CODE

1) Write a program that finds the prime numbers using the sieve method.

```
#include <stdio.h>
void seive_prime(int n){
   int prime[100] = {0};
   for (int i = 2; i <= n; i++){
      if (prime[i] == 0){
         for (int j = i * i; j <= n; j = j + i){
            prime[j] = 1;
         }
      }
   }
   for (int i = 1; i <= n; i++){
      if (prime[i] == 0)
         printf("%d ", i);
   }
}
int main()
{
   int n;
   scanf("%d", &n);
   seive_prime(n);
}
```

2) Write a program that finds the largest and smallest elements in an array.

```
#include <stdio.h>
int largest_element(int array[], int size){
   int mx = array[0];
   for (int i = 0; i < size; i++){
      if (array[i] > mx)
         mx = array[i];
   }
   return mx;
}
int smallest_element(int array[], int size){
   int mn = array[0];
   for (int i = 0; i < size; i++){
      if (array[i] < mn)
         mn = array[i];
   }
   return mn;
}
int main()
{
   int n;
```

```
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    int largest = largest_element(array, n);
    int smallest = smallest_element(array, n);
    printf("Largest element is: %d\nSmallest element is:%d\n", largest, smallest);
}
```

3) An array A Containing N elements is given. Write a program that captures the sum of array elements.

```
#include <stdio.h>
int find_sum(int array[], int size){
    int sum = 0;
    for (int i = 0; i < size; i++)
        sum += array[i];
    return sum;
}

int main()
{
    int n;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    int sum = find_sum(array, n);
    printf("Sum of this array: %d\n", sum);
}
```

4) Write a program to find whether the array of integers contains a duplicate number.

```
#include <stdio.h>
#include <stdbool.h>
bool duplicate(int array[], int size){
    for (int i = 0; i < size; i++){
        for (int j = i + 1; j < size; j++){
            if (array[i] == array[j])
                return true;
        }
    }
    return false;
}
int main()
{
    int n;
    scanf("%d", &n);
    int array[n];
```

```
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    if (duplicate(array, n))
        printf("Find Duplicate\n");
    else
        printf("No Duplicate found\n");
}
```

5) Write a program to insert a number at a given location in an array.

```
#include <stdio.h>
void insert_number(int array[], int size, int pos, int val){
    for (int i = size; i >= pos - 1; i--)
        array[i + 1] = array[i];
    array[pos - 1] = val;
    for (int i = 0; i < size; i++)
        printf("%d ", array[i]);
}

int main()
{
    int n;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    int pos, val;
    scanf("%d%d", &pos, &val);
    insert_number(array, n + 1, pos, val);
}
```

6) Write a program to delete a number from a given location in an array.

```
#include <stdio.h>
int main()
{
    int n,pos;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    scanf("%d", &pos);
    for (int i = pos - 1; i < n; i++)
        array[i] = array[i + 1];

    for (int i = 0; i < n - 1; i++)
        printf("%d ", array[i]);
}
```

7) Write a program to merge two sorted arrays.

```
#include <stdio.h>

void marge_array(int array_1[], int array_2[], int n, int m){
   int a = n + m;
   int marge[a];
   for (int i = 0; i < n; i++)
      marge[i] = array_1[i];
   for (int i = 0; i < m; i++)
      marge[i + n] = array_2[i];
   for (int i = 0; i < a; i++)
      printf("%d ", marge[i]);
}

int main(){
   int n, m;
   scanf("%d", &n);
   int array_1[n];
   for (int i = 0; i < n; i++)
      scanf("%d", &array_1[i]);
   scanf("%d", &m);
   int array_2[m];
   for (int i = 0; i < m; i++)
      scanf("%d", &array_2[i]);
   marge_array(array_1, array_2, n, m);
}
```

8)Write programs for implementing the following sorting methods to arrange a list of integers/strings in ascending/descending order:

a) Bubble Sort

```
#include <stdio.h>
void bubble_sort(int array[], int size){
   for (int i = 1; i < size; i++){
      for (int j = 0; j < size - 1; j++){
         if (array[j] > array[j + 1]){
            int temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
         }
      }
   }
   for (int i = 0; i < size; i++)
      printf("%d ", array[i]);
}

int main(){
```

```
    int n;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    bubble_sort(array, n);
}
```

b) Selection Sort

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

int main(){
    int n;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    selection_sort(array, n);
}
```

c) Insertion Sort

```
#include <stdio.h>
void insertion_sort(int array[], int n){
    for (int i = 1; i < n; i++){
        int key = array[i];
        int j = i - 1;
        while (array[j] > key && j >= 0){
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = key;
    }
    for (int i = 0; i < n; i++)
        printf("%d ", array[i]);
}
```

```
int main(){
   int n;
   scanf("%d", &n);
   int array[n];
   for (int i = 0; i < n; i++)
      scanf("%d", &array[i]);
   insertion_sort(array, n);
}
```

9) Write programs for search an element from a list of integers/strings

a) Liner Search

```
#include <stdio.h>
#include <stdbool.h>
bool linear_search(int array[], int n, int val){
   for (int i = 0; i < n; i++){
      if (array[i] == val)
         return true;
   }
   return false;
}

int main(){
   int n;
   scanf("%d", &n);
   int array[n];
   for (int i = 0; i < n; i++)
      scanf("%d", &array[i]);
   int val;
   scanf("%d", &val);
   if (linear_search(array, n, val))
      printf("Value found\n");
   else
      printf("Value not found\n");
}
```

b) Binary Search

```
#include <stdio.h>
#include <stdbool.h>
bool binary_search(int array[], int n, int val){
   int l_bound, up_bound, mid;
   l_bound = 0;
   up_bound = n - 1;
   while (l_bound <= up_bound){
      mid = (l_bound + up_bound) / 2;
      if (array[mid] == val)
         return true;
      else if (array[mid] > val)
```

```
            up_bound = mid - 1;
        else if (array[mid] < val)
            l_bound = mid + 1;
    }
    return false;
}

int main(){
    int n;
    scanf("%d", &n);
    int array[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);
    int val;
    scanf("%d", &val);
    if (binary_search(array, n, val))
        printf("Value found\n");
    else
        printf("Value not found\n");
}
```

10) Write a program to read and display a matrix.

