s) {
   if (!isEmpty(s)) {
      return s->arr[s->top--];
   }
   return '\0'; // Return null character if stack is empty
}

char peek(Stack *s) {
   if (!isEmpty(s)) {
      return s->arr[s->top];
   }
   return '\0'; // Return null character if stack is empty
}

int precedence(char op) {
   switch (op) {
      case '+':
      case '-':
         return 1;
      case '*':
```

```
    case '/':
    case '%':
      return 2;
    case '^':
      return 3;
    default:
      return 0;
  }
}

int isOperator(char c) {
  return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%' || c == '^');
}

void infixToPostfix(char *infix, char *postfix) {
  Stack s;
  initStack(&s);
  int i = 0, j = 0;

  for (i = 0; infix[i] != '\0'; i++) {
    char c = infix[i];

    // If the character is an operand, add it to output
    if (isalnum(c)) {
      postfix[j++] = c;
    }
    // If the character is '(', push it to stack
    else if (c == '(') {
      push(&s, c);
    }
    // If the character is ')', pop and output from the stack
    // until an '(' is encountered
    else if (c == ')') {
      while (!isEmpty(&s) && peek(&s) != '(') {
        postfix[j++] = pop(&s);
      }
      pop(&s); // Remove '(' from stack
    }
    // An operator is encountered
    else if (isOperator(c)) {
      while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(c)) {
        postfix[j++] = pop(&s);
      }
      push(&s, c);
    }
  }
  // Pop all the operators from the stack
  while (!isEmpty(&s)) {
    postfix[j++] = pop(&s);
  }
  postfix[j] = '\0'; // Null terminate the postfix string
}
```

****

Q6. Design, Develop and Implement a Program in C/C++ for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks

ANS:

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

#define MAX 100

typedef struct Stack {
  int arr[MAX];
  int top;
} Stack;

void initStack(Stack *s);
int isFull(Stack *s);
int isEmpty(Stack *s);
void push(Stack *s, int value);
int pop(Stack *s);
int evaluatePostfix(char *postfix);
void towerOfHanoi(int n, char from, char to, char aux);

int main() {
  int choice;
  char postfix[MAX];

  while (1) {
    printf("\nMenu:\n");
    printf("1. Evaluate Suffix Expression (Postfix)\n");
    printf("2. Solve Tower of Hanoi\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter a postfix expression (single-digit operands): ");
        scanf("%s", postfix);
        printf("Result: %d\n", evaluatePostfix(postfix));
        break;
      case 2:
        {
          int n;
          printf("Enter the number of disks: ");
          scanf("%d", &n);
          towerOfHanoi(n, 'A', 'C', 'B');
        }
        break;
      case 3:
```

```
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void initStack(Stack *s) {
   s->top = -1;
}

int isFull(Stack *s) {
   return s->top == MAX - 1;
}

int isEmpty(Stack *s) {
   return s->top == -1;
}

void push(Stack *s, int value) {
   if (!isFull(s)) {
      s->arr[++s->top] = value;
   }
}

int pop(Stack *s) {
   if (!isEmpty(s)) {
      return s->arr[s->top--];
   }
   return 0;
}

int evaluatePostfix(char *postfix) {
   Stack s;
   initStack(&s);
   int i = 0;

   while (postfix[i] != '\0') {
      if (isdigit(postfix[i])) {
         push(&s, postfix[i] - '0'); // Convert char to int
      } else {
         int b = pop(&s);
         int a = pop(&s);
         switch (postfix[i]) {
            case '+': push(&s, a + b); break;
            case '-': push(&s, a - b); break;
            case '*': push(&s, a * b); break;
            case '/': push(&s, a / b); break;
            case '%': push(&s, a % b); break;
            case '^': push(&s, (int)pow(a, b)); break;
```

```
        }
      }
      i++;
    }
    return pop(&s);
}

void towerOfHanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        printf("Move disk 1 from rod %c to rod %c\n", from, to);
        return;
    }
    towerOfHanoi(n - 1, from, aux, to);
    printf("Move disk %d from rod %c to rod %c\n", n, from, to);
    towerOfHanoi(n - 1, aux, to, from);
}
```

****

Q7. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on Circular QUEUE of Characters

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

Support the program with appropriate functions for each of the above operations

ANS:

