RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Affiliated to VTU, Recognized by GOK, Approved by AICTE, New Delhi
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

DEPARTMENT OF CSE (DATA SCIENCE)

DATA STRUCTURES LAB MANUAL

(BCSL305)

(As per Visvesvaraya Technological University Course type- PCCL)

Compiled by

DEPARTMENT OF CSE (DATA SCIENCE)

R N S Institute of Technology

Bengaluru-98

Name: _____

USN: _____

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Affiliated to VTU, Recognized by GOK, Approved by AICTE, New Delhi
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Estd : 2001

DEPARTMENT OF CSE (DATA SCIENCE)

VISION OF THE DEPARTMENT

Empowering students to solve complex real-time computing problems involving high volume multi-dimensional data.

MISSION OF THE DEPARTMENT

- Provide quality education in both theoretical and applied Computer Science to solve real world problems.
- Conduct research to develop algorithms that solve complex problems involving multi-dimensional high volume data through intelligent inferencing.
- Develop good linkages with industry and research organizations to expose students to global problems and find optimal solutions.
- Creating confident Graduates who can contribute to the nation through high levels of commitment following ethical practices and with integrity.

Disclaimer

Trademark

Estd : 2001

Edition: 2023- 24

Document Owner

The primary contact for questions regarding this document is:

Author(s):	1. Dr. Mohan H S 2. Dr. Sreevidya R C 3. Prof. Sunil G L
Department:	**CSE (Data Science)**
Contact email ids :	hod.datascience@rnsit.ac.in sreevidya.rc@rnsit.ac.in sunil.gl@rnsit.ac.in

COURSE OUTCOMES

Course Outcomes: At the end of this course, students are able to:
CO1- Analyze various linear and non-linear data structures
CO2- Demonstrate the working nature of different types of data structures and their applications
CO3- Use appropriate searching and sorting algorithms for the given scenarios
CO4- Apply the appropriate data structure for solving real-world problems

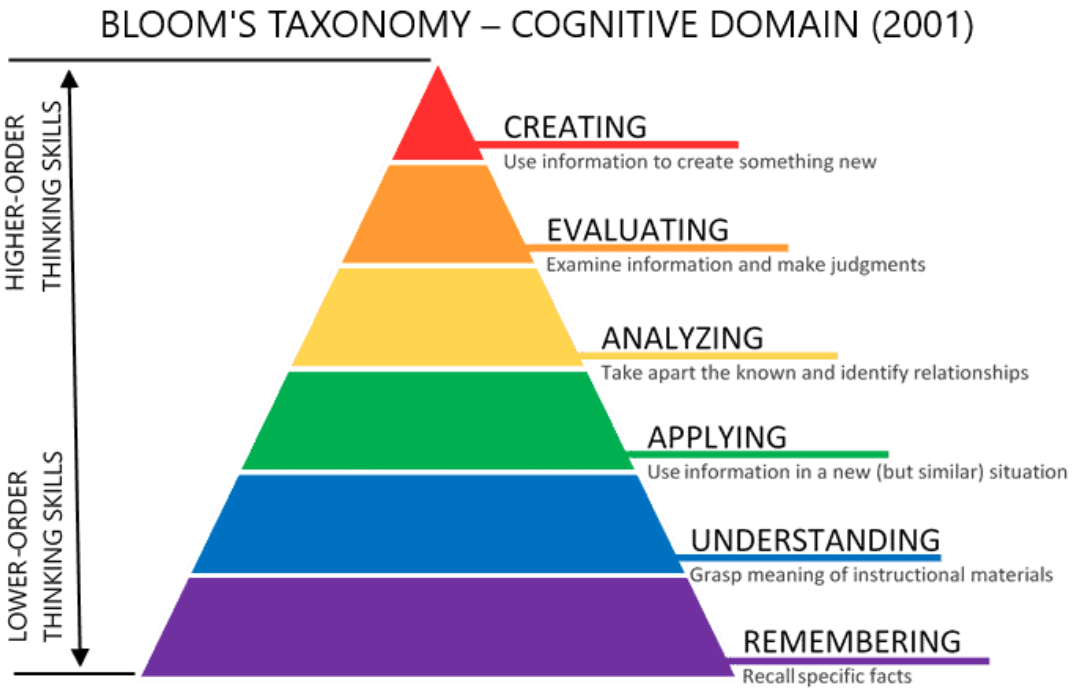
COs and POs Mapping of lab Component

COURSE OUTCOMES	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	3	3	3	3	3	2	2	2				2				
CO2	3	3	3	3	3	2	2	2				2				
CO3	3	3	3	3	3	2	2	2				2				
CO4	3	3	3	3	3	2	2	2				2				

Mapping of 'Graduate Attributes' (GAs) and 'Program Outcomes' (POs)

Graduate Attributes (GAs) (As per Washington Accord Accreditation)	Program Outcomes (POs) (As per NBA New Delhi)
Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
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.
Design/Development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety and the cultural, societal and environmental consideration.
Conduct Investigation 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.
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.
The engineer and society	Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequential responsibilities relevant to the professional engineering practice.
Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental context and demonstrate the knowledge of and need for sustainable development.
Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
Individual and team work	Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
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.
Project management & finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones won work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
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.

REVISED BLOOMS TAXONOMY (RBT)



PROGRAM LIST

Program 1:

AIM: 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.

Source code:

```
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# define DAYS 3
struct activity
{
        char *nday; int dday; char *desc;
};
typedef struct activity Plan; Plan* create();
void read(Plan *);
void display(Plan *);

int main()
{
        Plan *cal = NULL; cal = create( );
        read(cal);
        display(cal);
}

Plan* create()
{
        Plan *t = (Plan *) malloc(sizeof(Plan)*7);
        if (t == NULL)
        {
                printf("Sufficient memory not allocated\n"); return 0;
        }
        return t;
}

void read(Plan *p)
{

        int i;
```

```
        for(i=0;i<DAYS;i++)
        {
                p[i].nday = (char *) malloc(9);
                printf("Enter name of the day ");
                scanf(" %s",p[i].nday);
                printf("Enter date of the day ");
                scanf("%d",&(p[i].dday));
                printf("Enter description of the activity ");
                p[i].desc = (char *) malloc(400);
                scanf(" %[^\n]",p[i].desc);
                p[i].desc = (char *) realloc(p[i].desc,strlen(p[i].desc)+1);

        }
}

void display(Plan *p)
{
        int i;
        printf("**** Activity description for %d days ****\n",DAYS);
         for(i=0;i<DAYS;i++)
        {
                printf("\nName of the day : %s", p[i].nday);
                printf("\nDate of the day : %d", p[i].dday);
                printf("\nActivity description %s: ", p[i].desc);
        }
}
```

Sample Output:

```
C:\DS Lab Programs>gcc 1.c

C:\DS Lab Programs>a
Enter name of the day Monday
Enter date of the day 4
Enter description of the activity Assignment
Enter name of the day Wednesday
Enter date of the day 6
Enter description of the activity AICTE
Enter name of the day Tuesday
Enter date of the day 5
Enter description of the activity Project
**** Activity description for 3 days ****

Name of the day : Monday
Date of the day : 4
Activity description Assignment:
Name of the day : Wednesday
Date of the day : 6
Activity description AICTE:
Name of the day : Tuesday
Date of the day : 5
Activity description Project:
```

Program 2:

AIM: 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.