```
#include <stdio.h>
int main(){
    int n, m;
    scanf("%d%d", &n, &m);
    int matrix[n][m];
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++)
            scanf("%d", &matrix[i][j]);
    }
    printf("\n\n");
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
}
```

11) Write a program to add and multiply two matrices.

```
#include <stdio.h>
int matrix[100][100];
int matrix_1[100][100];
int matrix_2[100][100];

void set_matrix(int n, int m){
    printf("Enter 1st matrix: \n");
    for (int i = 0; i < n; i++){
```

```
      for (int j = 0; j < m; j++)
         scanf("%d", &matrix_1[i][j]);
   }
   printf("Enter 2nd matrix: \n");
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         scanf("%d", &matrix_2[i][j]);
   }
}

void add_matrix(int n, int m){
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         matrix[i][j] = matrix_1[i][j] + matrix_2[i][j];
   }
   printf("The sum of two martix:\n");
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         printf("%d ", matrix[i][j]);
      printf("\n");
   }
}

void mul_matrix(int n, int m){
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++){
         matrix[i][j] = 0;
         for (int k = 0; k < n; k++)
            matrix[i][j] = matrix[i][j] + matrix_1[i][j] * matrix_2[i][j];
      }
   }
   printf("Multipation of two martix:\n");
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         printf("%d ", matrix[i][j]);
      printf("\n");
   }
}

int main()
{
   int n, m;
   printf("Enter row and column: ");
   scanf("%d%d", &n, &m);
   set_matrix(n, m);
   add_matrix(n, m);
   mul_matrix(n, m);
}
```

12) Write a program that takes a matrix A and finds its transpose AT and displays it.

```
#include <stdio.h>

int main(){
   int n, m;
   scanf("%d%d", &n, &m);
   int matrix[n][m];
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         scanf("%d", &matrix[i][j]);
   }
   printf("Transpose Matrix: \n");
   for (int i = 0; i < m; i++){
      for (int j = 0; j < n; j++)
         printf("%d ", matrix[j][i]);
      printf("\n");
   }
}
```

13) Write a program that computes the sum of diagonal elements of a square matrix.

```
#include <stdio.h>

int sum_of_diagonal(int n, int m){
   int matrix[n][m];
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
         scanf("%d", &matrix[i][j]);
   }
   int sum = 0;
   for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++){
         if (i == j)
            sum += matrix[i][j];
      }
   }
   return sum;
}

int main(){
   int n, m;
   scanf("%d%d", &n, &m);
   int sum = sum_of_diagonal(n, m);
   printf("Sum of Diagonal element: %d\n", sum);
}
```

14) Write a program to find the length of a string.

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

```
int length(char string[]){
   int size = 0;
   for (int i = 0; string[i] != '\0'; i++)
      size++;
   return size;
}

int main(){
   char string[1000];
   gets(string);
   int size = length(string);
   printf("%d \n", size);
}
```

15) Write a program to concatenate two strings.

```
#include <stdio.h>
#include <string.h>
void concatenate(char st_1[], char st_2[]){
   char string[strlen(st_1) + strlen(st_2)];
   for (int i = 0; st_1[i] != '\0'; i++)
      string[i] = st_1[i];
   for (int i = 0; st_2[i] != '\0'; i++)
      string[i + strlen(st_1)] = st_2[i];
   string[strlen(st_1) + strlen(st_2)] = '\0';
   for (int i = 0; string[i] != '\0'; i++)
      printf("%c", string[i]);
}

int main(){
   char st_1[1000], st_2[1000];
   gets(st_1);
   gets(st_2);
   concatenate(st_1, st_2);
}
```

16) Write a program to compare two strings.

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool compare(char st_1[], char st_2[]){
   if (strlen(st_1) != strlen(st_2))
      return false;
   else{
      for (int i = 0; st_1[i] != '\0'; i++){
         if (st_1[i] != st_2[i])
            return false;
      }
   }
```

```
    return true;
}

int main(){
    char st_1[1000];
    char st_2[1000];
    gets(st_1);
    gets(st_2);
    if (compare(st_1, st_2))
        printf("Two string are same\n");
    else
        printf("String are not same\n");
}
```

17) Write a program to reverse a given string.

```
#include <stdio.h>
#include <string.h>
void reverse(char st[]){
    for (int i = strlen(st) - 1; i >= 0; i--)
        printf("%c", st[i]);
}

int main(){
    char st[1000];
    gets(st);
    reverse(st);
}
```

18) Write a program to extract a substring from a given string.

```
#include <stdio.h>
#include <string.h>
void extract_substring(char st[], int pos, int size){
    for (int i = pos - 1; i < (pos + size) - 1; i++)
        printf("%c", st[i]);
}

int main(){
    char st[1000];
    gets(st);
    int pos, size;
    scanf("%d%d", &pos, &size);
    extract_substring(st, pos, size);
}
```