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

#define MAX 100

typedef struct CircularQueue {
    char arr[MAX];
    int front;
    int rear;
} CircularQueue;

void initQueue(CircularQueue *q);
int isFull(CircularQueue *q);
int isEmpty(CircularQueue *q);
void enqueue(CircularQueue *q, char c);
char dequeue(CircularQueue *q);
void displayQueue(CircularQueue *q);
void demonstrateOverflow(CircularQueue *q);
void demonstrateUnderflow(CircularQueue *q);

int main() {
    CircularQueue q;
    initQueue(&q);
    int choice;
```

```
    char c;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert an Element onto Circular Queue\n");
        printf("2. Delete an Element from Circular Queue\n");
        printf("3. Demonstrate Overflow\n");
        printf("4. Demonstrate Underflow\n");
        printf("5. Display Circular Queue Status\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a character to insert: ");
                scanf(" %c", &c);
                enqueue(&q, c);
                break;
            case 2:
                c = dequeue(&q);
                if (c != '\0') {
                    printf("Deleted element: %c\n", c);
                }
                break;
            case 3:
                demonstrateOverflow(&q);
                break;
            case 4:
                demonstrateUnderflow(&q);
                break;
            case 5:
                displayQueue(&q);
                break;
            case 6:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

void initQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(CircularQueue *q) {
    return (q->front == (q->rear + 1) % MAX);
}
```

```
int isEmpty(CircularQueue *q) {
    return (q->front == -1);
}

void enqueue(CircularQueue *q, char c) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot insert '%c'.\n", c);
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->arr[q->rear] = c;
    printf("Inserted '%c' into the queue.\n", c);
}

char dequeue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot delete from the queue.\n");
        return '\0';
    }
    char c = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX;
    }
    return c;
}

void displayQueue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Circular Queue is empty.\n");
        return;
    }
    printf("Circular Queue elements: ");
    int i = q->front;
    while (1) {
        printf("%c ", q->arr[i]);
        if (i == q->rear) {
            break;
        }
        i = (i + 1) % MAX;
    }
    printf("\n");
}

void demonstrateOverflow(CircularQueue *q) {
    printf("Demonstrating Overflow by inserting 101 characters:\n");
    for (char c = 'A'; c <= 'Z'; c++) {
```

```
    enqueue(q, c);
  }
  for (char c = 'A'; c <= 'Z'; c++) {
    enqueue(q, c);
  }
  enqueue(q, 'O');
}

