

The National Education Society of Karnataka (Regd), Bengaluru



Dr. H N National College of Engineering

Approved by AICTE, Govt. of India and affiliated to VTU
36B Cross, Jayanagar 7th Block, Bengaluru – 560070



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES LABORATORY – BCSL305

III SEMESTER, B.E

[AS PER OUTCOME BASED EDUCATION (OBE) AND CHOICE BASED CREDIT SYSTEM (CBCS) 2022 SCHEME]

Academic Year: 2025-2026

LAB MANUAL

Prepared by:

Dr. Suma Swamy & Dr. Vikhyath K B

Department of CSE

Institute Vision, Mission and Values

Vision

To impart perseverant education leading to greater heights.

Mission

- By providing well-designed physical infrastructure with technology with a supportive community.
- By building resilience through self-awareness, stress management and growth mind set.
- By developing a sense of responsibility and accountability.
- By developing empathy, integrity, self-reflection and self-improvement.

Values

Resilience, Reflection, Grit, Persistence and determination.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DATA STRUCTURES LABORATORY SEMESTER – III			
Course Code	BCSL305	CIE Marks	50
Number of Contact Hours/Week	0:0:2	SEE Marks	50
Total Number of Lab Contact Hours	28	Exam Hours	03
Credits – 1			
Course Learning Objectives:			
This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of			
<ul style="list-style-type: none"> • Dynamic memory management • Linear data structures and their applications such as stacks, queues and lists • Non-Linear data structures and their applications such as trees and graphs 			
Descriptions (if any):			
<ul style="list-style-type: none"> • Implement all the programs in “C” Programming Language and Linux OS. 			
Programs List:			
1.	Develop a Program in C for the following: a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String). b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.		
2.	Develop a Program in C for the following operations on Strings. a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions.		
3.	Develop a menu driven Program in 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 check 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		

4.	Develop a Program in 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.
5.	Develop a Program in C for the following Stack Applications <ul style="list-style-type: none"> a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks
6.	Develop a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) <ul style="list-style-type: none"> 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
7.	Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Programme, Sem, PhNo <ul style="list-style-type: none"> a. Create a SLL of N Students Data by using front insertion. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit
8.	Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo <ul style="list-style-type: none"> a. Create a DLL of N Employees Data by using end insertion. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
9.	Develop a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes <ul style="list-style-type: none"> a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
10.	Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers . <ul style="list-style-type: none"> a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit
11.	Develop a Program in C for the following operations on Graph(G) of Cities <ul style="list-style-type: none"> a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method
12.	Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Develop a Program in C that uses Hash function H: $K \rightarrow L$ as $H(K)=K \bmod m$ (remainder method), and implement hashing

	technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.
--	---

Laboratory Outcomes: The student should be able to:

- **Analyze various linear and non-linear data structures**
- **Demonstrate the working nature of different types of data structures and their applications**
- **Use appropriate searching and sorting algorithms for the give scenario.**
- **Apply the appropriate data structure for solving real world problems**

Conduct of Practical Examination:

- **Experiment distribution**
 - **For laboratories having only one part:** Students are allowed to pick one experiment from the lot with equal opportunity.
 - **For laboratories having PART A and PART B:** Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- **Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.**
- **Marks Distribution (Need to change in accordance with university regulations)**
 - c) **For laboratories having only one part – Procedure + Execution + Viva-Voce:**
 $15+70+15 = 100$ Marks
 - d) **For laboratories having PART A and PART B**
 - i. **Part A – Procedure + Execution + Viva = $6 + 28 + 6 = 40$ Marks**
 - ii. **Part B – Procedure + Execution + Viva = $9 + 42 + 9 = 60$ Marks**

1. Develop a Program in C for the following:

- a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).
- b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

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

// Structure for a day in the calendar
struct Day {
    char *dayName;
    int date;
    char *activityDesc;
};

// Function to create the calendar
struct Day *create()
{
    struct Day *calendar = (struct Day *)malloc(7 * sizeof(struct Day));
    if (calendar == NULL)
    {
        printf("Memory allocation failed.\n");
        exit(0);
    }
    return calendar;
}

// Function to read data for each day
void read(struct Day *calendar)
{
    for (int i = 0; i < 7; ++i)
    {
        printf("Enter the name of the day: ");
        char tempName[50];
        scanf("%s", tempName);

        calendar[i].dayName = (char *)malloc((strlen(tempName) + 1) * sizeof(char));
        strcpy(calendar[i].dayName, tempName);

        printf("Enter the date of the day: ");
        scanf("%d", &calendar[i].date);
    }
}
```

```
printf("Enter the activity description for %s: ", calendar[i].dayName);
char tempDesc[100];
scanf(" %[^\n]s", tempDesc);

calendar[i].activityDesc = (char *)malloc((strlen(tempDesc) + 1) * sizeof(char));
strcpy(calendar[i].activityDesc, tempDesc);
}

// Function to display the calendar
void display(struct Day *calendar)
{
    printf("\n--- Weekly Activity Report ---\n");
    for (int i = 0; i < 7; i++)
    {
        printf("Day %d: %s\n", i + 1, calendar[i].dayName);
        printf("Date: %d\n", calendar[i].date);
        printf("Activity: %s\n\n", calendar[i].activityDesc);
    }
}

// Function to free allocated memory
void freeMemory(struct Day *calendar)
{
    for (int i = 0; i < 7; i++)
    {
        free(calendar[i].dayName);
        free(calendar[i].activityDesc);
    }
    free(calendar);
}

int main()
{
    struct Day *myCalendar = create();

    printf("Please enter the details for each day of the week:\n");
    read(myCalendar);

    display(myCalendar);

    freeMemory(myCalendar);

    return 0;
}
```

Output:

Please enter the details for each day of the week:

Enter the name of the day: Monday

Enter the date of the day: 1

Enter the activity description for Monday: Gym workout

Enter the name of the day: Tuesday

Enter the date of the day: 2

Enter the activity description for Tuesday: Project meeting

Enter the name of the day: Wednesday

Enter the date of the day: 3

Enter the activity description for Wednesday: Grocery shopping

Enter the name of the day: Thursday

Enter the date of the day: 4

Enter the activity description for Thursday: Office presentation

Enter the name of the day: Friday

Enter the date of the day: 5

Enter the activity description for Friday: Family dinner

Enter the name of the day: Saturday

Enter the date of the day: 6

Enter the activity description for Saturday: Movie night

Enter the name of the day: Sunday

Enter the date of the day: 7

Enter the activity description for Sunday: Picnic with friends

--- Weekly Activity Report ---

Day 1: Monday

Date: 1

Activity: Gym workout

Day 2: Tuesday

Date: 2

Activity: Project meeting

Day 3: Wednesday

Date: 3

Activity: Grocery shopping

Day 4: Thursday

Date: 4

Activity: Office presentation

DATA STRUCTURES LABORATORY – BCSL305

Day 5: Friday

Date: 5

Activity: Family dinner

Day 6: Saturday

Date: 6

Activity: Movie night

Day 7: Sunday

Date: 7

Activity: Picnic with friends

2. Develop a Program in C for the following operations on Strings.

a) Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)

b) Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR

Support the program with functions for each of the above operations. Don't use Built-in functions.