Source Code:

```
#include <stdio.h>
// Function to calculate the length of a string
int stringLength(const char *str) {
    int len = 0;
    while (str[len] != '\0') {
        len++;
    }
    return len;
}
// Function to check if pattern PAT exists in main string STR
int patternExists(const char *str, const char *pattern) {
    int i = 0, j = 0;

    while (str[i] != '\0') {
        j = 0;
        while (pattern[j] != '\0' && str[i + j] == pattern[j]) {
            j++;
        }
        if (pattern[j] == '\0') {
            return i; // Pattern found at index i in STR
        }
        i++;
    }
    return -1; // Pattern not found in STR
}

// Function to perform pattern matching and replacement
void findAndReplace(char *str, const char *pattern, const char *replace) {
    int patLen = stringLength(pattern);
    int repLen = stringLength(replace);
    int strLen = stringLength(str);

    int index = patternExists(str, pattern);
    while (index != -1) {
            // Copying replace string at the position of the pattern

        for (int i = 0; i < repLen; i++) {
            str[index + i] = replace[i];
        }
```

```
        strLen = stringLength(str);
        index = patternExists(str, pattern);
    }

    if (index == -1) {
        printf("Pattern not found in the main string.\n");
    }
}

int main() {
    char mainString[100], pattern[50], replace[50];

    // Input main string, pattern, and replace string
    printf("Enter the main string: \n");
    scanf(" %s", mainString); // Limit input to 99 characters to avoid buffer overflow
    printf("Enter the pattern string:\n ");
    scanf(" %s", pattern); // Limit input to 49 characters to avoid buffer overflow
    printf("Enter the replace string: \n");
    scanf(" %s", replace); // Limit input to 49 characters to avoid buffer overflow
    // Perform pattern matching and replacement
    findAndReplace(mainString, pattern, replace);
    // Output the modified main string after replacement
    printf("Modified String: %s\n", mainString);
    return 0;
}
```

Sample Output:

```
C:\DS Lab Programs>gcc Lab2.c

C:\DS Lab Programs>a
Enter the main string:
rnsitrnsitrnsit
Enter the pattern string:
 r
Enter the replace string:
R
Pattern not found in the main string.
Modified String: RnsitRnsitRnsit
```

Program 3:

AIM: 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

Source Code:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
// Define a stack data structure
struct Stack {
    int data[MAX];
    int top;
};
// Function to initialize the stack
void initialize(struct Stack *stack) {
    stack->top = -1;
}
// Function to push an element onto the stack
void push(struct Stack *stack, int value) {
    if (stack->top == MAX - 1) {
        printf("Stack Overflow: Cannot push element onto a full stack.\n");
    } else {
        stack->data[++stack->top] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
}

// Function to pop an element from the stack
int pop(struct Stack *stack) {
    if (stack->top == -1) {
        printf("Stack Underflow: Cannot pop element from an empty stack.\n");
        return -1; // Return a sentinel value to indicate underflow
    } else {
        return stack->data[stack->top--];
    }
}

// Function to check if a string is a palindrome using a stack
```

```
bool isPalindrome(const char *str)

 {
    struct Stack stack;
    initialize(&stack);

    int i = 0;
    while (str[i] != '\0') {
        push(&stack, str[i]);
        i++;
    }

    i = 0;
    while (str[i] != '\0') {
        if (str[i] != pop(&stack)) {
            return false;
        }
        i++;
    }

    return true;
}

// Function to display the stack's status
void displayStack(const struct Stack *stack) {
    printf("Stack Contents:\n");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d\n", stack->data[i]);
    }
}

int main() {
    struct Stack stack;
    initialize(&stack);

    int choice, element;
    char str[MAX];

    while (1) {
        printf("\nStack Operations Menu:\n");
        printf("1. Push Element\n");
        printf("2. Pop Element\n");
        printf("3. Check Palindrome\n");
        printf("4. Display Stack\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter an element to push onto the stack: ");
                scanf("%d", &element);
                push(&stack, element);
                break;
            case 2:
                element = pop(&stack);
                if (element != -1) {
                    printf("Popped %d from the stack.\n", element);
                }
                break;
            case 3:
                printf("Enter a string to check if it's a palindrome: ");
                scanf(" %[^\n]", str);
                if (isPalindrome(str)) {
                    printf("The string is a palindrome.\n");
                } else {
                    printf("The string is not a palindrome.\n");
                }
                break;
            case 4:
                displayStack(&stack);
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }

    return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>gcc lab3.c

C:\DSA Lab Program>a

Stack Operations Menu:
1. Push Element
2. Pop Element
3. Check Palindrome
4. Display Stack
5. Exit
Enter your choice: 1
Enter an element to push onto the stack: 10
Pushed 10 onto the stack.

Stack Operations Menu:
1. Push Element
2. Pop Element
3. Check Palindrome
4. Display Stack
5. Exit
Enter your choice: 1
Enter an element to push onto the stack: 20
Pushed 20 onto the stack.
```

```
Stack Operations Menu:
1. Push Element
2. Pop Element
3. Check Palindrome
4. Display Stack
5. Exit
Enter your choice: 4
Stack Contents:
10
20
```

Program 4:

AIM: 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.

Source Code:

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

#define MAX_STACK_SIZE 100

// Structure to represent a stack
struct Stack {
    int top;
    char items[MAX_STACK_SIZE];
};

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

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

// Function to push an element onto the stack
void push(struct Stack *stack, char item) {
    if (stack->top == MAX_STACK_SIZE - 1) {
        printf("Stack overflow!\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to pop an element from the stack
char pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow!\n");
        exit(1);
    }
    return stack->items[stack->top--];
}
```

```
// Function to check if a character is an operator
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%' || c == '^');
}

// Function to get the precedence of an operator
int getPrecedence(char c) {
    if (c == '^') return 3;
    if (c == '*' || c == '/' || c == '%') return 2;
    if (c == '+' || c == '-') return 1;
    return 0;
}

// Function to convert an infix expression to a postfix expression
void infixToPostfix(char *infix, char *postfix) {
    struct Stack stack;
    initialize(&stack);

    int i, j;
    i = j = 0;

    while (infix[i] != '\0') {
        char current = infix[i];

        if (isalnum(current)) {
            postfix[j++] = current;
        } else if (current == '(') {
            push(&stack, current);
        } else if (current == ')') {
            while (!isEmpty(&stack) && stack.items[stack.top] != '(') {
                postfix[j++] = pop(&stack);
            }
            if (!isEmpty(&stack) && stack.items[stack.top] == '(') {
                pop(&stack); // Pop the opening parenthesis
            }
        } else if (isOperator(current)) {
            while (!isEmpty(&stack) && getPrecedence(current) <=
getPrecedence(stack.items[stack.top])) {
                postfix[j++] = pop(&stack);
            }
            push(&stack, current);
        }
        i++;
    }

    while (!isEmpty(&stack)) {
        postfix[j++] = pop(&stack);
    }
```

```
    postfix[j] = '\0';
}

int main() {
    char infix[100];
    char postfix[100];

    printf("Enter an infix expression: ");
    scanf(" %[^\n]", infix);

    infixToPostfix(infix, postfix);

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

    return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>a
Enter an infix expression: (A+B*C)
Postfix expression: ABC*+

C:\DSA Lab Program>a
Enter an infix expression: (A/B)+(C*D)
Postfix expression: AB/CD*+
```

Proram 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 disk.