19) Write a program to insert a string in the main text.

```
#include <stdio.h>
#include <string.h>
void insert_string(char st[], char insert[], int pos){
```

```
    char final[100];
    for (int i = 0; i < strlen(st); i++)
        final[i] = st[i];
    for (int i = strlen(st) + strlen(insert) - 1; i >= pos - 1; i--)
        final[i] = final[i - strlen(insert)];
    for (int i = 0; i < strlen(insert); i++)
        final[i + pos - 1] = insert[i];
    int size = strlen(st) + strlen(insert);
    final[size] = '\0';
    puts(final);
}

int main(){
    char st[100];
    char insert[100];
    gets(st);
    gets(insert);
    int pos;
    scanf("%d", &pos);
    insert_string(st, insert, pos);
}
```

20) Write a program to delete every occurrence of a pattern (character) from a text.

```
#include <stdio.h>
#include <string.h>
void delete_every_occurence(char st[], char ch){
    for (int i = 0; st[i] != '\0'; i++){
        if (st[i] != ch)
            printf("%c", st[i]);
    }
}

int main(){
    char st[1000];
    gets(st);
    char ch;
    scanf("%c", &ch);
    delete_every_occurence(st, ch);
}
```

21) Write a program to replace a pattern (character) with another pattern in the text.

```
#include <stdio.h>
#include <string.h>
void replace(char st[], char ch, char rep){
    for (int i = 0; st[i] != '\0'; i++){
        if (st[i] == ch)
            printf("%c", rep);
        else
```

```
            printf("%c", st[i]);
        }
    }

    int main(){
        char st[1000];
        gets(st);
        char ch, rep;
        scanf(" %c", &ch);
        scanf(" %c", &rep);
        replace(st, ch, rep);
    }
```

22) Write a program to develop the first pattern matching algorithm (Brute Force based).

```
    #include <stdio.h>
    #include <string.h>
    void first_pattern(char st[], char chk[]){
        for (int i = 0; i <= strlen(st) - strlen(chk); i++){
            int flag = 0;
            for (int j = 0; j < strlen(chk); j++){
                if (st[i + j] != chk[j]){
                    flag = 1;
                    break;
                }
            }
            if (flag == 0)
                printf("Index: %d\n", i + 1);
        }
    }

    int main(){
        char st[1000];
        char chk[1000];
        gets(st);
        gets(chk);
        first_pattern(st, chk);
    }
```

23) Write a program to develop the second pattern matching algorithm (Finite Automata based).

```
#include <stdio.h>
#include <string.h>
#define MAX_CHARS 256
void computeFailureFunction(const char *pattern, int M, int *failure){
    int len = 0; // Length of the previous longest prefix suffix
    failure[0] = 0; // The first character always matches with itself
    int i = 1;
```

```
    while (i < M){
        if (pattern[i] == pattern[len]){
            len++;
            failure[i] = len;
            i++;
        }
        else{
            if (len != 0)
                len = failure[len - 1];
            else{
                failure[i] = 0;
                i++;
            }
        }
    }
}

void finiteAutomatonSearch(const char *text, const char *pattern){
    int M = strlen(pattern);
    int N = strlen(text);
    int failure[M]; // Failure function (partial match table)
    computeFailureFunction(pattern, M, failure);
    int i = 0; // Index for text[]
    int j = 0; // Index for pattern[]
    while (i < N){
        if (pattern[j] == text[i]){
            i++;
            j++;
        }
        if (j == M){
            printf("Pattern found at index %d\n", i - j);
            j = failure[j - 1];
        }
        else if (i < N && pattern[j] != text[i]){
            if (j != 0)
                j = failure[j - 1];
            else
                i++;
        }
    }
}

int main(){
    const char *text = "ABABDABACDABABCABAB";
    const char *pattern = "ABABCABAB";
    printf("Text: %s\n", text);
    printf("Pattern: %s\n", pattern);
    printf("Matching positions:\n");
```

```
    finiteAutomatonSearch(text, pattern);
}
```

24) Write a program that uses functions to perform the following operations on singly linked list:
a) Creation
b) Insertion
c) Deletion
d) Traversal

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct list node;
struct list{
   int data;
   node *next;
};
node *creat_new_node(int val){
   node *new_node = (node *)malloc(sizeof(node));
   new_node->data = val;
   new_node->next = NULL;
}
node *insertion_at_head(node *head, int val){
   node *new_node = creat_new_node(val);
   if (head == NULL){
      head = new_node;
      return head;
   }
   new_node->next = head;
   head = new_node;
   return head;
}

node *insertion_at_tail(node *head, int val){
   node *new_node = creat_new_node(val);
   if (head == NULL){
      head = new_node;
      return head;
   }
   node *temp = head;
   while (temp->next != NULL)
      temp = temp->next;
   temp->next = new_node;
   return head;
}
node *deletion_at_head(node *head){
```

