|  |  |  |  |
| --- | --- | --- | --- |
| **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** |

**Data Structures Manual**

**PROGRAM 01:**

/\*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.\*/

//Required Header files

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define the structure for a day

struct Day {

char \*name; // Dynamic string for day name

int date; // Integer for date

char \*description; // Dynamic string for activity description

};

// Function to create the calendar

void create(struct Day calendar[7]) {

for (int i = 0; i < 7; i++) {

// dynamically allocate memory for the day name and description

calendar[i].name = (char \*)malloc(50 \* sizeof(char)); // Assuming max length of day

name is 50 characters

calendar[i].description = (char \*)malloc(100 \* sizeof(char)); // Assuming max length of

description is 100 characters

}

}

// Function to read the weekly activity details

void read(struct Day calendar[7]) {

for (int i = 0; i < 7; i++) {

printf("Enter name for Day %d: ", i + 1);

scanf("%s", calendar[i].name);

printf("Enter date for Day %d: ", i + 1);

scanf("%d", &calendar[i].date);

printf("Enter description for Day %d: ", i + 1);

scanf("%s", calendar[i].description);

}

}

// Function to display the weekly activity details

void display(struct Day calendar[7]) {

for (int i = 0; i < 7; i++) {

printf("Day Name: %s\n", calendar[i].name);

printf("Date: %d\n", calendar[i].date);

printf("Description: %s\n", calendar[i].description);

printf("\n");

}

}

int main() {

struct Day calendar[7];

// Call the create function to create the calendar

create(calendar);

read(calendar);

// Display the weekly activity detail

printf("\nWeekly Activity Details:\n");

display(calendar);

// Free dynamically allocated memory

for (int i = 0; i < 7; i++) {

free(calendar[i].name);

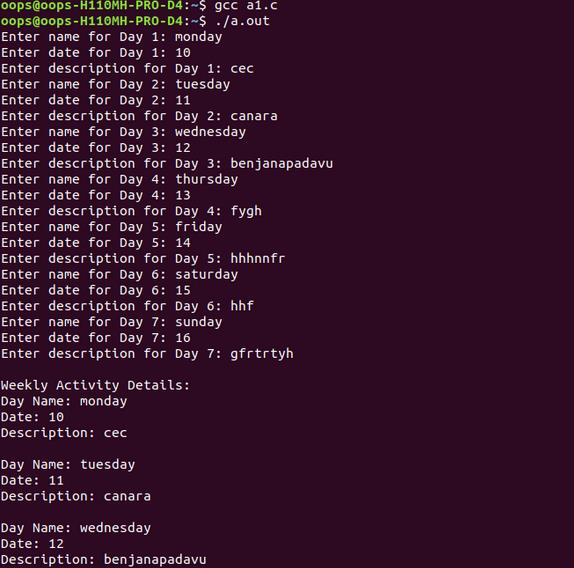
free(calendar[i].description);

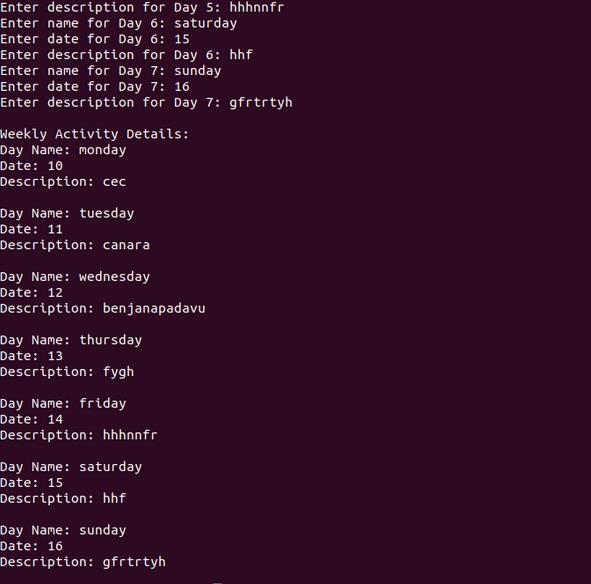
}

return 0;

}

**OUTPUT:**





Program 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>

#define MAX 1000

// Function to calculate the length of a string

int string\_length(char str[]) {

int len = 0;

while (str[len] != '\0')

len++;

return len;

}

// Function to find the pattern in the main string and replace it with the replacement string

void find\_and\_replace(char str[], char pat[], char rep[]) {

char result[MAX]; // To store the final modified string

int i = 0, j = 0, k, start, found = 0;

int str\_len = string\_length(str);

int pat\_len = string\_length(pat);

int rep\_len = string\_length(rep);

while (i < str\_len) {

// Check for pattern match

start = i;

for (k = 0; k < pat\_len; k++) {

if (str[i + k] != pat[k])

break;