void demonstrateUnderflow(CircularQueue *q) {
  printf("Demonstrating Underflow by deleting from an empty queue:\n");
  for (int i = 0; i < 5; i++) {
    dequeue(q);
  }
}
```

****

Q8. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on Singly Linked List (SLL)

a. Create a SLL.
b. Insert at Beginning
c. Insert at Last
d. Insert at any random location
e. Delete from Beginning
f. Delete from Last
g. Delete node after specified location
h. Search for an element
i. Show
j. Exit

ANS:

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

// Define the structure of a linked list node
struct Node {
  int data;
  struct Node* next;
};

// Function prototypes
void createSLL();
void insertAtBeginning(int data);
void insertAtLast(int data);
void insertAtPosition(int data, int position);
void deleteFromBeginning();
void deleteFromLast();
void deleteAfterPosition(int position);
void search(int key);
void display();

struct Node* head = NULL;
```

```
// Function to create a singly linked list
void createSLL() {
   int n, data;
   printf("Enter the number of nodes: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &data);
      insertAtLast(data);
   }
}

// Function to insert a node at the beginning
void insertAtBeginning(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

// Function to insert a node at the end
void insertAtLast(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   } else {
      struct Node* temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
}

// Function to insert a node at a specific position
void insertAtPosition(int data, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;

   if (position == 1) {
      newNode->next = head;
      head = newNode;
   } else {
      struct Node* temp = head;
      for (int i = 1; i < position - 1 && temp != NULL; i++) {
         temp = temp->next;
      }
      if (temp != NULL) {
         newNode->next = temp->next;
         temp->next = newNode;
```

```
    } else {
      printf("Position out of range\n");
    }
  }
}

// Function to delete a node from the beginning
void deleteFromBeginning() {
  if (head == NULL) {
    printf("List is empty\n");
  } else {
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("Node deleted from the beginning\n");
  }
}

// Function to delete a node from the end
void deleteFromLast() {
  if (head == NULL) {
    printf("List is empty\n");
  } else if (head->next == NULL) {
    free(head);
    head = NULL;
    printf("Node deleted from the last\n");
  } else {
    struct Node* temp = head;
    while (temp->next->next != NULL) {
      temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Node deleted from the last\n");
  }
}

// Function to delete a node after a specific position
void deleteAfterPosition(int position) {
  if (head == NULL) {
    printf("List is empty\n");
  } else {
    struct Node* temp = head;
    for (int i = 1; i < position && temp != NULL; i++) {
      temp = temp->next;
    }
    if (temp != NULL && temp->next != NULL) {
      struct Node* nodeToDelete = temp->next;
      temp->next = temp->next->next;
      free(nodeToDelete);
      printf("Node deleted after position %d\n", position);
    } else {
      printf("Position out of range\n");
```

```
    }
  }
}

// Function to search for an element in the linked list
void search(int key) {
  struct Node* temp = head;
  int position = 1;
  while (temp != NULL) {
    if (temp->data == key) {
      printf("Element %d found at position %d\n", key, position);
      return;
    }
    temp = temp->next;
    position++;
  }
  printf("Element %d not found in the list\n", key);
}

// Function to display the linked list
void display() {
  if (head == NULL) {
    printf("List is empty\n");
  } else {
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
    printf("NULL\n");
  }
}

int main() {
  int choice, data, position;

  do {
    printf("\nMenu:\n");
    printf("1. Create a SLL\n");
    printf("2. Insert at Beginning\n");
    printf("3. Insert at Last\n");
    printf("4. Insert at Any Position\n");
    printf("5. Delete from Beginning\n");
    printf("6. Delete from Last\n");
    printf("7. Delete Node after Specified Position\n");
    printf("8. Search for an Element\n");
    printf("9. Show\n");
    printf("10. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
```

```
        case 1:
          createSLL();
          break;
        case 2:
          printf("Enter data to insert at beginning: ");
          scanf("%d", &data);
          insertAtBeginning(data);
          break;
        case 3:
          printf("Enter data to insert at last: ");
          scanf("%d", &data);
          insertAtLast(data);
          break;
        case 4:
          printf("Enter data to insert: ");
          scanf("%d", &data);
          printf("Enter position to insert at: ");
          scanf("%d", &position);
          insertAtPosition(data, position);
          break;
        case 5:
          deleteFromBeginning();
          break;
        case 6:
          deleteFromLast();
          break;
        case 7:
          printf("Enter position after which to delete: ");
          scanf("%d", &position);
          deleteAfterPosition(position);
          break;
        case 8:
          printf("Enter element to search for: ");
          scanf("%d", &data);
          search(data);
          break;
        case 9:
          display();
          break;
        case 10:
          printf("Exiting program.\n");
          break;
        default:
          printf("Invalid choice. Try again.\n");
      }
    } while (choice != 10);

    return 0;
}
```

****

Q9. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on Doubly Linked List (DLL)

a. Create a DLL.
b. Print all the elements in DLL in forward traversal order
c. Print all elements in DLL in reverse traversal order
d. Exit

ANS:

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

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

Node* createDLL();
void printForward(Node* head);
void printReverse(Node* tail);
void freeList(Node* head);

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

  while (1) {
    printf("\nMenu:\n");
    printf("1. Create a Doubly Linked List\n");
    printf("2. Print elements in forward order\n");
    printf("3. Print elements in reverse order\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        head = createDLL();
        break;
      case 2:
        printForward(head);
        break;
      case 3:
        {
          Node* tail = head;
          while (tail && tail->next) {
            tail = tail->next;
          }
          printReverse(tail);
        }
        break;
      case 4:
```

```
            freeList(head);
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
    }

    return 0;
}

Node* createDLL() {
    Node* head = NULL;
    Node* tail = NULL;
    int n, data;