```
    if (head == NULL){
        printf("Underflow!!!\n");
        return head;
    }
    head = head->next;
    return head;
}

node *deletion_at_tail(node *head){
    if (head == NULL){
        printf("Underflow\n");
        return head;
    }
    node *temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;
    temp->next = NULL;
    return head;
}
void display(node *head){
    node *temp = head;
    while (temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

int main(){
    node *head = NULL;
    int n;
    scanf("%d", &n);
    while (n--){
        int a;
        scanf("%d", &a);
        head = insertion_at_tail(head, a);
    }
    display(head);
}
```

25) Write a program to create a circular linked list. Perform insertion and deletion at the beginning and end of the list.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct list node;
struct list{
    int data;
```

```
    node *next;
};

node *creat_new_node(int val){
    node *new_node = (node *)malloc(sizeof(node));
    new_node->data = val;
    new_node->next = NULL;
}

node *insertion_at_head(node *head, int val){
    node *new_node = creat_new_node(val);
    if (head == NULL){
        head = new_node;
        new_node->next = head;
        return head;
    }
    node *temp = head;
    while (temp->next != head)
        temp = temp->next;
    new_node->next = head;
    head = new_node;
    temp->next = head;
    return head;
}

node *insertion_at_tail(node *head, int val){
    node *new_node = creat_new_node(val);
    if (head == NULL){
        head = new_node;
        new_node->next = head;
        return head;
    }
    node *temp = head;
    while (temp->next != head)
        temp = temp->next;
    temp->next = new_node;
    new_node->next = head;
    return head;
}

node *deletiong_at_head(node *head){
    if (head == NULL){
        printf("Underflow!!!\n");
        return head;
```

```
    }
    node *temp = head;
    while (temp->next != head)
        temp = temp->next;
    head = head->next;
    temp->next = head;
    return head;
}

node *deletiong_at_tail(node *head){
    if (head == NULL){
        printf("Underflow!!!\n");
        return head;
    }
    node *temp = head;
    while (temp->next->next != head)
        temp = temp->next;
    temp->next = head;
    return head;
}

void display(node *head){
    node *temp = head;
    while (temp->next != head){
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("%d \n", temp->data);
}

int main(){
    node *head = NULL;
    int n;
    scanf("%d", &n);
    while (n--){
        int a;
        scanf("%d", &a);
        head = insertion_at_tail(head, a);
    }
    display(head);
}
```

26) Write programs that uses stack operations to convert a given infix expression into its postfix equivalent. Implement the stack using an array.

Header file:

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
typedef struct STACK stack;
int cnt = 0;

struct STACK{
   int top;
   int size;
   char *array;
};

bool is_empty(stack *s){
   if (s->top == -1)
      return true;
   return false;
}

bool is_full(stack *s){
   if (s->top == s->size - 1)
      return true;
   return false;
}

void PUSH(stack *s, char val){
   if (is_full(s)){
      printf("Stack is full!!!");
      return;
   }
   s->top++;
   cnt++;
   s->array[s->top] = val;
}

void POP(stack *s){
   if (is_empty(s)){
      printf("Stack is under flow!!!");
      return;
   }
   s->top--;
   cnt--;
}
```

```
int SIZE(){
   return cnt;
}

char TOP(stack *s){
   return s->array[s->top];
}

int precedence(char ch){
   if (ch == '*' || ch == '/')
      return 3;
   else if (ch == '+' || ch == '-')
      return 2;
   else
      return 0;
}

bool is_operator(char ch){
   if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
      return true;
   else
      return false;
}
```

Main Code:

```
#include <stdio.h>
#include "stack.h"
char *infix_to_postfix(char *infix){
   stack *s = (stack *)malloc(sizeof(stack));
   s->size = 1e3;
   s->top = -1;
   s->array = (char *)malloc(s->size * sizeof(char));
   char *postfix = (char *)malloc((strlen(infix) + 1) * sizeof(char));
   int i = 0, j = 0;
   while (infix[i] != '\0'){
      if (!is_operator(infix[i])){
         postfix[j] = infix[i];
         i++;
         j++;
      }
      else
      {
         if (precedence(infix[i]) > precedence(TOP(s))){
            PUSH(s, infix[i]);
```

```
                i++;
            }
            else{
                postfix[j] = TOP(s);
                POP(s);
                j++;
            }
        }
    }
    while (!is_empty(s)){
        postfix[j] = TOP(s);
        POP(s);
        j++;
    }
    postfix[j] = '\0';
    return postfix;
}

int main()
{
    char *infix = "a-b+t/6";
    printf("%s", infix_to_postfix(infix));
}
```

27) Write a Program in C to Implement

a) Stacks using arrays

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
int cnt = 0;
typedef struct STACK stack;
struct STACK{
    int top;
    int size;
    int *array;
};

bool empty(stack *s){
    if (s->top == -1)
        return true;
    return false;
}

bool is_full(stack *s){
```

```
    if (s->top == s->size - 1)
        return true;
    return false;
}

void PUSH(stack *s, int val){
    if (is_full(s)){
        printf("Stack is Overflow!!!\n");
        return;
    }
    s->top++;
    cnt++;
    s->array[s->top] = val;
}

void POP(stack *s){
    if (empty(s)){
        printf("Stack is Underflow!!!\n");
        return;
    }
    s->top--;
    cnt--;
}

int TOP(stack *s){
    return s->array[s->top];
}

int SIZE(){
    return cnt;
}

int main(){
    stack *s = (stack *)malloc(sizeof(stack));
    s->top = -1;
    s->size = 1e3;
    s->array = (int *)malloc(sizeof(int));
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++){
        int a;
        scanf("%d", &a);
        PUSH(s, a);
    }
```

```
    printf("%d\n", SIZE());
    while (!empty(s)){
        printf("%d ", TOP(s));
        POP(s);
    }
}
```

b) Stacks using linked list

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

struct Stack{
    struct Node *top;
};

void initialize(struct Stack *stack){
    stack->top = NULL;
}

int isEmpty(struct Stack *stack){
    return stack->top == NULL;
}

void push(struct Stack *stack, int item){
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL){
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = item;
    newNode->next = stack->top;
    stack->top = newNode;
}

int pop(struct Stack *stack){
    if (isEmpty(stack)){
        printf("Stack Underflow\n");
        exit(1);
    }
    struct Node *temp = stack->top;
```