```
#include <stdio.h>

// Function to read string manually (until newline)
void readString(char str[])
{
    char ch;
    int i = 0;
    while ((ch = getchar()) != '\n' && ch != EOF)
    {
        str[i++] = ch;
    }
    str[i] = '\0';
}

// Function to find length of a string
int strLength(char str[])
{
    int len = 0;
    while (str[len] != '\0')
    {
        len++;
    }
    return len;
}

// Function to check if PAT matches at given position in STR
int isMatch(char str[], char pat[], int pos)
{
    int i = 0;
    while (pat[i] != '\0')
    {
        if (str[pos + i] != pat[i])
        {
            return 0; // mismatch
        }
        i++;
    }
    return 1; // match found
}
```

```
// Function to find & replace PAT with REP
void findAndReplace(char str[], char pat[], char rep[])
{
    char result[500];
    int i = 0, j = 0, k, found = 0;
    int lenStr = strLength(str);
    int lenPat = strLength(pat);
    int lenRep = strLength(rep);

    while (i < lenStr)
    {
        if (isMatch(str, pat, i))
        {
            // Copy REP into result
            for (k = 0; k < lenRep; k++)
            {
                result[j++] = rep[k];
            }
            i += lenPat; // skip over PAT
            found = 1;
        }
        else
        {
            result[j++] = str[i++];
        }
    }
    result[j] = '\0';

    if (found)
    {
        printf("\nString after replacement: %s\n", result);
    }
    else
    {
        printf("\nPattern not found in the main string.\n");
    }
}

int main()
{
    char str[200], pat[50], rep[50];

    printf("Enter the main string (STR): ");
    readString(str);

    printf("Enter the pattern string (PAT): ");
    readString(pat);

    printf("Enter the replace string (REP): ");
    readString(rep);
```

```
    findAndReplace(str, pat, rep);  
  
    return 0;  
}
```

Output:

1. Enter the main string (STR): I love programming in C
Enter the pattern string (PAT): programming
Enter the replace string (REP): coding

String after replacement: I love coding in C

2. Enter the main string (STR): hello world
Enter the pattern string (PAT): test
Enter the replace string (REP): check

Pattern not found in the main string.

3. Enter the main string (STR): abcabcabc
Enter the pattern string (PAT): abc
Enter the replace string (REP): xyz

String after replacement: xyzxyzxyz

3. Develop a menu driven Program in 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 check 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

```
#include <stdio.h>
#include <stdlib.h>
#define smax 6

void push(int);
int pop();
void palin();
void display();

int top = -1, stk[smax];

void main()
{
    int ch, ele, item;
    while (1)
    {
        printf("\nSTACK MENU\n");
        printf("1. PUSH\n");
        printf("2. POP\n");
        printf("3. DISPLAY\n");
        printf("4. PALINDROME CHECK\n");
        printf("5. EXIT\n");
```

```
printf("ENTER YOUR CHOICE\n");
scanf("%d", &ch);

switch (ch)
{
    case 1:
        printf("Enter the item to be inserted\n");
        scanf("%d", &item);
        push(item);
        break;

    case 2:
        ele = pop();
        if (ele != -1)
            printf("The deleted element is %d\n", ele);
        break;

    case 3:
        display();
        break;

    case 4:
        palin();
        break;

    case 5:
        exit(0);

    default:
        printf("INVALID CHOICE\n");
}
```

```
void push(int item)
{
    if (top == smax - 1)
    {
        printf("\nSTACK OVERFLOW\n");
    }
    else
    {
        stk[++top] = item;
        printf("Pushed %d onto stack.\n", item);
    }
}

int pop()
{
    int d;
    if (top == -1)
    {
        printf("\nSTACK UNDERFLOW\n");
        return -1;
    }
    else
    {
        d = stk[top--];
        return d;
    }
}

void display()
{
    int i;
    if (top == -1)
    {
        printf("\nSTACK UNDERFLOW\n");
    }
```

```
    }
else
{
    printf("\nCONTENTS OF STACK ARE\n");
    for (i = top; i >= 0; i--)
        printf("%d\n", stk[i]);
}
}

void palin()
{
    int n, a, d, temp, flag = 1;

    // Clear stack
    while (top != -1)
        a = pop();

    printf("Enter the number\n");
    scanf("%d", &n);

    temp = n;
    while (temp != 0)
    {
        d = temp % 10;
        push(d);
        temp = temp / 10;
    }

    if (top == 0)
    {
        printf("\nPALINDROME CANNOT BE CHECKED\n");
        return;
    }
```

```
temp = n;
while (temp != 0)
{
    if (pop() != temp % 10)
    {
        flag = 0;
        break;
    }
    temp /= 10;
}

if (flag == 1)
    printf("The number %d is a palindrome\n", n);
else
    printf("The number %d is not palindrome\n", n);
}
```

Output:

```
STACK MENU
1. PUSH
2. POP
3. DISPLAY
4. PALINDROME CHECK
5. EXIT
ENTER YOUR CHOICE
1
Enter the item to be inserted
10
Pushed 10 onto stack.
```

STACK MENU

1. PUSH

- 2. POP
- 3. DISPLAY
- 4. PALINDROME CHECK
- 5. EXIT

ENTER YOUR CHOICE

1

Enter the item to be inserted

20

Pushed 20 onto stack.

STACK MENU

- 1. PUSH
- 2. POP
- 3. DISPLAY
- 4. PALINDROME CHECK
- 5. EXIT

ENTER YOUR CHOICE

3

CONTENTS OF STACK ARE

20

10

STACK MENU

- 1. PUSH
- 2. POP
- 3. DISPLAY
- 4. PALINDROME CHECK
- 5. EXIT

ENTER YOUR CHOICE

2

The deleted element is 20

STACK MENU

1. PUSH
2. POP
3. DISPLAY
4. PALINDROME CHECK
5. EXIT

ENTER YOUR CHOICE

4

Enter the number

121

The number 121 is a palindrome

STACK MENU

1. PUSH
2. POP
3. DISPLAY
4. PALINDROME CHECK
5. EXIT

ENTER YOUR CHOICE

4

Enter the number

123

The number 123 is not palindrome

STACK MENU

1. PUSH
2. POP
3. DISPLAY
4. PALINDROME CHECK
5. EXIT

ENTER YOUR CHOICE

5

4. Develop a Program in 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.

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

#define MAX 50

int precedence(char);
char pop();
void push(char);

char postfix[MAX], infix[MAX], s[MAX];
int top = -1;

int main()
{
    int i, j = 0;
    char ch, elem;

    printf("Enter a valid infix expression:\n");
    fgets(infix, sizeof(infix), stdin);

    // Remove newline added by fgets
    infix[strcspn(infix, "\n")] = '\0';

    push('#'); // Sentinel at stack bottom

    for (i = 0; infix[i] != '\0'; i++)
    {
        ch = infix[i];
```

```
if (isalnum(ch))           // Operand → directly to postfix
{
    postfix[j++] = ch;
}
else if (ch == '(')
{
    push(ch);
}
else if (ch == ')')
{
    while (s[top] != '(')
        postfix[j++] = pop();
    elem = pop();           // remove '('
}
else      // Operator
{
    while (precedence(s[top]) >= precedence(ch))
    {
        if ((ch == '^') && (s[top] == '^'))
            break;          // right-associativity of ^
        postfix[j++] = pop();
    }
    push(ch);
}

// Pop remaining operators (excluding sentinel)
while (s[top] != '#')
    postfix[j++] = pop();

postfix[j] = '\0';

printf("The postfix expression is: %s\n", postfix);
```

```
return 0;  
}  
  
int precedence(char e)  
{  
    switch (e)  
    {  
        case '#': return -1;  
        case '(': return 0;  
        case '+':  
        case '-': return 1;  
        case '*':  
        case '/':  
        case '%': return 2;  
        case '^': return 3;  
        default: return -1; // safety  
    }  
}  
  
void push(char ele)  
{  
    if (top < MAX - 1)  
        s[++top] = ele;  
}  
  
char pop()  
{  
    if (top >= 0)  
        return s[top--];  
    return '#'; // fallback, should not happen  
}
```