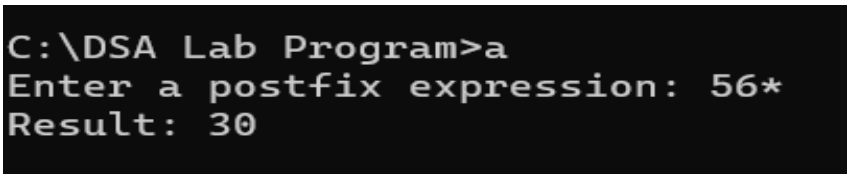
Source Code:

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

#define MAX_STACK_SIZE 100

// Structure to represent a stack
struct Stack {
    int top;
    int items [MAX_STACK_SIZE];
};

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

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

// Function to push an element onto the stack
void push(struct Stack *stack, int item) {
    if (stack->top == MAX_STACK_SIZE - 1) {
        printf("Stack overflow!\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to pop an element from the stack
int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow!\n");
        exit(1);
    }
    return stack->items[stack->top--];
}
```

```
// Function to evaluate a postfix expression

int evaluatePostfix(char *expression) {
    struct Stack stack;
    initialize(&stack);

    int len = strlen(expression);

    for (int i = 0; i < len; i++) {
        char current = expression[i];

        if (isdigit(current)) {
            push(&stack, current - '0');
        } else {
            int operand2 = pop(&stack);
            int operand1 = pop(&stack);
            int result;

            switch (current) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    result = operand1 / operand2;
                    break;
                case '%':
                    result = operand1 % operand2;
                    break;
                case '^':
                    result = 1;
                    for (int j = 0; j < operand2; j++) {
                        result *= operand1;
                    }
                    break;
                default:
                    printf("Invalid operator in expression.\n");
                    exit(1);
            }

            push(&stack, result);
        }
    }
```

```
    if (stack.top == 0) {

        return stack.items[0];

    }
 else {
        printf("Invalid expression.\n");
        exit(1);
    }
}

int main() {
    char expression[100];

    printf("Enter a postfix expression: ");
    scanf(" %[^\n]", expression);

    int result = evaluatePostfix(expression);

    printf("Result: %d\n", result);

    return 0;
}
```

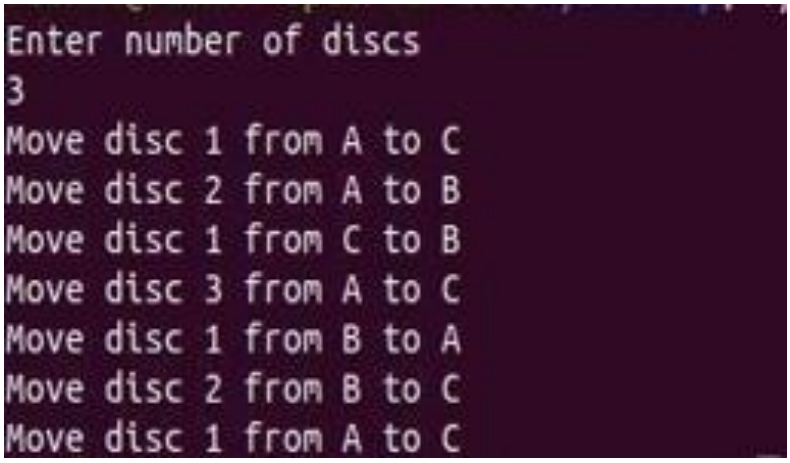
Sample Output:



5b. Solving Tower of Hanoi problem with n disks

```
# include <stdio.h>
void hanoi (int n, char S, char T, char D)
{
        if (n==0) return;
                hanoi(n-1, S, D, T);
                printf("Move disc %d from %c to %c\n", n, S, D); hanoi(n-1, T, S, D);
}
int main( )
{
        int n;
        printf("Enter number of discs\n");
        scanf("%d",&n);
        hanoi(n , 'A', 'B', 'C');
}
```

Sample Output:



Program 6:

AIM: 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

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define Max 3
void insert(char [], int *, int *); void del(char [], int *, int *);
void display(char [], int, int);
int main()
{
        char q[Max];
        int r=-1,f=0,cnt=0; int ch;
        while(1)
        {
                printf("1: Insert\n2: Delete\n3: Display\n4: Exit\n");
                printf("Enter choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: insert(q,&r,&cnt);
                        break;
                        case 2: del(q,&f,&cnt);
                         break;
                        case 3: display(q,f,cnt);
                        break;
                        default: exit(0);
                }

        }
}

void insert(char q[],int *r,int *cnt)
{
        char ele;
        if((*cnt) == Max)
        {
                printf("C Q overflow\n"); return;
```

```
        }
        (*r) = ((*r)+ 1 ) % Max;
        printf("enter the ele\n");
        scanf(" %c", &ele); q[*r]=ele;
        (*cnt)++;
        }

void del(char q[],int *f,int *cnt)
{
        if((*cnt) == 0)
        {
        printf("C Queue is empty\n"); return;}
        printf("Element deleted from circular queue is %c\n",q[(*f)]);
        (*f) = ((*f) + 1) % Max;
        (*cnt)--;
}

void display(char q[], int f, int cnt)
{
        int i,j;
        if (cnt==0)
        {
                printf("Circular Queue is empty\n"); return;} printf("Circular Queue contents are\n");
                for(i=f,j=0;j<cnt;j++)
        {
                printf("%d : %c\n",i,q[i]); i = (i + 1) % Max;
      }
}
```

Sample Output:

```
C:\DS Lab Programs>gcc 6.c

C:\DS Lab Programs>a
1: Insert
2: Delete
3: Display
4: Exit
Enter choice
1
enter the ele
9
1: Insert
2: Delete
3: Display
4: Exit
Enter choice
1
enter the ele
8
1: Insert
2: Delete
3: Display
4: Exit
Enter choice
3
Circular Queue contents are
0 : 9
1 : 8
```

Program 7:

AIM: 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

Source Code:

```
 #include<stdio.h>
 #include<stdlib.h>
typedef struct node
{
        char USN[10],
        name[20],
        branch[10];
        int sem;
        long int ph;
        struct node *link;
}nd;

nd* create(nd *);
void status(nd *);
nd* ins_front(nd *);
 nd* ins_rear(nd *);
nd* del_front(nd *);
 nd* del_rear(nd *);
void display(nd *);

int main()
{
nd * first = NULL; int ch;
for(;;)
{
printf("1. Create N students\n 2. Status of SLL\n");
printf("3. Insert front\n4. Insert rear\n5. Delete front\n");
printf("6. Delete rear\n7. Display\n8. Exit\n");
scanf("%d", &ch);
switch(ch)
{
        case 1: first = create(first);
```

```
            break;
            case 2: status(first);
            break;
            case 3: first = ins_front(first);
            break;
            case 4: first = ins_rear(first);
            break;
            case 5: first = del_front(first);
             break;
            case 6: first = del_rear(first);
             break;
            case 7: display(first);
            break;
            case 8:
            exit(0);
            }
  }
}

nd * del_front(nd *f)
{
nd *t;
if (f==NULL)
{
            printf("SLL is empty\n");
            return NULL;
}
printf("Information to be deleted is...\n");
printf("%s\t%s\t%s\t%d\t%ld\n",(f->USN),(f->name),(f->branch),(f->sem),(f->ph));
t = f->link; free(f); return t;
}

nd * del_rear(nd *f)
{
            nd *t,*p;
            if (f==NULL) {
            printf("SLL is empty\n"); return NULL;
}
for(p=NULL,t=f;t->link!=NULL;p=t,t=t->link);
printf("Information to be deleted is...\n");
printf("%s\t%s\t%s\t%d\t%ld\n",(t->USN),(t->name),(t->branch),(t->sem),(t->ph));
free(t);
if (p!=NULL)
{
            p->link=NULL;
             return f;
}
```

```
else
        return NULL;
}
nd * ins_rear(nd * f)
{
        nd *p=f;
        nd *t=(nd*)malloc(sizeof(nd));
         t->link=NULL;
        printf("Enter USN, Name, Branch, Sem and Phone of the student:\n");
        scanf("%s%s%s%d%ld",(t->USN),(t->name),(t->branch), &(t->sem), &(t->ph));
if (f==NULL)
return t;
for(;p->link!=NULL; p=p->link);
         p->link=t;
        return f;
}

void status(nd *f)
{
int cnt=0;
if (f==NULL)
{
printf("SLL is empty\n"); return;
}

for(;f!=NULL;f=f->link,cnt++); printf("Number of nodes in SLL is %d\n",cnt);
}

nd* create(nd *f)
{
int n,i;
printf("Enter value for n\n");
scanf("%d",&n); for(i=0;i<n;i++)
f = ins_front(f); return f; }
nd* ins_front(nd *f)
{
nd *t=(nd*)malloc(sizeof(nd));
printf("Enter USN, Name, Branch, Sem and Phone of the student:\n");
scanf("%s%s%s%d%ld",(t->USN),(t->name),(t->branch), &(t->sem), &(t->ph));

t->link = f;
 return t;
}
void display(nd *f)
{
        if (f==NULL)
        {
                printf("Contents of SLL are empty\n"); return;
```

```
        }
        printf("Contents of the list\n");
        while(f!=NULL)
        {
                printf("%s\t%s\t%s\t%d\t%ld\n",(f->USN),(f->name),(f->branch),(f->sem),(f->ph));     f =
                f->link;
        }
}
```

Sample Output:

```
C:\DS Lab Programs>gcc 7.c

C:\DS Lab Programs>a
1. Create N students
2. Status of SLL
3. Insert front
4. Insert rear
5. Delete front
6. Delete rear
7. Display
8. Exit
1
Enter value for n
2
Enter USN, Name, Branch, Sem and Phone of the student:
1RN22
ALEX
ABCD
3
900400
Enter USN, Name, Branch, Sem and Phone of the student:
1RN21
JOSH
XYZ
3
98765
1. Create N students
2. Status of SLL
3. Insert front
4. Insert rear
5. Delete front
6. Delete rear
7. Display
8. Exit
7
Contents of the list
1RN21   JOSH    XYZ     3       98765
1RN22   ALEX    ABCD    3       900400
```

Program 8:

AIM: 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

Source code:

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

// Structure to represent an employee
struct Employee {
    char SSN[15];
    char Name[50];
    char Dept[50];
    char Designation[50];
    float Sal;
    char PhNo[15];
    struct Employee *prev;
    struct Employee *next;
};

// Function to create a new employee node
struct Employee *createEmployee() {
    struct Employee *newEmployee = (struct Employee *)malloc(sizeof(struct
Employee));
    if (newEmployee == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    printf("Enter Employee Details:\n");
    printf("SSN: ");
    scanf(" %[^\n]", newEmployee->SSN);
    printf("Name: ");
    scanf(" %[^\n]", newEmployee->Name);
    printf("Department: ");
    scanf(" %[^\n]", newEmployee->Dept);
    printf("Designation: ");
    scanf(" %[^\n]", newEmployee->Designation);
    printf("Salary: ");
    scanf("%f", &newEmployee->Sal);
```

```
    printf("Phone Number: ");
    scanf(" %[^\n]", newEmployee->PhNo);

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

// Function to display an employee's details
void displayEmployee(struct Employee *employee) {
    printf("SSN: %s\n", employee->SSN);
    printf("Name: %s\n", employee->Name);
    printf("Department: %s\n", employee->Dept);
    printf("Designation: %s\n", employee->Designation);
    printf("Salary: %.2f\n", employee->Sal);
    printf("Phone Number: %s\n", employee->PhNo);
}

// Function to create a doubly linked list of employees with end insertion
struct Employee *createDLL(struct Employee *head, int N) {
    for (int i = 0; i < N; i++) {
        struct Employee *newEmployee = createEmployee();
        if (head == NULL) {
            head = newEmployee;
        } else {
            struct Employee *current = head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newEmployee;
            newEmployee->prev = current;
        }
        printf("Employee added to the end of the list.\n");
    }
    return head;
}

// Function to display the status of the DLL and count the number of nodes
void displayDLL(struct Employee *head) {
    struct Employee *current = head;
    int count = 0;

    if (current == NULL) {
        printf("Doubly Linked List is empty.\n");
    } else {
        printf("Doubly Linked List Contents:\n");
        while (current != NULL) {
```

```
            displayEmployee(current);

            current = current->next;
            count++;
        }
        printf("Total number of nodes: %d\n", count);
    }
}

// Function to insert an employee at the end of the DLL
struct Employee *insertEnd(struct Employee *head) {
    struct Employee *newEmployee = createEmployee();

    if (head == NULL) {
        head = newEmployee;
    } else {
        struct Employee *current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newEmployee;
        newEmployee->prev = current;
    }
    printf("Employee added to the end of the list.\n");
    return head;
}

// Function to delete an employee from the front of the DLL
struct Employee *deleteFront(struct Employee *head) {
    if (head == NULL) {
        printf("Doubly Linked List is empty. Nothing to delete.\n");
    } else {
        struct Employee *temp = head;
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        printf("Employee deleted from the front of the list.\n");
    }
    return head;
}

// Function to demonstrate how the DLL can be used as a double-ended queue
struct Employee *enqueueFront(struct Employee *head) {
    head = insertEnd(head); // Enqueue at the end to simulate enqueue at the front
    return head;
}
```