}

// If pattern is found

if (k == pat\_len) {

found = 1;

// Copy replacement string to result

for (k = 0; k < rep\_len; k++) {

result[j++] = rep[k];

}

i += pat\_len; // Move past the pattern in main string

} else {

// Copy current character from main string to result

result[j++] = str[i++];

}

}

result[j] = '\0'; // Null terminate the result string

if (found) {

printf("Modified String: %s\n", result);

} else {

printf("Pattern not found in the main string.\n");

}

}

int main() {

char str[MAX], pat[MAX], rep[MAX];

// Reading input strings

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

gets(str);

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

gets(pat);

printf("Enter the replacement string (REP): ");

gets(rep);

// Perform pattern matching and replacement

find\_and\_replace(str, pat, rep);

return 0;

}

**OUTPUT:**

**Enter the main string (STR) : Hi Hello!**

**Enter the pattern string (PAT) :Hi**

**Enter the replacement string (REP): Bye**

**Modified String: Bye Hello!**

**Enter the main string (STR) : Hi Hello!**

**Enter the pattern string (PAT) :Bye**

**Enter the replacement string (REP): Hello**

**Pattern not found in the main string**

**PROGRAM 03:**

/\* Design, Develop and Implement 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>

#define MAX 5

int stack[MAX];

int top=-1;

//a. Push an Element on to Stack

void push()

{

int item;

// Stack Overflow situations

if(top==(MAX-1))

printf("\n Stack Overflow");

else

{

printf("\n Enter the element to be pushed :");

scanf("%d",&item);

// pushing element to the top of stack

stack[++top]=item;

}

}

//b. Pop an Element from Stack

void pop()

{

// Stack Underflow situations

if(top==-1)

printf("\n Stack Underflow");

else

printf(" \nPoped element is %d ",stack[top--]); // poping element from the top of stack

}

//e. Display the status of Stack

void display()

{

int i;

if(top==-1)

printf("\n Sorry Empty Stack");

else

{

printf("\nThe elements of the stack are\n");

for(i=top;i>=0;i--)

printf("stack[%d] = %d\n",i, stack[i]);

}

}

//c. Demonstrate how Stack can be used to check Palindrome

void palindrome()

{

int i,count=0;

for(i=0; i<=(top/2); i++)

{

if(stack[i] == stack[top-i])

count++;

}

if((top/2+1)==count)

printf("\n Stack contents are Palindrome");

else

printf("\nStack contents are not palindrome");

}

void main()

{

int choice;

while(1)

{

printf("\n STACK OPERATIONS\n");

printf("1.Push\n 2.Pop\n 3.Display\n 4.Palindrome\n 5.Exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:push();

break;

case 2:pop();

break;

case 3:display();

break;

case 4:palindrome();

break;

case 5:return;

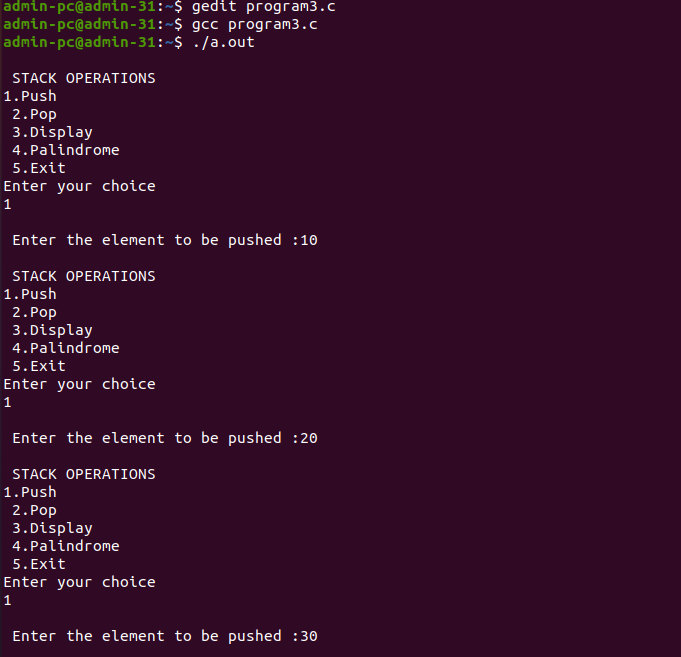
default: printf("Invalid choice\n");