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

    for (int i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        Node* newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = NULL;
        newNode->prev = tail;

        if (tail) {
            tail->next = newNode;
        } else {
            head = newNode;
        }
        tail = newNode;
    }

    return head;
}

void printForward(Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    printf("Doubly Linked List in forward order: ");
    Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```
void printReverse(Node* tail) {
  if (tail == NULL) {
    printf("The list is empty.\n");
    return;
  }

  printf("Doubly Linked List in reverse order: ");
  Node* current = tail;
  while (current != NULL) {
    printf("%d ", current->data);
    current = current->prev;
  }
  printf("\n");
}

void freeList(Node* head) {
  Node* current = head;
  while (current != NULL) {
    Node* nextNode = current->next;
    free(current);
    current = nextNode;
  }
}
```

****

Q10. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on Binary Search Tree (BST) of Integers

a. Create a BST of N Integers: 8, 10, 3, 1, 6, 14, 7
b. Traverse the BST in Inorder
c. Traverse the BST in Preorder
d. Traverse the BST in and Post Order

ANS:

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

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

Node* createNode(int data);
Node* insert(Node* root, int data);
void inorderTraversal(Node* root);
void preorderTraversal(Node* root);
void postorderTraversal(Node* root);
void freeTree(Node* root);

int main() {
  Node* root = NULL;
  int choice;
```

```
    int initialValues[] = {8, 10, 3, 1, 6, 14, 7};
    int n = sizeof(initialValues) / sizeof(initialValues[0]);
    for (int i = 0; i < n; i++) {
        root = insert(root, initialValues[i]);
    }

    while (1) {
        printf("\nMenu:\n");
        printf("1. Traverse the BST in Inorder\n");
        printf("2. Traverse the BST in Preorder\n");
        printf("3. Traverse the BST in Postorder\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 2:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 4:
                freeTree(root);
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

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

Node* insert(Node* root, int data) {
    if (root == NULL) {
```

```
    return createNode(data);
  }

  if (data < root->data) {
    root->left = insert(root->left, data);
  } else if (data > root->data) {
    root->right = insert(root->right, data);
  }
  return root;
}

void inorderTraversal(Node* root) {
  if (root != NULL) {
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
  }
}
void preorderTraversal(Node* root) {
  if (root != NULL) {
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
  }
}
void postorderTraversal(Node* root) {
  if (root != NULL) {
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
  }
}
void freeTree(Node* root) {
  if (root != NULL) {
    freeTree(root->left);
    freeTree(root->right);
    free(root);
  }
}
```

****

Q11. Design, Develop and Implement a Program in C/C++ for the following operations on Graph(G)

a. Create a Graph N using Adjacency Matrix.

b. Print all the nodes reachable from a given starting node in a digraph using BFS method

c. Print all the nodes reachable from a given starting node in a digraph using DFS method.

ANS:

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

#define MAX 100
```

```
// Function prototypes
void createGraph(int graph[MAX][MAX], int n);
void bfs(int graph[MAX][MAX], int n, int start);
void dfs(int graph[MAX][MAX], int n, int start);
void dfsUtil(int graph[MAX][MAX], int n, int start, int visited[]);

int main() {
  int graph[MAX][MAX], n, start, choice;

  for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++) {
      graph[i][j] = 0;
    }
  }

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

  while (1) {
    printf("\nMenu:\n");
    printf("1. Print nodes reachable using BFS\n");
    printf("2. Print nodes reachable using DFS\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter starting node (0 to %d): ", n-1);
        scanf("%d", &start);
        bfs(graph, n, start);
        break;
      case 2:
        printf("Enter starting node (0 to %d): ", n-1);
        scanf("%d", &start);
        dfs(graph, n, start);
        break;
      case 3:
        printf("Exiting the program.\n");
        exit(0);
      default:
        printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}

void createGraph(int graph[MAX][MAX], int n) {
  int edges, u, v;
  printf("Enter number of edges: ");
```

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
    }
}

void bfs(int graph[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};
    int queue[MAX], front = -1, rear = -1;

    visited[start] = 1;
    queue[++rear] = start;

    printf("BFS starting from node %d: ", start);

    while (front < rear) {
        front++;
        int current = queue[front];
        printf("%d ", current);

        for (int i = 0; i < n; i++) {
            if (graph[current][i] && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
    printf("\n");
}

void dfs(int graph[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};
    printf("DFS starting from node %d: ", start);
    dfsUtil(graph, n, start, visited);
    printf("\n");
}

void dfsUtil(int graph[MAX][MAX], int n, int start, int visited[]) {
    visited[start] = 1;
    printf("%d ", start);

    for (int i = 0; i < n; i++) {
        if (graph[start][i] && !visited[i]) {
            dfsUtil(graph, n, i, visited);
        }
    }
}
```

****