Output:

1. Enter a valid infix expression:

$(A+B)*C$

The postfix expression is: AB+C*

2. Enter a valid infix expression:

$A+B*C$

The postfix expression is: ABC*+

3. Enter a valid infix expression:

$(A+B)*(C-D)$

The postfix expression is: AB+CD-*

4. Enter a valid infix expression:

A^B^C

The postfix expression is: ABC^^

5. Develop a Program in 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

// Evaluation of Suffix (Postfix) expression

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

#define MAX 20

// Function prototypes
void push(int e);
int pop();

// Global declarations
int a[MAX], top = -1;

void main()
{
    char post[MAX], ch;
    int i, op1, op2, result;

    printf("Program to evaluate postfix expression\n");
    printf("Enter the postfix expression:\n");
    scanf("%s", post);

    for (i = 0; post[i] != '\0'; i++)
    {
        ch = post[i];           // Take one symbol from the expression
        if (isdigit(ch))        // Check if it is an operand
        {
            push(ch - '0');
        }
        else
        {
            op2 = pop();
            op1 = pop();
            switch (ch)          // Perform the operation depending upon the operator
            {
                case '+': result = op1 + op2; break;
                case '-': result = op1 - op2; break;
                case '*': result = op1 * op2; break;
                case '/': result = op1 / op2; break;
            }
            push(result);
        }
    }
}
```

```
        case '^': result = pow(op1, op2); break;
        case '%': result = op1 % op2; break;
        default: printf("Invalid character\n");
    }
    push(result); // Push the result onto the stack
}
}

// Pop the final result from the stack
printf("Result of the above expression is %d\n", pop());
}

/* Function to push */
void push(int e)
{
    ++top;
    a[top] = e;
}

/* Function to pop */
int pop()
{
    int ele;
    ele = a[top];
    --top;
    return ele;
}
```

Output:

1. Program to evaluate postfix expression

Enter the postfix expression:

63+5*

Result of the above expression is 45

2. Program to evaluate postfix expression

Enter the postfix expression:

82/3-

Result of the above expression is 1

// Solving Tower of Hanoi problem with n disks

```
#include <stdio.h>

int count = 0;

void tower(int n, char s, char t, char d)
{
    if (n == 1)
    {
        printf("Move disc 1 from %c to %c\n", s, d);
        count++;
        return;
    }

    tower(n - 1, s, d, t);
    printf("Move disc %d from %c to %c\n", n, s, d);
    count++;
    tower(n - 1, t, s, d);
}

int main()
{
    int n;
    printf("Enter the number of discs\n");
    scanf("%d", &n);

    tower(n, 'A', 'B', 'C');
    printf("Total number of moves = %d\n", count);

    return 0;
}
```

Output:

Enter the number of discs

3

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total number of moves = 7

6. Develop a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

- 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.

```
#include <stdio.h>
#include <stdlib.h>
#define smax 5

void insert();
void del();
void display();

int rear = -1, front = 0, count = 0;
char queue[smax];

void main()
{
    int ch;

    while (1)
    {
        printf("\nCIRCULAR QUEUE IMPLEMENTATION\n");
        printf("1. INSERT\n");
        printf("2. DELETE\n");
        printf("3. DISPLAY\n");
        printf("4. EXIT\n");
        printf("\nEnter YOUR CHOICE: ");
        scanf("%d", &ch);
    }
}
```

```
switch (ch)
{
    case 1:
        insert();
        break;
    case 2:
        del();
        break;
    case 3:
        display();
        break;
    case 4:
        exit(0);
    default:
        printf("\nINVALID CHOICE\n");
}
}

void insert()
{
    char item, ch;

    if (count >= smax)
    {
        printf("\nQUEUE OVERFLOW\n");
    }
    else
    {
        ch = getchar() // to consume leftover newline
        printf("Enter the item to be inserted: ");
        scanf("%c", &item);
```

```
rear = (rear + 1) % smax;
queue[rear] = item;
count++;
}

}

void del()
{
char d;

if (count <= 0)
{
printf("\nQUEUE UNDERFLOW\n");
}
else
{
d = queue[front];
front = (front + 1) % smax;
printf("\nThe deleted element is %c\n", d);
count--;
}
}

void display()
{
int i, j;