```
    int item = temp->data;
    stack->top = temp->next;
    free(temp);
    return item;
}

int peek(struct Stack *stack)
{
    if (isEmpty(stack)){
        printf("Stack is empty\n");
        exit(1);
    }
    return stack->top->data;
}

int main(){
    struct Stack stack;
    initialize(&stack);
    push(&stack, 1);
    push(&stack, 2);
    push(&stack, 3);
    printf("Top of stack: %d\n", peek(&stack));
    while (!isEmpty(&stack))
        printf("Popped: %d\n", pop(&stack));
}
```

c) Queue using arrays

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
int cnt = 0;
typedef struct QUEUE queue;

struct QUEUE{
    int front;
    int rear;
    int size;
    int *array;
};

bool empty(queue *q){
    if (q->front == -1 && q->rear == -1)
        return true;
    return false;
```

```
}

bool is_full(queue *q){
   if (q->rear == q->size - 1)
      return true;
   return false;
}

void PUSH(queue *q, int val){
   if (is_full(q)){
      printf("Queue is Overflow!!!\n");
      return;
   }
   if (q->front == -1)
      q->front = 0;
   q->rear++;
   cnt++;
   q->array[q->rear] = val;
}

void POP(queue *q){
   if (empty(q)){
      printf("Queue is Underflow!!!\n");
      return;
   }
   q->front++;
   cnt--;
   if (q->front == q->rear + 1){
      q->front = -1;
      q->rear = -1;
   }
}

int TOP(queue *q){
   return q->array[q->front];
}

int SIZE(){
   return cnt;
}

int main(){
   queue *q = (queue *)malloc(sizeof(queue));
   q->front = -1;
```

```
    q->rear = -1;
    q->size = 1e3;
    q->array = (int *)malloc(q->size * sizeof(int));
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++){
        int a;
        scanf("%d", &a);
        PUSH(q, a);
    }
    printf("%d\n", SIZE());
    while (!empty(q)){
        printf("%d ", TOP(q));
        POP(q);
    }
}
```

d) circular queue using arrays

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct CircularQueue{
    int items[MAX_SIZE];
    int front, rear;
};

void initialize(struct CircularQueue *queue){
    queue->front = -1;
    queue->rear = -1;
}

int isEmpty(struct CircularQueue *queue){
    return (queue->front == -1 && queue->rear == -1);
}

int isFull(struct CircularQueue *queue){
    return ((queue->rear + 1) % MAX_SIZE == queue->front);
}

void enqueue(struct CircularQueue *queue, int item){
    if (isFull(queue)){
        printf("Queue Overflow\n");
        return;
    }
```

```
    if (isEmpty(queue))
        queue->front = queue->rear = 0;
    else
        queue->rear = (queue->rear + 1) % MAX_SIZE;
    queue->items[queue->rear] = item;
}

int dequeue(struct CircularQueue *queue){
    if (isEmpty(queue)){
        printf("Queue Underflow\n");
        exit(1);
    }
    int item = queue->items[queue->front];
    if (queue->front == queue->rear)
        queue->front = queue->rear = -1;
    else
        queue->front = (queue->front + 1) % MAX_SIZE;
    return item;
}

int peek(struct CircularQueue *queue){
    if (isEmpty(queue)){
        printf("Queue is empty\n");
        exit(1);
    }
    return queue->items[queue->front];
}

int main(){
    struct CircularQueue queue;
    initialize(&queue);
    enqueue(&queue, 1);
    enqueue(&queue, 2);
    enqueue(&queue, 3);
    printf("Front of queue: %d\n", peek(&queue));
    while (!isEmpty(&queue))
        printf("Dequeued: %d\n", dequeue(&queue));
}
```

28) Write a program to evaluate a postfix expression.

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

```
struct Stack{
   int items[MAX_SIZE];
   int top;
};

void initialize(struct Stack *stack){
   stack->top = -1;
}

bool isEmpty(struct Stack *stack){
   return stack->top == -1;
}

void push(struct Stack *stack, int item){
   if (stack->top == MAX_SIZE - 1){
      printf("Stack Overflow\n");
      exit(1);
   }
   stack->items[++stack->top] = item;
}

int pop(struct Stack *stack){
   if (isEmpty(stack)){
      printf("Stack Underflow\n");
      exit(1);
   }
   return stack->items[stack->top--];
}

int evaluatePostfix(char postfix[]){
   struct Stack stack;
   initialize(&stack);

   int i = 0;
   while (postfix[i] != '\0'){
      char token = postfix[i];
      if (isdigit(token))
         push(&stack, token - '0'); // Convert char digit to integer
      else if (token == '+' || token == '-' || token == '*' || token == '/'){
         int operand2 = pop(&stack);
         int operand1 = pop(&stack);
         int result;
         switch (token){
```