```
struct Employee *dequeueEnd(struct Employee *head) {
    head = deleteFront(head); // Dequeue from the front to simulate dequeue at the end

    return head;
}

int main() {
    struct Employee *head = NULL;
    int choice, N;

    while (1) {
        printf("\nDoubly Linked List Operations Menu:\n");
        printf("1. Create DLL of N Employees Data by End Insertion\n");
        printf("2. Display DLL and Count Nodes\n");
        printf("3. Insert Employee at End\n");
        printf("4. Delete Employee from 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 (N): ");
                scanf("%d", &N);
                head = createDLL(head, N);
                break;
            case 2:
                displayDLL(head);
                break;
            case 3:
                head = insertEnd(head);
                break;
            case 4:
                head = deleteFront(head);
                break;
            case 5:
                printf("Demonstrating DLL as a Double-Ended Queue:\n");
                printf("1. Enqueue at Front\n");
                printf("2. Dequeue at End\n");
                printf("3. Exit\n");
                printf("Enter your choice: ");
                scanf("%d", &choice);
                switch (choice) {
                    case 1:
                        head = enqueueFront(head);
                        break;
```

```
                case 2:

                    head = dequeueEnd(head);
                    break;
                case 3:

                    break;
                default:
                    printf("Invalid choice. Please select a valid option.\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice. Please select a valid option.\n");
    }
  }

  return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>gcc lab8.c

C:\DSA Lab Program>a

Doubly Linked List Operations Menu:
1. Create DLL of N Employees Data by End Insertion
2. Display DLL and Count Nodes
3. Insert Employee at End
4. Delete Employee from Front
5. Demonstrate DLL as Double-Ended Queue
6. Exit
Enter your choice: 1
Enter the number of employees (N): 2
Enter Employee Details:
SSN: 12
Name: ALEX
Department: XYZ
Designation: Asst. Professor
Salary: 20000
Phone Number: 984567
Employee added to the end of the list.
Enter Employee Details:
SSN: 333
Name: JOHN
Department: ABC
Designation: Professor
Salary: 450000
Phone Number: 98765
Employee added to the end of the list.
```

```
Doubly Linked List Operations Menu:
1. Create DLL of N Employees Data by End Insertion
2. Display DLL and Count Nodes
3. Insert Employee at End
4. Delete Employee from Front
5. Demonstrate DLL as Double-Ended Queue
6. Exit
Enter your choice: 2
Doubly Linked List Contents:
SSN: 12
Name: ALEX
Department: XYZ
Designation: Asst. Professor
Salary: 20000.00
Phone Number: 984567
SSN: 333
Name: JOHN
Department: ABC
Designation: Professor
Salary: 450000.00
Phone Number: 98765
Total number of nodes: 2
```

Program 9:

AIM: Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes

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

Source Code:

```
#include<stdio.h>
#include<malloc.h>
#include<math.h>
struct node
{
        int cf, px, py, pz;
int flag;
        struct node *link;
};
typedef struct node NODE;

NODE* getnode()
{
        NODE *x;
        x=(NODE*)malloc(sizeof(NODE));
        if(x==NULL)
        {
                printf("Insufficient memory\n");
                exit(0);
        }
        return x;
}
void display(NODE *head)
{

        NODE *temp;
        if(head->link==head)
        {
                printf("Polynomial does not exist\n");
                return;
        }
        temp=head->link;
        printf("\n");
        while(temp!=head)
        {
                printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);
                if(temp->link != head)
```

```
                printf(" + ");
                temp=temp->link;
        }
        printf("\n");
}
NODE* insert_rear(int cf,int x,int y,int z,NODE *head)
{

        NODE *temp,*cur;
        temp=getnode();
        temp->cf=cf;
        temp->px=x;
        temp->py=y;
        temp->pz=z;
        temp->flag=0;
        cur=head->link;
        while(cur->link!=head)
        {
                cur=cur->link;
        }
        cur->link=temp;
        temp->link=head;
        return head;
}

NODE* read_poly(NODE *head)
{
        int px, py, pz, cf;
        int ch;
        printf("\nEnter coeff: ");
        scanf("%d",&cf);
        printf("\nEnter x, y, z powers(0-indiacate NO term): ");
scanf("%d%d%d", &px, &py, &pz);
        head=insert_rear(cf,px,py,pz,head);
        printf("\nIf you wish to continue press 1 otherwise 0: ");
        scanf("%d",&ch);
        while(ch!=0)
        {
                printf("\nEnter coeff: ");
                scanf("%d",&cf);
                printf("\nEnter x, y, z powers(0-indiacate NO term): ");
                scanf("%d%d%d", &px, &py, &pz); head=insert_rear(cf,px,py,pz,head);
                printf("\nIf you wish to continue press 1 otherwise 0: ");
                scanf("%d", &ch);
        }
        return head;
}
```

```
NODE* add_poly(NODE *h1,NODE *h2,NODE *h3)
{
        NODE *p1,*p2;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        p1=h1->link;
        while(p1!=h1)
        {
                x1=p1->px;
                y1=p1->py;
                z1=p1->pz;
                cf1=p1->cf;
                p2=h2->link;
                while(p2!=h2)
                {
                        x2=p2->px;
                        y2=p2->py;
                        z2=p2->pz;
                        cf2=p2->cf;
                        if(x1==x2 && y1==y2 && z1==z2)
                        break;
                        p2=p2->link;
                }
                if(p2!=h2)
                {
                        cf=cf1+cf2;
                        p2->flag=1;
                        if(cf!=0)
                        h3=insert_rear(cf,x1,y1,z1,h3);
                }
                else
                        h3=insert_rear(cf1,x1,y1,z1,h3);
                        p1=p1->link;
        }
        p2=h2->link;
        while(p2!=h2)
        {
                if(p2->flag==0)
                h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
                p2=p2->link;
        }
        return h3;
}
void evaluate(NODE *he)
{
        NODE *head;
        int x, y, z;
        float result=0.0;
        head=he;
```

```
        printf("\nEnter x, y, z, terms to evaluate:\n");
        scanf("%d%d%d", &x, &y, &z);
        he=he->link;
        while(he != head)
        {
                result = result + (he->cf * pow(x,he->px) * pow(y,he->py) * pow(z,he-
>pz));
                he=he->link;
        }
        printf("\nPolynomial result is: %f", result);
}

void main()
{

        NODE *h1,*h2,*h3,*he;
        int ch;
        while(1)
        {
                printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");
                printf("Enter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: he=getnode();
                                he->link=he;
printf("\nEnter polynomial to evaluate:\n");
                                he=read_poly(he);
                                display(he);
                                evaluate(he);
                                free(he);
                                break;
                        case 2: h1=getnode();
                                h2=getnode();
        h3=getnode();
        h1->link=h1;
        h2->link=h2;
        h3->link=h3;
printf("\nEnter the first polynomial:");
                                h1=read_poly(h1);
                                printf("\nEnter the second polynomial:");
                                h2=read_poly(h2);
                                h3=add_poly(h1,h2,h3);
                                printf("\nFirst polynomial is: ");
                                display(h1);
                                printf("\nSecond polynomial is: ");
                                display(h2);
                                printf("\nThe sum of 2 polynomials is: ");
```

```
        display(h3);
                                break;

                        case 3:exit(0);
                                break;
                        default:printf("\nInvalid entry");
                                break;
                }

        }
        }
```

Sample Output:

```
C:\DS Lab Programs>gcc 9.c

C:\DS Lab Programs>a
1. Evaluate
2. Polynomial addition
3. Exit
Choice: 1
Enter number of terms
2
Enter coef, xexp, yexp and zexp of the term
6
2
2
1
Enter coef, xexp, yexp and zexp of the term
-4
0
1
5
Terms in polynomial are...
(-4x^0y^1z^5) +(6x^2y^2z^1) +
Enter value of x,y and z
1
2
3
Evaluation of polynomial is -1872
1. Evaluate
2. Polynomial addition
3. Exit
```

Program 10:

AIM: 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

Source Code:

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

// Structure to represent a node in the BST
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

// Function to create a new node
struct Node *createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the BST
struct Node *insertNode(struct 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 to traverse the BST in Inorder
```

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

// Function to traverse the BST in Preorder
void preorderTraversal(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function to traverse the BST in Postorder
void postorderTraversal(struct Node *root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to search for a key in the BST
int searchBST(struct Node *root, int key) {
    if (root == NULL) {
        return 0; // Key not found
    }

    if (root->data == key) {
        return 1; // Key found
    }

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

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

    // Create a BST with given values
```

```
int values[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
int numValues = sizeof(values) / sizeof(values[0]);
for (int i = 0; i < numValues; i++) {
   root = insertNode(root, values[i]);
}

while (1) {
   printf("\nBinary Search Tree (BST) Operations Menu:\n");
   printf("1. Inorder Traversal\n");
   printf("2. Preorder Traversal\n");
   printf("3. Postorder Traversal\n");
   printf("4. Search for a Key\n");
   printf("5. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
         printf("Inorder Traversal: ");
         inorderTraversal(root);
         printf("\n");
         break;
      case 2:
         printf("Preorder Traversal: ");
         preorderTraversal(root);
         printf("\n");
         break;
      case 3:
         printf("Postorder Traversal: ");
         postorderTraversal(root);
         printf("\n");
         break;
      case 4:
         printf("Enter the key to search: ");
         scanf("%d", &key);
         if (searchBST(root, key)) {
            printf("Key found in the BST.\n");
         } else {
            printf("Key not found in the BST.\n");
         }
         break;
      case 5:
         return 0;
      default:
         printf("Invalid choice. Please select a valid option.\n");
   }
}
```

```
    return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>gcc lab10.c

C:\DSA Lab Program>a

Binary Search Tree (BST) Operations Menu:
1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Search for a Key
5. Exit
Enter your choice: 1
Inorder Traversal: 2 5 6 7 8 9 14 15 24

Binary Search Tree (BST) Operations Menu:
1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Search for a Key
5. Exit
Enter your choice: 2
Preorder Traversal: 6 5 2 9 8 7 15 14 24

Binary Search Tree (BST) Operations Menu:
1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Search for a Key
5. Exit
Enter your choice: 3
Postorder Traversal: 2 5 7 8 14 24 15 9 6
```

Program 11:

AIM: 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

Source Code:

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

#define MAX_CITIES 20

// Structure to represent a graph
struct Graph {
    int vertices;
    int adjMatrix[MAX_CITIES][MAX_CITIES];
};

// Function to create a graph of cities
struct Graph createGraph(int N) {
    struct Graph graph;
    graph.vertices = N;

    // Initialize the adjacency matrix with all zeros
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            graph.adjMatrix[i][j] = 0;
        }
    }

    // Populate the adjacency matrix with connections between cities
    printf("Enter connections between cities (0 for no connection, 1 for connection):\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i != j) {
                printf("Is there a connection between city %d and city %d? (1/0): ", i + 1, j +
1);
                scanf("%d", &graph.adjMatrix[i][j]);
            }
        }
    }

    return graph;
}

// Function to perform Depth-First Search (DFS) from a starting node
```

```
void DFS(struct Graph graph, int start, int visited[]) {
    visited[start] = 1;

    printf("City %d is reachable.\n", start + 1);

    for (int i = 0; i < graph.vertices; i++) {
        if (graph.adjMatrix[start][i] == 1 && !visited[i]) {
            DFS(graph, i, visited);
        }
    }
}

// Function to print all nodes reachable from a given starting node using DFS
void printDFS(struct Graph graph, int start) {
    int visited[MAX_CITIES] = {0};
    printf("Nodes reachable from City %d using DFS:\n", start + 1);
    DFS(graph, start, visited);
}

// Function to perform Breadth-First Search (BFS) from a starting node
void BFS(struct Graph graph, int start, int visited[]) {
    int queue[MAX_CITIES];
    int front = -1, rear = -1;

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

    while (front <= rear) {
        int current = queue[front++];
        printf("City %d is reachable.\n", current + 1);

        for (int i = 0; i < graph.vertices; i++) {
            if (graph.adjMatrix[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}

// Function to print all nodes reachable from a given starting node using BFS
void printBFS(struct Graph graph, int start) {
    int visited[MAX_CITIES] = {0};
    printf("Nodes reachable from City %d using BFS:\n", start + 1);
    BFS(graph, start, visited);
}

int main() {
```

```
    int N, startCity;

    printf("Enter the number of cities: ");

    scanf("%d", &N);

    struct Graph cityGraph = createGraph(N);

    printf("Enter the starting city to explore: ");
    scanf("%d", &startCity);

    if (startCity > 0 && startCity <= N) {
        // Print reachable nodes using DFS
        printDFS(cityGraph, startCity - 1);

        // Print reachable nodes using BFS
        printBFS(cityGraph, startCity - 1);
    } else {
        printf("Invalid starting city. Please enter a valid city number.\n");
    }

    return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>gcc lab11.c

C:\DSA Lab Program>a
Enter the number of cities: 3
Enter connections between cities (0 for no connection, 1 for connection):
Is there a connection between city 1 and city 2? (1/0): 1
Is there a connection between city 1 and city 3? (1/0): 1
Is there a connection between city 2 and city 1? (1/0): 1
Is there a connection between city 2 and city 3? (1/0): 1
Is there a connection between city 3 and city 1? (1/0): 1
Is there a connection between city 3 and city 2? (1/0): 0
Enter the starting city to explore: 1
Nodes reachable from City 1 using DFS:
City 1 is reachable.
City 2 is reachable.
City 3 is reachable.
Nodes reachable from City 1 using BFS:
City 40 is reachable.
City 1 is reachable.
City 2 is reachable.
City 3 is reachable.
```

Program 12:

AIM: 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.