if (count <= 0)
{
printf("\nQUEUE UNDERFLOW\n");
}
else
{
i = front;
```

```
printf("\nQueue elements are:\n");
for (j = 1; j <= count; j++)
{
    printf("%c\n", queue[i]);
    i = (i + 1) % smax;
}
}
```

Output:

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: A

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: B

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: C

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 3

Queue elements are:

A
B
C

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 2

The deleted element is A

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 3

Queue elements are:

B
C

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: D

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: E

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 1

Enter the item to be inserted: F

QUEUE OVERFLOW

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 2

The deleted element is B

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 2

The deleted element is C

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT

ENTER YOUR CHOICE: 2

The deleted element is D

CIRCULAR QUEUE IMPLEMENTATION

1. INSERT

- 2. DELETE
- 3. DISPLAY
- 4. EXIT

ENTER YOUR CHOICE: 2

The deleted element is E

CIRCULAR QUEUE IMPLEMENTATION

- 1. INSERT
- 2. DELETE
- 3. DISPLAY
- 4. EXIT

ENTER YOUR CHOICE: 2

QUEUE UNDERFLOW

CIRCULAR QUEUE IMPLEMENTATION

- 1. INSERT
- 2. DELETE
- 3. DISPLAY
- 4. EXIT

ENTER YOUR CHOICE: 4

7. Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Programme, Sem, PhNo

- a. Create a SLL of N Students Data by using front insertion.**
- b. Display the status of SLL and count the number of nodes in it**
- c. Perform Insertion / Deletion at End of SLL**
- d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)**
- e. Exit**

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

// Define the student node structure
typedef struct Student {
    char USN[20];
    char Name[50];
    char Programme[30];
    int Sem;
    char PhNo[15];
    struct Student* next;
} Student;

Student* head = NULL;
```

```
// Function to create a new node
Student* createNode() {
    Student* newNode = (Student*)malloc(sizeof(Student));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}
```

```
printf("Enter USN: ");
scanf("%s", newNode->USN);

printf("Enter Name: ");
getchar() // clear leftover newline
fgets(newNode->Name, sizeof(newNode->Name), stdin);
newNode->Name[strcspn(newNode->Name, "\n")] = '\0'; // remove newline

printf("Enter Programme: ");
scanf("%s", newNode->Programme);

printf("Enter Semester: ");
scanf("%d", &newNode->Sem);

printf("Enter Phone Number: ");
scanf("%s", newNode->PhNo);

newNode->next = NULL;
return newNode;
}

// a. Insert at front
void insertFront() {
    Student* newNode = createNode();
    newNode->next = head;
    head = newNode;
    printf("Student inserted at front.\n");
}

// c. Insert at end
void insertEnd() {
    Student* newNode = createNode();
    if (head == NULL) {
        head = newNode;
    }
}
```

```
    } else {
        Student* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("Student inserted at end.\n");
}

// d. Delete from front
void deleteFront() {
    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    Student* temp = head;
    head = head->next;
    printf("Deleted student: %s (%s)\n", temp->Name, temp->USN);
    free(temp);
}

// c. Delete from end
void deleteEnd() {
    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    if (head->next == NULL) {
        printf("Deleted student: %s (%s)\n", head->Name, head->USN);
        free(head);
        head = NULL;
        return;
    }
    Student* temp = head;
```

```
while (temp->next->next != NULL)
    temp = temp->next;

printf("Deleted student: %s (%s)\n", temp->next->Name, temp->next->USN);
free(temp->next);
temp->next = NULL;
}

// b. Display and count
void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    Student* temp = head;
    int count = 0;

    printf("\nStudent List:\n");
    printf("USN\tName\tProgramme\tSem\tPhone\n");
    printf("-----\n");

    while (temp != NULL) {
        printf("%s\t%s\t%s\t%d\t%s\n",
               temp->USN, temp->Name, temp->Programme,
               temp->Sem, temp->PhNo);
        count++;
        temp = temp->next;
    }

    printf("\nTotal students: %d\n", count);
}

// Free memory before exit
```

```
void freeList() {
    Student* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

// Main menu
int main() {
    int choice, n, i;

    while (1) {
        printf("\n===== Singly Linked List Menu =====\n");
        printf("1. Create SLL with N students (Front Insertion)\n");
        printf("2. Display SLL and Count Nodes\n");
        printf("3. Insert at End\n");
        printf("4. Delete from End\n");
        printf("5. Insert at Front (Stack Push)\n");
        printf("6. Delete from Front (Stack Pop)\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter number of students: ");
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    printf("\n--- Student %d ---\n", i + 1);
                    insertFront();
                }
                break;
        }
    }
}
```

```
case 2:  
    display();  
    break;  
  
case 3:  
    insertEnd();  
    break;  
  
case 4:  
    deleteEnd();  
    break;  
  
case 5:  
    insertFront();  
    break;  
  
case 6:  
    deleteFront();  
    break;  
  
case 7:  
    freeList();  
    printf("Exiting program. Memory freed.\n");  
    exit(0);  
  
default:  
    printf("Invalid choice. Try again.\n");  
}  
}  
  
return 0;  
}
```

Output:

===== Singly Linked List Menu =====

1. Create SLL with N students (Front Insertion)
2. Display SLL and Count Nodes
3. Insert at End
4. Delete from End
5. Insert at Front (Stack Push)
6. Delete from Front (Stack Pop)
7. Exit

Enter your choice: 1

Enter number of students: 2

--- Student 1 ---

Enter USN: 1HC24EC001

Enter Name: Arjun

Enter Programme: ECE

Enter Semester: 3

Enter Phone Number: 9876543210

Student inserted at front.

--- Student 2 ---

Enter USN: 1HC24EC002

Enter Name: Priya

Enter Programme: ECE

Enter Semester: 3

Enter Phone Number: 9876500000

Student inserted at front.

===== Singly Linked List Menu =====

1. Create SLL with N students (Front Insertion)
2. Display SLL and Count Nodes
3. Insert at End

DATA STRUCTURES LABORATORY – BCSL305

- 4. Delete from End
- 5. Insert at Front (Stack Push)
- 6. Delete from Front (Stack Pop)
- 7. Exit

Enter your choice: 2

Student List:

USN	Name	Programme	Sem	Phone
<hr/>				
1HC24EC002	Priya	ECE	3	9876500000
1HC24EC001	Arjun	ECE	3	9876543210

Total students: 2

===== Singly Linked List Menu =====

Enter your choice: 3

--- Insert at End ---

Enter USN: 1HC24EC003

Enter Name: Kiran

Enter Programme: ECE

Enter Semester: 5

Enter Phone Number: 9876512345

Student inserted at end.

===== Singly Linked List Menu =====

Enter your choice: 5

--- Insert at Front (Stack Push) ---

Enter USN: 1HC24EC004

Enter Name: Sneha

Enter Programme: ECE

Enter Semester: 2

Enter Phone Number: 9876522222

DATA STRUCTURES LABORATORY – BCSL305

Student inserted at front.

===== Singly Linked List Menu =====

Enter your choice: 2

Student List:

USN	Name	Programme	Sem	Phone
1HC24EC004	Sneha	ECE	2	9876522222
1HC24EC002	Priya	ECE	3	9876500000
1HC24EC001	Arjun	ECE	3	9876543210
1HC24EC003	Kiran	ECE	5	9876512345

Total students: 4

===== Singly Linked List Menu =====

Enter your choice: 6

Deleted student: Sneha (1HC24EC004)

===== Singly Linked List Menu =====

Enter your choice: 4

Deleted student: Kiran (1HC24EC003)

===== Singly Linked List Menu =====

Enter your choice: 2

Student List:

USN	Name	Programme	Sem	Phone
1HC24EC002	Priya	ECE	3	9876500000
1HC24EC001	Arjun	ECE	3	9876543210

Total students: 2

===== Singly Linked List Menu =====

Enter your choice: 7

Exiting program. Memory freed.

8. Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo.