```
            case '+':
               result = operand1 + operand2;
               break;
            case '-':
               result = operand1 - operand2;
               break;
            case '*':
               result = operand1 * operand2;
               break;
            case '/':
               if (operand2 == 0){
                  printf("Division by zero is not allowed\n");
                  exit(1);
               }
               result = operand1 / operand2;
               break;
            }
            push(&stack, result);
         }
         i++;
      }

      if (!isEmpty(&stack) && stack.top == 0)
         return stack.items[0];
      else{
         printf("Invalid postfix expression\n");
         exit(1);
      }
   }

   int main(){
      char postfix[MAX_SIZE];
      printf("Enter a postfix expression: ");
      scanf("%s", postfix);
      int result = evaluatePostfix(postfix);
      printf("Result: %d\n", result);
   }
```

29) Write a program to convert an infix notation to postfix notation

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

```
#define MAX_SIZE 100
struct Stack{
   char items[MAX_SIZE];
   int top;
};

void initialize(struct Stack *stack){
   stack->top = -1;
}

bool isEmpty(struct Stack *stack){
   return stack->top == -1;
}

void push(struct Stack *stack, char item){
   if (stack->top == MAX_SIZE - 1){
      printf("Stack Overflow\n");
      exit(1);
   }
   stack->items[++stack->top] = item;
}

char pop(struct Stack *stack){
   if (isEmpty(stack)){
      printf("Stack Underflow\n");
      exit(1);
   }
   return stack->items[stack->top--];
}

char peek(struct Stack *stack){
   if (isEmpty(stack)){
      printf("Stack is empty\n");
      exit(1);
   }
   return stack->items[stack->top];
}

int precedence(char op){
   if (op == '+' || op == '-')
      return 1;
   if (op == '*' || op == '/')
      return 2;
   return 0;
```

```
}

void infixToPostfix(char infix[], char postfix[]){
   struct Stack stack;
   initialize(&stack);
   int i = 0, j = 0;
   while (infix[i] != '\0'){
      char token = infix[i];
      if (isalnum(token))
         postfix[j++] = token; // Append operands to postfix
      else if (token == '(')
         push(&stack, token);
      else if (token == ')'){
         while (!isEmpty(&stack) && peek(&stack) != '(')
            postfix[j++] = pop(&stack); // Pop operators until '('
         if (!isEmpty(&stack) && peek(&stack) != '('){
            printf("Mismatched parentheses\n");
            exit(1);
         }
         else
            pop(&stack); // Pop '('
      }
      else{ // Operator encountered
         while (!isEmpty(&stack) && precedence(token) <= precedence(peek(&stack)))
            postfix[j++] = pop(&stack); // Pop higher precedence operators
         push(&stack, token); // Push the current operator
      }
      i++;
   }
   while (!isEmpty(&stack)){
      char op = pop(&stack);
      if (op == '('){
         printf("Mismatched parentheses\n");
         exit(1);
      }
      postfix[j++] = op; // Append remaining operators to postfix
   }
   postfix[j] = '\0';
}

int main(){
   char infix[MAX_SIZE], postfix[MAX_SIZE];
   printf("Enter an infix expression: ");
   scanf("%s", infix);
```

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

30) Write a program to calculate the factorial of a given number.

```
#include <stdio.h>
typedef long long ll;
ll factorial(int n){
    ll sum = 1;
    for (int i = 1; i <= n; i++)
        sum *= i;
    return sum;
}

int main(){
    int n;
    scanf("%d", &n);
    ll ans = factorial(n);
    printf("%lld ", ans);
}
```

31) Write a program to print the Fibonacci series using recursion.

```
#include <stdio.h>
int fibonacci(int n){
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return (fibonacci(n - 1) + fibonacci(n - 2));
}

int main(){
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        printf("%d ", fibonacci(i));
}
```

32) Write a program to solve the Towers of Hanoi Problem using recursion.

```
#include <stdio.h>
void tower(int n, char beg, char aux, char end){
   if (n <= 0)
      printf("Illegal entry\n");
   else if (n == 1)
      printf("move to disc from %c to %c\n", beg, end);
   else{
      tower(n - 1, beg, end, aux);
      tower(1, beg, aux, end);
      tower(n - 1, aux, beg, end);
   }
}

int main(){
   int n;
   scanf("%d", &n);
   char a, b, c;
   tower(n, 'a', 'b', 'c');
}
```

33) Write a program to calculate the factorial of a given number.(recursion)

```
#include <stdio.h>
int factorial(int n){
   if (n >= 1)
      return n * factorial(n - 1);
   else
      return 1;
}

int main(){
   int n;
   scanf("%d", &n);
   printf("%d ", factorial(n));
}
```

34) Write a program to demonstrate several tree operations
a) Insertion
b) Inorder
c) Preorder
d) Postorder

```
#include <stdio.h>
#include <stdlib.h>
typedef struct tree_node tree;
```

```
struct tree_node{
   int data;
   tree *left_child;
   tree *right_child;
};

void *set_tree(int val){
   tree *new_node = (tree *)malloc(sizeof(tree));
   new_node->data = val;
   new_node->left_child = NULL;
   new_node->right_child = NULL;
}

void in_order(tree *root){
   if (root == NULL)
      return;
   in_order(root->left_child);
   printf("%d ", root->data);
   in_order(root->right_child);
}

void pre_order(tree *root){
   if (root == NULL)
      return;
   printf("%d ", root->data);
   pre_order(root->left_child);
   pre_order(root->right_child);
}

void post_order(tree *root){
   if (root == NULL)
      return;
   post_order(root->left_child);
   post_order(root->right_child);
   printf("%d ", root->data);
}

int main(){
   int n;
   scanf("%d", &n);
   tree *all_nodes[n];
   for (int i = 0; i < n; i++)
      all_nodes[i] = set_tree(-1);
   for (int i = 0; i < n; i++){
```