Source Code:

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

#define MAX_MEMORY_LOCATIONS 100
#define MAX_KEYS 100

// Structure to represent an employee record
struct EmployeeRecord {
    int key; // 4-digit key
    // Add other fields as needed
};

// Structure to represent a memory location in the hash table
struct MemoryLocation {
    int key;
    struct EmployeeRecord employee;
};

// Function to initialize the hash table with empty memory locations
void initializeHashTable(struct MemoryLocation hashTable[], int m) {
    for (int i = 0; i < m; i++) {
        hashTable[i].key = -1; // Empty memory location
    }
}

// Function to perform linear probing and insert an employee record into the hash
table
void insertEmployee(struct MemoryLocation hashTable[], int m, struct
EmployeeRecord employee) {
    int key = employee.key;
    int index = key % m; // Initial index using the hash function

    // Linear probing to resolve collisions
    while (hashTable[index].key != -1) {
        index = (index + 1) % m; // Move to the next location
    }
```

```
    // Insert the employee record at the empty memory location

    hashTable[index].key = key;
    hashTable[index].employee = employee;
}

// Function to display the contents of the hash table
void displayHashTable(struct MemoryLocation hashTable[], int m) {
    printf("\nHash Table Contents:\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i].key != -1) {
            printf("Location %d: Key: %d\n", i, hashTable[i].key);
            // You can print other employee record fields here
        }
    }
}

int main() {
    int m; // Number of memory locations in the hash table
    int n; // Number of employee records
    struct EmployeeRecord employeeRecords[MAX_KEYS];
    struct MemoryLocation hashTable[MAX_MEMORY_LOCATIONS];

    printf("Enter the number of memory locations (m) in the hash table: ");
    scanf("%d", &m);

    printf("Enter the number of employee records (n): ");
    scanf("%d", &n);

    if (m <= 0 || n <= 0) {
        printf("Invalid input. Please enter valid values for m and n.\n");
        return 1;
    }

    initializeHashTable(hashTable, m);

    printf("Enter the %d employee records (each record includes a 4-digit key):\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &employeeRecords[i].key);
        // You can add code to input other fields of the employee record here
        insertEmployee(hashTable, m, employeeRecords[i]);
    }

    displayHashTable(hashTable, m);

    return 0;
}
```

Sample Output:

```
C:\DSA Lab Program>a
Enter the number of memory locations (m) in the hash table: 3
Enter the number of employee records (n): 2
Enter the 2 employee records (each record includes a 4-digit key):
2345
3456

Hash Table Contents:
Location 0: Key: 3456
Location 2: Key: 2345
```

Sample Viva Questions:

1. **What is a data structure?**

A data structure is a way of organizing and storing data to perform operations efficiently.

2. **Differentiate between an array and a linked list.**

An array is a fixed-size, contiguous memory structure, while a linked list is a dynamic data structure where elements are linked using pointers.

3. **Explain the concept of time complexity.**

Time complexity represents the amount of time an algorithm takes to complete as a function of the input size.

4. **What is the difference between a stack and a queue?**

A stack follows the Last In, First Out (LIFO) principle, while a queue follows the First In, First Out (FIFO) principle.

5. **How does a binary search work?**

Binary search is a divide-and-conquer algorithm that efficiently finds the position of a target value within a sorted array.

6. **What is recursion, and how is it used in data structures?**

Recursion is a programming technique where a function calls itself. It is often used in data structures like trees and graphs.

7. **Explain the concept of a hash table.**

A hash table is a data structure that maps keys to values using a hash function, allowing for efficient retrieval and storage of data.

8. **What is the difference between a linked list and an array?**

Arrays have a fixed size, and their elements are stored in contiguous memory locations, while linked lists can dynamically change in size and store elements with non-contiguous memory.

9. **Define a tree in terms of data structures.**

A tree is a hierarchical data structure composed of nodes, each having a value and zero or more child nodes**.**

10. **Explain the process of tree traversal.**

Tree traversal involves visiting all the nodes of a tree in a specific order. Common methods are in-order, pre-order, and post-order traversals.

11. **What is a priority queue?**

A priority queue is a data structure that stores elements with associated priorities and allows for efficient retrieval of the element with the highest priority.

12. **Describe the concept of a graph.**

A graph is a collection of nodes (vertices) and edges that connect pairs of nodes. It can be directed or undirected.

13. What is dynamic programming, and how is it related to data structures?

Dynamic programming is a technique for solving complex problems by breaking them down into simpler overlapping subproblems. Data structures like arrays or tables are often used to store solutions to subproblems to avoid redundant computations.

14. What is the purpose of a doubly linked list?

In a doubly linked list, each node contains a data element and two pointers, one pointing to the next node and another pointing to the previous node. This allows for easy traversal in both directions.

15. How does a bubble sort algorithm work?

Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

16. What is a stack?

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle, where the last element added is the first one to be removed.

17. What are main operations of a stack.

The main operations are:

Push: Adds an element to the top of the stack.

Pop: Removes the element from the top of the stack.

Peek (or Top): Returns the element at the top of the stack without removing it.

18. How is a stack implemented?

A stack can be implemented using an array or a linked list. In the array implementation, a fixed-size array is used to store elements, and a variable keeps track of the top of the stack.

19. What is the significance of the top of the stack?

The top of the stack is the position where the next push or pop operation will occur. It keeps track of the most recently added element.

20. Explain the concept of stack overflow.

Stack overflow occurs when the stack reaches its maximum capacity, and a push operation tries to add an element beyond the available space, leading to a runtime error.

21. How is stack different from a queue?

A stack follows the Last In, First Out (LIFO) principle, while a queue follows the First In, First Out (FIFO) principle.

22. What is the purpose of a call stack in programming?

The call stack is used to manage function calls in a program. When a function is called, its local variables and the return address are pushed onto the stack. When the function completes, its stack frame is popped.

23. Can you give an example of a real-world scenario where a stack is useful?

One example is the Undo feature in software applications. Each action is pushed onto a stack, and the Undo operation pops the last action to revert the changes.

24. How does a stack help in reversing a string?

By pushing each character of the string onto the stack and then popping them in reverse order, the string can be effectively reversed.

25. Explain the use of parentheses matching using a stack.

A stack can be used to check if parentheses in an expression are balanced. When an opening parenthesis is encountered, it is pushed onto the stack. When a closing parenthesis is encountered, it is checked against the top of the stack.

26. What is the significance of the stack in recursive function calls?

The call stack is used to manage the execution of recursive functions by keeping track of each function call's local variables and return addresses.

27. How is an expression evaluation carried out using a stack?

In expression evaluation, a stack can be used to store operands and operators. The stack helps in maintaining the order of operations and ensures proper evaluation.

28. Explain the concept of a stack frame.

A stack frame is a block of memory on the call stack that stores local variables, function parameters, and the return address for a specific function call.

29. Can a stack be implemented using queues? How?

Yes, a stack can be implemented using two queues. Push operation is simulated by enqueueing elements into one queue, and pop operation is simulated by dequeuing elements from the other queue.

30. What is the role of the stack in managing recursive algorithms?

The stack is crucial in managing recursive algorithms as it keeps track of the function calls and their associated local variables, allowing for proper execution and backtracking.

31. What is a queue?

A queue is a linear data structure that follows the First In, First Out (FIFO) principle, where the first element added is the first one to be removed.

32. Explain the main operations of a queue.

The main operations are:

Enqueue: Adds an element to the rear (end) of the queue.

Dequeue: Removes the element from the front (beginning) of the queue.

Peek (or Front): Returns the element at the front of the queue without removing it.

33. How is a queue implemented?

A queue can be implemented using an array or a linked list. In the array implementation, a fixed-size array is used to store elements, and two pointers (front and rear) keep track of the queue's boundaries.

34. Differentiate between a queue and a stack.

A queue follows the First In, First Out (FIFO) principle, while a stack follows the Last In, First Out (LIFO) principle.

35. Explain the concept of circular queues.

In a circular queue, the front and rear pointers wrap around to the beginning of the queue when they reach the end, forming a circle. This allows for better space utilization.

36. What is the significance of the front and rear pointers in a queue?

The front pointer points to the first element in the queue, and the rear pointer points to the last element. They help in maintaining the order of elements and perform enqueue and dequeue operations efficiently.

37. What is a priority queue? How is it different from a regular queue?

A priority queue is a variation of a queue where each element has an associated priority. Elements with higher priority are dequeued before elements with lower priority.

38.Explain the concept of double-ended queues (deque).

A deque is a queue that allows insertion and deletion at both the front and rear ends. It supports the operations of both stacks and queues.

38. How can a queue be implemented using two stacks?

Two stacks can be used to implement a queue by using one stack for enqueue operations and the other for dequeue operations**.**

39. Explain the concept of a priority queue with a real-world example.

A real-world example of a priority queue is the scheduling of tasks in an operating system, where tasks with higher priority are scheduled to run before those with lower priority.

40. How can a queue be implemented using a linked list?

In a linked list implementation of a queue, each node contains data and a pointer to the next node. Enqueue and dequeue operations involve adding and removing nodes, respectively, at the front and rear of the linked list.

41. What is a linked list?

A linked list is a data structure that consists of a sequence of elements, where each element points to the next one in the sequence.

42. Explain the advantages of a linked list over an array.

Linked lists can dynamically change in size, require less memory allocation, and support efficient insertions and deletions compared to arrays.

43. What are the types of linked lists?

Types of linked lists include singly linked lists, doubly linked lists, and circular linked lists.

Differentiate between a singly linked list and a doubly linked list.

In a singly linked list, each node points to the next node, while in a doubly linked list, each node points to both the next and the previous nodes.

44. How is a circular linked list different from a linear linked list?

In a circular linked list, the last node points back to the first node, forming a circle, while a linear linked list has a distinct end.

45. Explain the concept of a node in a linked list.

A node is a fundamental building block of a linked list, containing data and a reference (or link) to the next node (and, in the case of a doubly linked list, to the previous node).

46. What is the significance of a head pointer in a linked list?

The head pointer points to the first node in the linked list, serving as the entry point for accessing and traversing the list.

47. Explain the process of inserting a node in the middle of a linked list.

To insert a node in the middle, the pointers of the preceding node are adjusted to point to the new node, and the new node is linked to the next node.

48. How is a node removed from a linked list?

To remove a node, the links of the preceding node are adjusted to bypass the node to be removed, and the memory for the removed node is deallocated.

49. What is a NULL pointer, and how is it used in linked lists?

A NULL pointer is a pointer that does not point to any memory location. In linked lists, it is used to signify the end of the list, as the last node's next (or previous) pointer points to NULL.

50. Explain the concept of a tail pointer in a linked list.

The tail pointer points to the last node in a linked list, making it faster to perform operations at the end of the list, such as insertions and deletions.

51. How does a linked list facilitate dynamic memory allocation?

Linked lists allow for dynamic memory allocation because nodes can be added or removed without the need for contiguous memory blocks.

52. How can a linked list be reversed?

To reverse a linked list, each node's next pointer is adjusted to point to the preceding node iteratively or recursively.

Additional Programs

Program 1:

AIM: Program to demonstrate dynamic memory allocation using malloc() or calloc(), realloc(), free() function.

Source Code:

```
#include<stdio.h>

#include<stdlib.h>

void main(){
	int *p;
	p = (int *)malloc(2*sizeof(int));
	if(p==NULL){
		printf("failed\n");
		exit(1);
	}
	p[0]=1;
	p[1]=2;

	p = (int *)realloc(p,3*sizeof(int));
	if(p==NULL){
		printf("failed\n");
		exit(1);
	}
	p[2]=10;

	for(int i=0;i<3;i++){
		printf("%d\n",p[i]);
	}
	free(p);
}
```

Sample Output:

```
C:\DS Lab Programs\additional>gcc melloc.c

C:\DS Lab Programs\additional>a
1
2
10
```

Program 2:

AIM: Program to calculate the GCD of two numbers using recursive function

Source Code:

```
#include <stdio.h>
// Function to calculate GCD using recursion
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}
int main() {
    int num1, num2;
    // Input two numbers from the user
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Calculate and display the GCD
    printf("GCD of %d and %d is: %d\n", num1, num2, gcd(num1, num2));
    return 0;
}
```

Sample Output:

```
C:\DS Lab Programs\additional>gcc gcd.c

C:\DS Lab Programs\additional>a
Enter two numbers: 25 45
GCD of 25 and 45 is: 5
```

Program 3:

AIM: **Program to illustrate the use of structures.**

```
#include <stdio.h>
  // declaring structure with name str1
struct str1 {
    int i;
    char c;
    float f;
    char s[30];
};
 // declaring structure with name str2
struct str2 {
    int ii;
    char cc;
    float ff;
} var; // variable declaration with structure template
  // Driver code
int main()
{
    // variable declaration after structure template
    // initialization with initializer list and designated
    //    initializer list
    struct str1 var1 = { 1, 'A', 1.00, "RNSIT" },
                var2;
    struct str2 var3 = { .ff = 5.00, .ii = 5, .cc = 'a' };

    // copying structure using assignment operator
    var2 = var1;
      printf("Struct 1:\n\ti = %d, c = %c, f = %f, s = %s\n",
          var1.i, var1.c, var1.f, var1.s);
    printf("Struct 2:\n\ti = %d, c = %c, f = %f, s = %s\n",
          var2.i, var2.c, var2.f, var2.s);
```

```
    printf("Struct 3\n\ti = %d, c = %c, f = %f\n", var3.ii,
          var3.cc, var3.ff);
     return 0;
}
```

Sample Output

```
Struct 1:
   i = 1, c = A, f = 1.000000, s = RNSIT
Struct 2:
   i = 1, c = A, f = 1.000000, s = RNSIT
Struct 3
   i = 5, c = a, f = 5.000000
```

Program 4:

AIM: **Program to print address and value of a variable using pointers.**

```
#include <stdio.h>
int main()
{
  int* pc, c;
  c = 22;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c);  // 22
  pc = &c;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 22
  c = 11;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 11
  *pc = 2;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c); // 2
  return 0;
}
```

Sample Output:

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2

Program 5:

AIM: Program to concatenate two string without using inbuild function.

```
#include<stdio.h>
int main(){
  int i,j;
  printf("Enter the string 1: ");
  char pk[10];
  gets(pk);
  printf("Enter the string 2: ");
  char ps[10];
  gets(ps);
  char ms[20];
  i=0;
  while(pk[i]!='\0'){
    ms[i]=pk[i];
    i++;
  }
  j=0;
  while(ps[j]!='\0'){
    ms[i]=ps[j];
    i++;
    j++;
  }
  ms[i]='\0';
```

```
  puts(ms);
  return 0;
}
```

```
C:\DS Lab Programs>gcc  07_concatanate.c

C:\DS Lab Programs>a
Enter the string 1: RNSIT
Enter the string 2: BANGALORE
RNSITBANGALORE
```