- a. Create a DLL of N Employees Data by using end insertion.**
- b. Display the status of DLL and count the number of nodes in it**
- c. Perform Insertion and Deletion at End of DLL**
- d. Perform Insertion and Deletion at Front of DLL**
- e. Demonstrate how this DLL can be used as Double Ended Queue.**
- f. Exit**

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

struct Employee
{
    char SSN[20];
    char name[50];
    char dept[30];
    char designation[30];
    float sal;
    char phNo[15];
    struct Employee* prev;
    struct Employee* next;
};

typedef struct Employee Employee;

Employee* head = NULL;
// Function to create a new Employee node
Employee* createEmployeeNode()
{
    Employee* newEmployee = (Employee*)malloc(sizeof(Employee));
```

```
if (newEmployee == NULL)
{
    printf("Memory allocation failed!\n");
    return NULL;
}

printf("\nEnter SSN: ");
scanf("%s", newEmployee->SSN);
printf("Enter Name: ");
scanf(" %[^\n]", newEmployee->name);           // allows spaces
printf("Enter Department: ");
scanf(" %[^\n]", newEmployee->dept);
printf("Enter Designation: ");
scanf(" %[^\n]", newEmployee->designation);
printf("Enter Salary: ");
scanf("%f", &newEmployee->sal);
printf("Enter Phone Number: ");
scanf("%s", newEmployee->phNo);

newEmployee->prev = NULL;
newEmployee->next = NULL;

return newEmployee;
}

// Function to create a DLL of N employees
void createDLL(int N)
{
    Employee* newEmployee;
    Employee* temp = NULL;

    for (int i = 0; i < N; i++)
    {
        newEmployee = createEmployeeNode();
```

```
if (head == NULL)
{
    head = newEmployee;
}
else
{
    temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newEmployee;
    newEmployee->prev = temp;
}

printf("\nDoubly Linked List created with %d employees.\n", N);
}

// Function to display the status of DLL and count nodes
void displayStatus()
{
    if (head == NULL)
    {
        printf("The list is empty.\n");
        return;
    }

Employee* temp = head;
int count = 0;

printf("\nEmployee Data in DLL:\n");
printf("-----\n");
```

```
while (temp != NULL)
{
    printf("SSN: %s | Name: %s | Dept: %s | Designation: %s | Salary: %.2f | PhNo: %s\n",
           temp->SSN, temp->name, temp->dept, temp->designation, temp->sal, temp->phNo);
    count++;
    temp = temp->next;
}

printf("-----\n");
printf("Total number of employees: %d\n", count);
}

// Function to insert at end of DLL
void insertAtEnd()
{
    Employee* newEmployee = createEmployeeNode();

    if (head == NULL)
    {
        head = newEmployee;
    }
    else
    {
        Employee* temp = head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newEmployee;
        newEmployee->prev = temp;
    }
    printf("\nEmployee data inserted at the end.\n");
}
```

```
// Function to delete from end of DLL
void deleteAtEnd()
{
    if (head == NULL)
    {
        printf("The list is empty. No data to delete.\n");
        return;
    }

Employee* temp = head;
while (temp->next != NULL)
{
    temp = temp->next;

    if (temp->prev != NULL)
    {
        temp->prev->next = NULL;
    }
    else
    {
        head = NULL; // only one node
    }

    free(temp);
    printf("\nEmployee data deleted from the end.\n");
}

// Function to insert at front of DLL
void insertAtFront()
{
    Employee* newEmployee = createEmployeeNode();

    if (head == NULL)
```

```
{  
    head = newEmployee;  
}  
else  
{  
    newEmployee->next = head;  
    head->prev = newEmployee;  
    head = newEmployee;  
}  
printf("\nEmployee data inserted at the front.\n");  
}  
  
// Function to delete from front of DLL  
void deleteAtFront()  
{  
    if (head == NULL)  
    {  
        printf("The list is empty. No data to delete.\n");  
        return;  
    }  
  
    Employee* temp = head;  
    head = head->next;  
    if (head != NULL)  
    {  
        head->prev = NULL;  
    }  
  
    free(temp);  
    printf("\nEmployee data deleted from the front.\n");  
}  
  
// Function to demonstrate DLL as Double Ended Queue (DEQ)
```

```
void demoDEQ()
{
    int choice;
    do
    {
        printf("\nDouble Ended Queue Operations:\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at End\n");
        printf("3. Delete from Front\n");
        printf("4. Delete from End\n");
        printf("5. Display\n");
        printf("6. Go back to Main Menu\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1: insertAtFront(); break;
            case 2: insertAtEnd(); break;
            case 3: deleteAtFront(); break;
            case 4: deleteAtEnd(); break;
            case 5: displayStatus(); break;
            case 6: return;
            default: printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);
}

int main()
{
    int choice, N, subChoice;

    do
    {
```

```
printf("\nMain Menu:\n");
printf("1. Create DLL of N Employee Data\n");
printf("2. Display DLL and count the nodes\n");
printf("3. Insertion and Deletion at End\n");
printf("4. Insertion and Deletion at Front\n");
printf("5. Demonstrate DLL as Double Ended Queue\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
    case 1:
        printf("Enter the number of employees: ");
        scanf("%d", &N);
        createDLL(N);
        break;

    case 2:
        displayStatus();
        break;

    case 3:
        printf("\nEnd Operations:\n");
        printf("1. Insert at End\n");
        printf("2. Delete from End\n");
        printf("Enter your choice: ");
        scanf("%d", &subChoice);
        if (subChoice == 1) insertAtEnd();
        else if (subChoice == 2) deleteAtEnd();
        else printf("Invalid choice!\n");
        break;
}
```

case 4:

```
printf("\nFront Operations:\n");
printf("1. Insert at Front\n");
printf("2. Delete from Front\n");
printf("Enter your choice: ");
scanf("%d", &subChoice);
if (subChoice == 1) insertAtFront();
else if (subChoice == 2) deleteAtFront();
else printf("Invalid choice!\n");
break;
```

case 5:

```
demoDEQ();
break;
```

case 6:

```
printf("Exiting the program.\n");
break;
```

default:

```
printf("Invalid choice! Please try again.\n");
}
} while (choice != 6);
```

```
return 0;
}
```

Output:

Main Menu:

1. Create DLL of N Employee Data
2. Display DLL and count the nodes
3. Insertion and Deletion at End
4. Insertion and Deletion at Front

DATA STRUCTURES LABORATORY – BCSL305

5. Demonstrate DLL as Double Ended Queue

6. Exit

Enter your choice: 1

Enter the number of employees: 2

Enter SSN: E101

Enter Name: Alice Johnson

Enter Department: HR

Enter Designation: Manager

Enter Salary: 55000

Enter Phone Number: 9876543210

Enter SSN: E102

Enter Name: Bob Smith

Enter Department: IT

Enter Designation: Engineer

Enter Salary: 45000

Enter Phone Number: 9876501234

Doubly Linked List created with 2 employees.

Main Menu:

Enter your choice: 2

Employee Data in DLL:

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |

PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 | PhNo: 9876501234

Total number of employees: 2

DATA STRUCTURES LABORATORY – BCSL305

Main Menu:

Enter your choice: 3

End Operations:

1. Insert at End
2. Delete from End

Enter your choice: 1

Enter SSN: E103

Enter Name: Charlie Davis

Enter Department: Finance

Enter Designation: Analyst

Enter Salary: 40000

Enter Phone Number: 9876009876

Employee data inserted at the end.

Main Menu:

Enter your choice: 2

Employee Data in DLL:

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |

PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 | PhNo: 9876501234

SSN: E103 | Name: Charlie Davis | Dept: Finance | Designation: Analyst | Salary: 40000.00 |
PhNo: 9876009876

Total number of employees: 3

Main Menu:

Enter your choice: 3

DATA STRUCTURES LABORATORY – BCSL305

End Operations:

1. Insert at End
2. Delete from End

Enter your choice: 2

Employee data deleted from the end.

Main Menu:

Enter your choice: 2

Employee Data in DLL:

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |
PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 | PhNo:
9876501234

Total number of employees: 2

Main Menu:

Enter your choice: 4

Front Operations:

1. Insert at Front
2. Delete from Front

Enter your choice: 1

Enter SSN: E100

Enter Name: David Miller

Enter Department: Sales

Enter Designation: Executive

Enter Salary: 42000

Enter Phone Number: 9876123456

DATA STRUCTURES LABORATORY – BCSL305

Employee data inserted at the front.

Main Menu:

Enter your choice: 2

Employee Data in DLL:

SSN: E100 | Name: David Miller | Dept: Sales | Designation: Executive | Salary: 42000.00 |

PhNo: 9876123456

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |

PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 |

PhNo: 9876501234

Total number of employees: 3

Main Menu:

Enter your choice: 4

Front Operations:

1. Insert at Front
2. Delete from Front

Enter your choice: 2

Employee data deleted from the front.

Main Menu:

Enter your choice: 2

Employee Data in DLL:

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |

PhNo: 9876543210

DATA STRUCTURES LABORATORY – BCSL305

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 | PhNo: 9876501234

Total number of employees: 2

Main Menu:

Enter your choice: 5

Double Ended Queue Operations:

1. Insert at Front
2. Insert at End
3. Delete from Front
4. Delete from End
5. Display
6. Go back to Main Menu

Enter your choice: 1

Enter SSN: E099

Enter Name: Emma Watson

Enter Department: Admin

Enter Designation: Clerk

Enter Salary: 30000

Enter Phone Number: 9876111111

Employee data inserted at the front.

Double Ended Queue Operations:

Enter your choice: 2

Enter SSN: E104

Enter Name: Frank Lee

Enter Department: IT

Enter Designation: Support

Enter Salary: 38000

DATA STRUCTURES LABORATORY – BCSL305

Enter Phone Number: 9876222222

Employee data inserted at the end.

Double Ended Queue Operations:

Enter your choice: 5

Employee Data in DLL:

SSN: E099 | Name: Emma Watson | Dept: Admin | Designation: Clerk | Salary: 30000.00 |

PhNo: 9876111111

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |

PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 |

PhNo: 9876501234

SSN: E104 | Name: Frank Lee | Dept: IT | Designation: Support | Salary: 38000.00 | PhNo:

9876222222

Total number of employees: 4

Double Ended Queue Operations:

Enter your choice: 3

Employee data deleted from the front.

Double Ended Queue Operations:

Enter your choice: 4

Employee data deleted from the end.

Double Ended Queue Operations:

Enter your choice: 5

DATA STRUCTURES LABORATORY – BCSL305

Employee Data in DLL:

SSN: E101 | Name: Alice Johnson | Dept: HR | Designation: Manager | Salary: 55000.00 |
PhNo: 9876543210

SSN: E102 | Name: Bob Smith | Dept: IT | Designation: Engineer | Salary: 45000.00 | PhNo:
9876501234

Total number of employees: 2

Double Ended Queue Operations:

Enter your choice: 6

Main Menu:

Enter your choice: 6

Exiting the program.

9. Develop a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes

a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$

b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

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

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

// Define the node structure for a Singly Circular Linked List (SCLL) with header node
typedef struct Node {
    int coefficient;
    int expX;
    int expY;
    int expZ;
    struct Node* next;
} Node;

// Function to create a header node (circular self-loop)
Node* createHeader() {
    Node* header = (Node*)malloc(sizeof(Node));
    header->coefficient = 0;
    header->expX = header->expY = header->expZ = 0;
    header->next = header; // circular property
    return header;
}

// Function to create a new term node
Node* createNode(int coef, int x, int y, int z) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coefficient = coef;
    newNode->expX = x;
```

```
newNode->expY = y;
newNode->expZ = z;
newNode->next = NULL;
return newNode;
}

// Insert a new term into polynomial (end insertion, after header)
void insertTerm(Node* header, int coef, int x, int y, int z) {
    if (coef == 0) return; // skip zero coefficient terms
    Node* newNode = createNode(coef, x, y, z);
    Node* temp = header;
    while (temp->next != header) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = header;
}

// Add term to polynomial (combining like terms)
void addTerm(Node* header, int coef, int x, int y, int z) {
    if (coef == 0) return;

    Node* temp = header->next;
    while (temp != header) {
        if (temp->expX == x && temp->expY == y && temp->expZ == z) {
            temp->coefficient += coef;
            return;
        }
        temp = temp->next;
    }
    insertTerm(header, coef, x, y, z);
}

// Evaluate polynomial at given (x,y,z)
```

```
int evaluatePolynomial(Node* header, int x, int y, int z) {
    int result = 0;
    Node* temp = header->next;
    while (temp != header) {
        int termValue = temp->coefficient;
        termValue *= (int)pow(x, temp->expX);
        termValue *= (int)pow(y, temp->expY);
        termValue *= (int)pow(z, temp->expZ);
        result += termValue;
        temp = temp->next;
    }
    return result;
}

// Add two polynomials and return the result polynomial
Node* addPolynomials(Node* poly1, Node* poly2) {
    Node* result = createHeader();
    Node* temp = poly1->next;
    while (temp != poly1) {
        addTerm(result, temp->coefficient, temp->expX, temp->expY, temp->expZ);
        temp = temp->next;
    }
    temp = poly2->next;
    while (temp != poly2) {
        addTerm(result, temp->coefficient, temp->expX, temp->expY, temp->expZ);
        temp = temp->next;
    }
    return result;
}

// Display polynomial in readable format
void displayPolynomial(Node* header) {
    Node* temp = header->next;
    if (temp == header) {
```

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

int first = 1;
while (temp != header) {
    if (temp->coefficient != 0) {
        if (!first) {
            if (temp->coefficient > 0) printf(" + ");
            else printf(" - ");
        } else {
            if (temp->coefficient < 0) printf("-");
            first = 0;
        }
    }

    int coef = abs(temp->coefficient);
    printf("%d", coef);

    if (temp->expX > 0) printf("x^%d", temp->expX);
    if (temp->expY > 0) printf("y^%d", temp->expY);
    if (temp->expZ > 0) printf("z^%d", temp->expZ);
}

temp = temp->next;
}
printf("\n");
}

// Main function
int main() {
    Node* poly1 = createHeader();
    Node* poly2 = createHeader();
    Node* polySum;

    // Polynomial 1: P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3
```

```
insertTerm(poly1, 6, 2, 2, 1);
insertTerm(poly1, -4, 0, 1, 5);
insertTerm(poly1, 3, 3, 1, 1);
insertTerm(poly1, 2, 1, 5, 1);
insertTerm(poly1, -2, 1, 1, 3);

printf("Polynomial 1: ");
displayPolynomial(poly1);

// Polynomial 2: define your own (example: x^2y + 5z^3)
insertTerm(poly2, 1, 2, 1, 0);
insertTerm(poly2, 5, 0, 0, 3);

printf("Polynomial 2: ");
displayPolynomial(poly2);

// Add the polynomials
polySum = addPolynomials(poly1, poly2);
printf("Sum of Polynomials: ");
displayPolynomial(polySum);