```
        int val, left, right;
        scanf("%d%d%d", &val, &left, &right);
        all_nodes[i]->data = val;
        if (left != -1)
            all_nodes[i]->left_child = all_nodes[left];
        if (right != -1)
            all_nodes[i]->right_child = all_nodes[right];
    }
    in_order(all_nodes[0]);
    printf("\n");
    pre_order(all_nodes[0]);
    printf("\n");
    post_order(all_nodes[0]);
}
```

35) Write a program to create a binary search tree

```
#include <stdio.h>
#include <stdlib.h>
typedef struct tree_node tree;
struct tree_node{
    int data;
    tree *left_child;
    tree *right_child;
};

void *set_tree(int val){
    tree *new_tree = (tree *)malloc(sizeof(tree));
    new_tree->data = val;
    new_tree->left_child = NULL;
    new_tree->right_child = NULL;
}

tree *insertion_BST(tree *root, int val){
    tree *new_node = set_tree(val);
    if (root == NULL){
        root = new_node;
        return root;
    }
    if (val < root->data)
        root->left_child = insertion_BST(root->left_child, val);
    else if (val > root->data)
        root->right_child = insertion_BST(root->right_child, val);
    return root;
```

```
}

void in_order(tree *root){
   if (root == NULL)
      return;
   in_order(root->left_child);
   printf("%d ", root->data);
   in_order(root->right_child);
}

int main(){
   int n;
   scanf("%d", &n);
   tree *root = NULL;
   for (int i = 0; i < n; i++){
      int val;
      scanf("%d", &val);
      root = insertion_BST(root, val);
   }
   in_order(root);
}
```

36) Write a program to create a graph of n vertices using an adjacency list.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int vertex;
   struct node *next;
};
struct node *createNode(int);
struct Graph
{
   int numVertices;
   struct node **adjLists;
};

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

```
struct Graph *createAGraph(int vertices)
{
   struct Graph *graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;
   graph->adjLists = malloc(vertices * sizeof(struct node *));
   int i;
   for (i = 0; i < vertices; i++)
      graph->adjLists[i] = NULL;
   return graph;
}

void addEdge(struct Graph *graph, int s, int d)
{
   struct node *newNode = createNode(d);
   newNode->next = graph->adjLists[s];
   graph->adjLists[s] = newNode;
   newNode = createNode(s);
   newNode->next = graph->adjLists[d];
   graph->adjLists[d] = newNode;
}

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

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

```
    addEdge(graph, 1, 2);
    printGraph(graph);
}
```

37) Write a program to implement Warshall's algorithm to find the path matrix

```
#include <stdio.h>
#define nV 4
#define INF 999
void printMatrix(int matrix[][nV]);
void floydWarshall(int graph[][nV]){
    int matrix[nV][nV], i, j, k;
    for (i = 0; i < nV; i++)
        for (j = 0; j < nV; j++)
            matrix[i][j] = graph[i][j];
    for (k = 0; k < nV; k++){
        for (i = 0; i < nV; i++){
            for (j = 0; j < nV; j++){
                if (matrix[i][k] + matrix[k][j] < matrix[i][j])
                    matrix[i][j] = matrix[i][k] + matrix[k][j];
            }
        }
    }
    printMatrix(matrix);
}

void printMatrix(int matrix[][nV]){
    for (int i = 0; i < nV; i++){
        for (int j = 0; j < nV; j++){
            if (matrix[i][j] == INF)
                printf("%4s", "INF");
            else
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

int main(){
    int graph[nV][nV] = {{0, 3, INF, 5},
                         {2, 0, INF, 4},
                         {INF, 1, 0, INF},
                         {INF, INF, 2, 0}};
    floydWarshall(graph);
}
```

38) Write a program to implement Warshall's algorithm to find the all pair shortest path

```
#include <stdio.h>
#define INF 9999
#define MAX_VERTICES 100
int numVertices;
int distMatrix[MAX_VERTICES][MAX_VERTICES];

void initializeMatrix(){
   for (int i = 0; i < numVertices; i++){
      for (int j = 0; j < numVertices; j++){
         if (i == j)
            distMatrix[i][j] = 0;
         else
            distMatrix[i][j] = INF;
      }
   }
}

void addEdge(int src, int dest, int weight){
   distMatrix[src][dest] = weight;
}

void warshall(){
   for (int k = 0; k < numVertices; k++){
      for (int i = 0; i < numVertices; i++){
         for (int j = 0; j < numVertices; j++){
            if (distMatrix[i][k] + distMatrix[k][j] < distMatrix[i][j])
               distMatrix[i][j] = distMatrix[i][k] + distMatrix[k][j];
         }
      }
   }
}

void displayDistanceMatrix(){
   printf("All Pair Shortest Path Matrix:\n");
   for (int i = 0; i < numVertices; i++){
      for (int j = 0; j < numVertices; j++){
         if (distMatrix[i][j] == INF)
            printf("INF\t");
         else
            printf("%d\t", distMatrix[i][j]);
      }
      printf("\n");
   }
```

```
}

int main(){
   printf("Enter the number of vertices: ");
   scanf("%d", &numVertices);
   initializeMatrix();
   int numEdges;
   printf("Enter the number of edges: ");
   scanf("%d", &numEdges);
   printf("Enter the edges and weights (source, destination, weight):\n");
   for (int i = 0; i < numEdges; i++){
      int src, dest, weight;
      scanf("%d %d %d", &src, &dest, &weight);
      addEdge(src, dest, weight);
   }
   warshall();
   displayDistanceMatrix();
}
```