// Evaluate polynomial sum at x=1, y=1, z=1
int result = evaluatePolynomial(polySum, 1, 1, 1);
printf("Evaluation of POLYSUM(1,1,1): %d\n", result);

return 0;
}
```

Output:

Polynomial 1: $6x^2y^2z^1 - 4y^1z^5 + 3x^3y^1z^1 + 2x^1y^5z^1 - 2x^1y^1z^3$

Polynomial 2: $1x^2y^1 + 5z^3$

Sum of Polynomials: $6x^2y^2z^1 - 4y^1z^5 + 3x^3y^1z^1 + 2x^1y^5z^1 - 2x^1y^1z^3 + 1x^2y^1 + 5z^3$

Evaluation of POLYSUM(1,1,1): 11

10. Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.

- a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
- b. Traverse the BST in Inorder, Preorder and Post Order
- c. Search the BST for a given element (KEY) and report the appropriate message
- d. Exit

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

// Define the structure for a node in the Binary Search Tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

typedef struct Node Node;

// Function to create a new node with a given value
Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a value into the BST (ignoring duplicates)
Node* insertNode(Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

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

// Function for Inorder traversal (Left, Root, Right)
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
```

```

        }

    }

// Function for Preorder traversal (Root, Left, Right)
void preorderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function for Postorder traversal (Left, Right, Root)
void postorderTraversal(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to search for a given key in the BST
Node* searchNode(Node* root, int key) {
    if (root == NULL || root->data == key) {
        return root;
    }

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

// Function to delete the entire tree (Free memory)
void deleteTree(Node* root) {
    if (root != NULL) {
        deleteTree(root->left);
        deleteTree(root->right);
        free(root);
    }
}

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

    // Predefined list of integers
    int values[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
    int n = sizeof(values) / sizeof(values[0]);
}

```

```
do {
    printf("\nMenu:\n");
    printf("1. Create BST of Integers (Predefined List)\n");
    printf("2. Traverse the BST in Inorder\n");
    printf("3. Traverse the BST in Preorder\n");
    printf("4. Traverse the BST in Postorder\n");
    printf("5. Search the BST for a given key\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            // Delete old tree if it exists
            deleteTree(root);
            root = NULL;
            for (int i = 0; i < n; i++) {
                root = insertNode(root, values[i]);
            }
            printf("BST created with predefined integers: 6, 9, 5, 2, 8, 15, 24, 14, 7\n");
            break;

        case 2:
            printf("Inorder Traversal: ");
            inorderTraversal(root);
            printf("\n");
            break;

        case 3:
            printf("Preorder Traversal: ");
            preorderTraversal(root);
            printf("\n");
            break;

        case 4:
            printf("Postorder Traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;

        case 5:
            printf("Enter key to search: ");
            scanf("%d", &key);
            Node* foundNode = searchNode(root, key);
            if (foundNode != NULL) {
                printf("Key %d found in the BST.\n", key);
            } else {
                printf("Key %d not found in the BST.\n", key);
            }
    }
}
```

```
        break;

    case 6:
        printf("Exiting program.\n");
        break;

    default:
        printf("Invalid choice. Please try again.\n");
    }

} while (choice != 6);

// Delete the BST and free memory
deleteTree(root);
return 0;
}
```

Output:

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 1

BST created with predefined integers: 6, 9, 5, 2, 8, 15, 24, 14, 7

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 2

Inorder Traversal: 2 5 6 7 8 9 14 15 24

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 3

Preorder Traversal: 6 5 2 9 8 7 15 14 24

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 4

Postorder Traversal: 2 5 7 8 14 24 15 9 6

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 5

Enter key to search: 14

Key 14 found in the BST.

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 5

Enter key to search: 10

Key 10 not found in the BST.

Menu:

1. Create BST of Integers (Predefined List)
2. Traverse the BST in Inorder
3. Traverse the BST in Preorder
4. Traverse the BST in Postorder
5. Search the BST for a given key
6. Exit

Enter your choice: 6

Exiting program.

11. Develop a Program in C for the following operations on Graph(G) of Cities

- a. Create a Graph of N cities using Adjacency Matrix.
- b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

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

#define MAX_CITIES 20 // Maximum number of cities in the graph

// Function to initialize the adjacency matrix for the graph
void initializeGraph(int graph[MAX_CITIES][MAX_CITIES], int cities) {
    for (int i = 0; i < cities; i++) {
        for (int j = 0; j < cities; j++) {
            graph[i][j] = 0; // No edge between cities initially
        }
    }
}

// Function to add an edge between two cities in the adjacency matrix
void addEdge(int graph[MAX_CITIES][MAX_CITIES], int u, int v) {
    graph[u][v] = 1; // Directed edge from u to v
}

// Function to print the adjacency matrix of the graph
void printGraph(int graph[MAX_CITIES][MAX_CITIES], int cities) {
    printf("\nAdjacency Matrix of the Graph:\n");
    for (int i = 0; i < cities; i++) {
        for (int j = 0; j < cities; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}

// DFS function to print all nodes reachable from a starting node
void DFS(int graph[MAX_CITIES][MAX_CITIES], int visited[MAX_CITIES], int cities, int start) {
    printf("%d ", start);
    visited[start] = 1; // Mark the node as visited

    // Recur for all the vertices adjacent to this vertex
    for (int i = 0; i < cities; i++) {
        if (graph[start][i] == 1 && !visited[i]) {
            DFS(graph, visited, cities, i);
        }
    }
}
```

```

// BFS function using circular queue
void BFS(int graph[MAX_CITIES][MAX_CITIES], int visited[MAX_CITIES], int cities, int start) {
    int queue[MAX_CITIES];
    int front = 0, rear = 0;

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

    while (front != rear || visited[queue[rear]] == 1) {
        int currentNode = queue[front];
        front = (front + 1) % MAX_CITIES;
        printf("%d ", currentNode);

        for (int i = 0; i < cities; i++) {
            if (graph[currentNode][i] == 1 && !visited[i]) {
                rear = (rear + 1) % MAX_CITIES;
                queue[rear] = i;
                visited[i] = 1;
            }
        }
        if (front == (rear + 1) % MAX_CITIES) break; // Queue empty
    }
}

// Main function
int main() {
    int graph[MAX_CITIES][MAX_CITIES];
    int visited[MAX_CITIES];
    int cities, edges, u, v, start, choice;

    // Input number of cities
    do {
        printf("Enter number of cities (max %d): ", MAX_CITIES);
        scanf("%d", &cities);
        if (cities <= 0 || cities > MAX_CITIES) {
            printf("Invalid number of cities. Try again.\n");
        }
    } while (cities <= 0 || cities > MAX_CITIES);

    initializeGraph(graph, cities);

    // Input number of edges
    do {
        printf("Enter number of directed edges: ");
        scanf("%d", &edges);
        if (edges < 0 || edges > cities * (cities - 1)) {
            printf("Invalid number of edges. Try again.\n");
        }
    } while (edges < 0 || edges > cities * (cities - 1));
}

```

```

// Add edges
for (int i = 0; i < edges; i++) {
    do {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        if (u < 0 || u >= cities || v < 0 || v >= cities) {
            printf("Invalid edge! Node numbers should be between 0 and %d.\n", cities - 1);
        } else {
            addEdge(graph, u, v);
            break;
        }
    } while (1);
}

// Print adjacency matrix
printGraph(graph, cities);

// Menu-driven program
do {
    printf("\nMenu:\n");
    printf("1. Perform DFS (Depth First Search)\n");
    printf("2. Perform BFS (Breadth First Search)\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter starting node for DFS: ");
            scanf("%d", &start);
            if (start < 0 || start >= cities) {
                printf("Invalid starting node.\n");
                break;
            }
            for (int i = 0; i < cities; i++) visited[i] = 0;
            printf("DFS starting from node %d: ", start);
            DFS(graph, visited, cities, start);
            printf("\n");
            break;

        case 2:
            printf("Enter starting node for BFS: ");
            scanf("%d", &start);
            if (start < 0 || start >= cities) {
                printf("Invalid starting node.\n");
                break;
            }
            for (int i = 0; i < cities; i++) visited[i] = 0;
            printf("BFS starting from node %d: ", start);
    }
}

```

```
BFS(graph, visited, cities, start);
printf("\n");
break;

case 3:
printf("Exiting the program.\n");
break;

default:
printf("Invalid choice. Please try again.\n");
}

} while (choice != 3);

return 0;
}
```

Output:

```
Enter number of cities (max 20): 4
Enter number of directed edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 2
Enter edge (u v): 2 3
```

Adjacency Matrix of the Graph:

```
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
```

Menu:

1. Perform DFS (Depth First Search)
2. Perform BFS (Breadth First Search)
3. Exit

Enter your choice: 1

Enter starting node for DFS: 0

DFS starting from node 0: 0 1 2 3

Menu:

1. Perform DFS (Depth First Search)
2. Perform BFS (Breadth First Search)
3. Exit

Enter your choice: 2

Enter starting node for BFS: 0

BFS starting from node 0: 0 1 2 3

Menu:

1. Perform DFS (Depth First Search)

2. Perform BFS (Breadth First Search)

3. Exit

Enter your choice: 3

Exiting the program.

12. Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Develop a Program in C that uses Hash function H:

K → L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define MAX 100 // Maximum number of records in the hash table

// Structure to store employee record
struct Employee {
    int empID;          // Employee key (unique 4-digit integer)
    char name[50];      // Employee name
    char department[50]; // Department
    double salary;       // Salary
};

// Hash Table
struct Employee* hashTable[MAX];

// Hash function: H(K) = K mod m
int hashFunction(int key, int m) {
    return key % m;
}

// Function to create and initialize the hash table
void initHashTable(int m) {
    for (int i = 0; i < m; i++) {
        hashTable[i] = NULL;
    }
}

// Linear probing insertion function
void insert(int key, struct Employee emp, int m) {
    int idx = hashFunction(key, m);
    int startIdx = idx;

    // Linear probing: find the next available slot
    do {
        if (hashTable[idx] == NULL) break; // Found empty slot
        if (hashTable[idx]->empID == key) { // Duplicate key
            printf("Employee with ID %d already exists.\n", key);
            return;
        }
        idx++;
    } while (idx < m);
    hashTable[idx] = &emp;
}

// Function to search for an employee by key
struct Employee* search(int key, int m) {
    int idx = hashFunction(key, m);
    while (hashTable[idx] != NULL) {
        if (hashTable[idx]->empID == key) return hashTable[idx];
        idx++;
    }
    return NULL;
}
```

```

        }
        idx = (idx + 1) % m;
    } while (idx != startIdx);

if (hashTable[idx] != NULL) {
    printf("Hash table is full. Cannot insert more records.\n");
    return;
}

// Insert employee record
hashTable[idx] = (struct Employee*)malloc(sizeof(struct Employee));
hashTable[idx]->empID = emp.empID;
strcpy(hashTable[idx]->name, emp.name);
strcpy(hashTable[idx]->department, emp.department);
hashTable[idx]->salary = emp.salary;
}

// Function to search for an employee by key (empID)
struct Employee* search(int key, int m) {
    int idx = hashFunction(key, m);
    int startIdx = idx;

    while (hashTable[idx] != NULL) {
        if (hashTable[idx]->empID == key) {
            return hashTable[idx];
        }
        idx = (idx + 1) % m;
        if (idx == startIdx) break; // Full loop, key not found
    }
    return NULL; // Employee not found
}

// Function to display employee details
void display(struct Employee* emp) {
    if (emp != NULL) {
        printf("Employee ID: %d\n", emp->empID);
        printf("Name: %s\n", emp->name);
        printf("Department: %s\n", emp->department);
        printf("Salary: %.2lf\n", emp->salary);
    } else {
        printf("Employee not found.\n");
    }
}

int main() {
    int m, N;

    printf("Enter the size of hash table (m): ");
    scanf("%d", &m);
    if (m > MAX) {

```

```
printf("Maximum allowed size is %d. Setting m = %d\n", MAX, MAX);
m = MAX;
}

// Initialize hash table
initHashTable(m);

// Enter the number of employees
printf("Enter the number of employee records (N): ");
scanf("%d", &N);
if (N > m) {
    printf("Warning: Number of employees exceeds table size, some inserts may fail.\n");
}

// Input employee data
for (int i = 0; i < N; i++) {
    struct Employee emp;
    printf("\nEnter details for Employee %d:\n", i + 1);

    printf("Employee ID (4-digit key): ");
    scanf("%d", &emp.empID);
    getchar(); // Consume newline character

    printf("Employee Name: ");
    fgets(emp.name, 50, stdin);
    emp.name[strcspn(emp.name, "\n")] = '\0';

    printf("Department: ");
    fgets(emp.department, 50, stdin);
    emp.department[strcspn(emp.department, "\n")] = '\0';

    printf("Salary: ");
    scanf("%lf", &emp.salary);

    // Insert employee into hash table
    insert(emp.empID, emp, m);
}

// Search for an employee by key
int searchKey;
printf("\nEnter Employee ID (key) to search: ");
scanf("%d", &searchKey);

struct Employee* foundEmp = search(searchKey, m);
printf("\nEmployee Search Result:\n");
display(foundEmp);

// Free allocated memory
for (int i = 0; i < m; i++) {
    if (hashTable[i] != NULL) free(hashTable[i]);
```

```
    }  
  
    return 0;  
}
```

Output:

Enter the size of hash table (m): 10
Enter the number of employee records (N): 3

Enter details for Employee 1:
Employee ID (4-digit key): 1001
Employee Name: Alice
Department: HR
Salary: 50000

Enter details for Employee 2:
Employee ID (4-digit key): 1002
Employee Name: Bob
Department: IT
Salary: 60000

Enter details for Employee 3:
Employee ID (4-digit key): 1011
Employee Name: Charlie
Department: Finance
Salary: 55000

Enter Employee ID (key) to search: 1002

Employee Search Result:
Employee ID: 1002
Name: Bob
Department: IT
Salary: 60000.00