39) Create a word processor using C/C++. It should be a menu driven program.
a) Text must be read from the file and after processing written into file
b) Number of line, characters, words, etc
c) Find a pattern from the text
d) Insert, delete, append a string
e) Replace a string

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <algorithm>
using namespace std;

void countStats(const string &text, int &numLines, int &numChars, int &numWords){
   numLines = 0;
   numChars = 0;
   numWords = 0;
   stringstream ss(text);
   string line;
   while (getline(ss, line)){
      numLines++;
      numChars += line.length();
      stringstream wordStream(line);
      string word;
```

```
        while (wordStream >> word){
            numWords++;
        }
    }
}

void findAndReplace(string &text, const string &pattern, const string &replacement)
{
    size_t pos = text.find(pattern);
    while (pos != string::npos){
        text.replace(pos, pattern.length(), replacement);
        pos = text.find(pattern, pos + replacement.length());
    }
}

int main(){
    string fileName;
    cout << "Enter the file name: ";
    cin >> fileName;
    ifstream inputFile(fileName);
    if (!inputFile){
        cerr << "Error opening the file." << endl;
        return 1;
    }
    string text;
    string line;
    while (getline(inputFile, line))
        text += line + "\n";
    inputFile.close();
    int choice;
    do{
        cout << "\nWord Processor Menu:" << endl;
        cout << "1. Display Text" << endl;
        cout << "2. Count Lines, Characters, and Words" << endl;
        cout << "3. Find and Replace" << endl;
        cout << "4. Insert Text" << endl;
        cout << "5. Delete Text" << endl;
        cout << "6. Append Text" << endl;
        cout << "7. Save and Exit" << endl;
        cout << "Enter your choice (1-7): ";
        cin >> choice;
        switch (choice){
        case 1:{
            cout << "Text Content:" << endl;
```

```
    cout << text << endl;
    break;
}
case 2:{
    int numLines, numChars, numWords;
    countStats(text, numLines, numChars, numWords);
    cout << "Number of Lines: " << numLines << endl;
    cout << "Number of Characters: " << numChars << endl;
    cout << "Number of Words: " << numWords << endl;
    break;
}
case 3:{
    string findPattern, replacePattern;
    cout << "Enter the pattern to find: ";
    cin.ignore();
    getline(cin, findPattern);
    cout << "Enter the replacement pattern: ";
    getline(cin, replacePattern);
    findAndReplace(text, findPattern, replacePattern);
    cout << "Text after replacement:" << endl;
    cout << text << endl;
    break;
}
case 4:{
    string insertText;
    int insertPos;
    cout << "Enter the text to insert: ";
    cin.ignore();
    getline(cin, insertText);
    cout << "Enter the position to insert (0-" << text.length() << "): ";
    cin >> insertPos;
    if (insertPos >= 0 && insertPos <= text.length()){
        text.insert(insertPos, insertText);
        cout << "Text after insertion:" << endl;
        cout << text << endl;
    }
    else
        cout << "Invalid insertion position." << endl;
    break;
}
case 5:{
    int deletePos, deleteLength;
    cout << "Enter the starting position to delete (0-" << text.length() - 1 << "): ";
    cin >> deletePos;
```

```
            cout << "Enter the number of characters to delete: ";
            cin >> deleteLength;
            if (deletePos >= 0 && deletePos < text.length() && deleteLength > 0){
               text.erase(deletePos, deleteLength);
               cout << "Text after deletion:" << endl;
               cout << text << endl;
            }
            else
               cout << "Invalid deletion position or length." << endl;
            break;
         }
         case 6:{
            string appendText;
            cout << "Enter the text to append: ";
            cin.ignore();
            getline(cin, appendText);
            text += appendText;
            cout << "Text after appending:" << endl;
            cout << text << endl;
            break;
         }
         case 7:{
            ofstream outputFile(fileName);
            if (!outputFile){
               cerr << "Error saving to the file." << endl;
               return 1;
            }
            outputFile << text;
            outputFile.close();
            cout << "Changes saved. Exiting." << endl;
            break;
         }
         default:{
            cout << "Invalid choice. Try again." << endl;
            break;
         }
         }
      } while (choice != 7);
   }
```

40) Using C structure create student records of CSE L2-I students. It should be menu driven program.

a) Fields are Roll no, Name, CGPA, address
b) Display the records.
c) Insert a new record in desired location
d) Delete a record from a desired location
e) Searching a record by Roll no
f) Sorting the records

```
#include <stdio.h>
struct student{
   int roll;
   char firstName[50];
   double cgpa;
   char adress[100];
};

int main(){
   struct student s[70];
   int i, n;
   printf("How many student in class :");
   scanf("%d", &n);
   for (i = 1; i <= n; i++){
      printf("\nEnter information of students%d:\n", i);
      printf("Enter roll: ");
      scanf("%d", &s[i].roll);
      printf("Enter first name: ");
      scanf("%s", s[i].firstName);
      printf("Enter CGPA: ");
      scanf("%lf", &s[i].cgpa);
      printf("Enter adress: ");
      scanf("%s", s[i].adress);
      printf("\n");
   }
   printf("\nDisplaying Information:\n\n");
   for (i = 1; i <= n; i++){
      printf("Information of student%d:\n", i);
      printf("Roll: %d\n", s[i].roll);
      printf("First name: ");
      puts(s[i].firstName);
      printf("CGPA: %.2lf\n", s[i].cgpa);
      printf("Adress: ");
      puts(s[i].adress);
      printf("\n");
   }
}
```