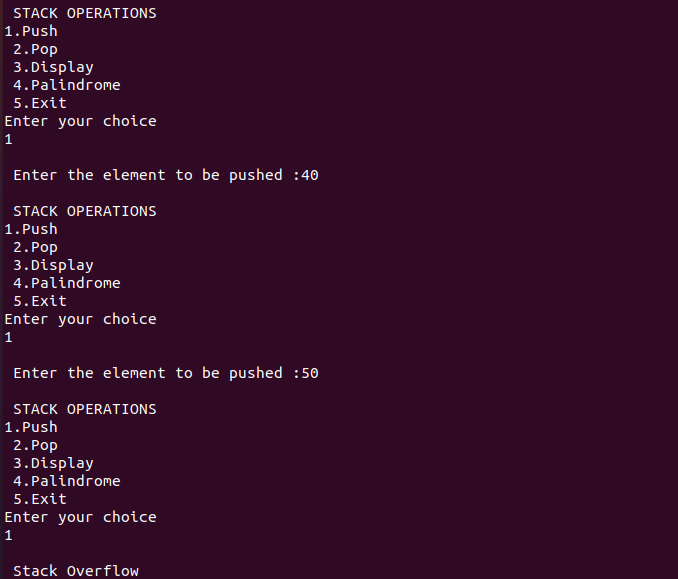
}

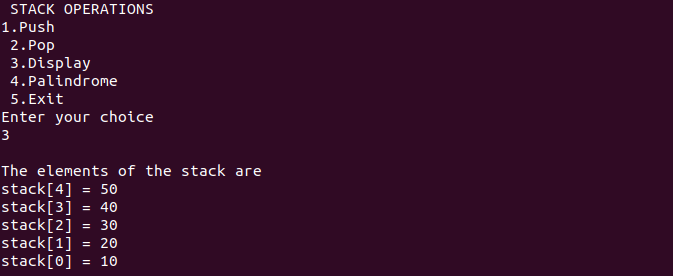
}

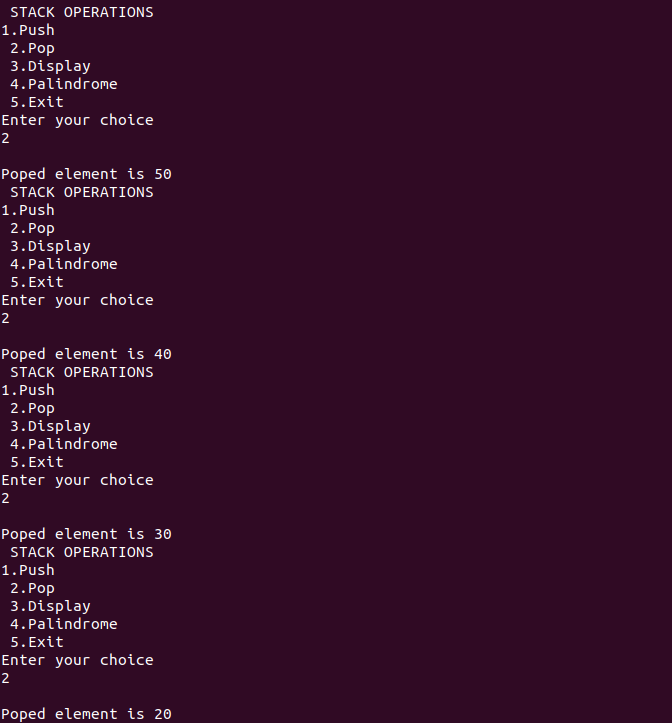
}

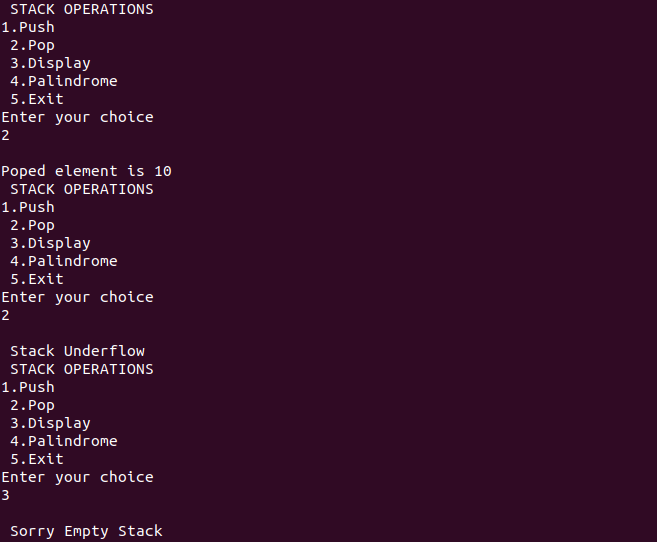
**OUTPUT:**

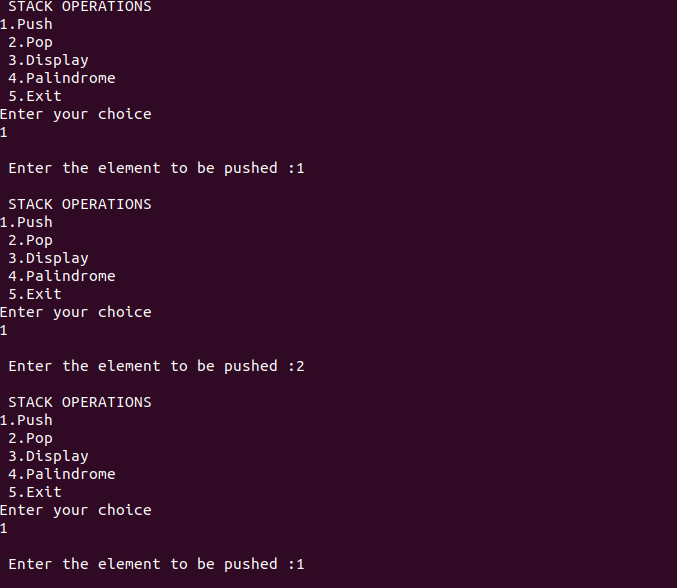


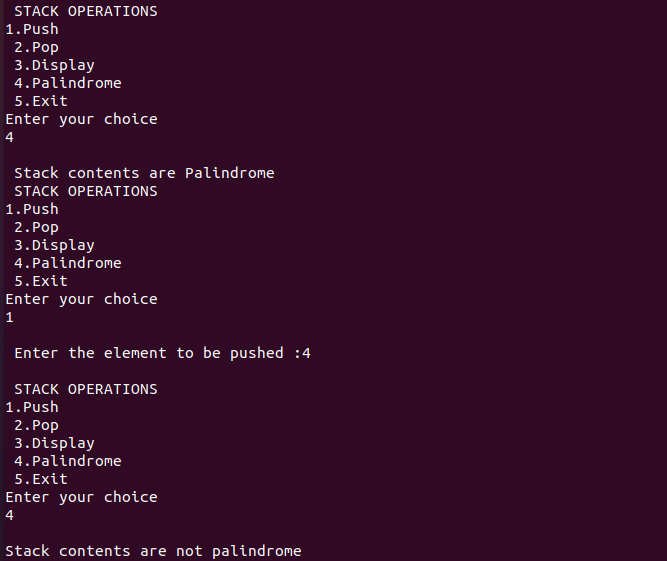












**PROGRAM 04:**

/\*Design, Develop and Implement 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 <stdlib.h>

#include <string.h>

int F(char symbol) {

switch (symbol) {

case '+':

case '-': return 2;

case '\*':

case '/': return 4;

case '^':

case '$': return 5;

case '(': return 0;

case '#': return -1;

default: return 8;

}

}

int G(char symbol) {

switch (symbol) {

case '+':

case '-': return 1;

case '\*':

case '/': return 3;

case '^':

case '$': return 6;

case '(': return 9;

case ')': return 0;

default: return 7;

}

}

void infix\_postfix(char infix[], char postfix[])

{

int top, j, i;

char s[30], symbol;

top = -1;

s[++top] = '#';

j = 0;

for (i = 0; i < strlen(infix); i++)

{

symbol = infix[i]; // Assign the character from infix to symbol

while ( F(s[top]) > G(symbol) )

{

postfix[j] = s[top--];

j++;

}

if ( F(s[top]) != G(symbol) )

s[++top] = symbol;

else

top--;

}

while ( s[top] != '#' )

{

postfix[j++] = s[top--];

}

postfix[j] = '\0'; // Terminate the postfix string

}

int main() {

char infix[20];

char postfix[20];

printf("Enter a valid infix expression: ");

scanf("%s", infix);

infix\_postfix(infix, postfix);

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

return 0;

}

**OUTPUT:**

**Enter a valid infix expression: (A+(B-C)\*D)**

**The postfix expression is: ABC-D\*+**

**PROGRAM 05:**

/\* Design, Develop and Implement a Program in C for the following Stack Applications

a. Evaluation of postfix expression with single digit operands and operators: +, -, \*, /, %, ^

b. Solving Tower of Hanoi problem with N disks

\*/

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <ctype.h>

int count = 0; // Global counter for move count

// Function to compute basic operations

double compute(char symbol, double op1, double op2) {

switch (symbol) {

case '+': return op1 + op2;

case '-': return op1 - op2;

case '\*': return op1 \* op2;

case '/': return op1 / op2;

case '%': return (int)op1 % (int)op2; // Modulo operator for integer division

case '$':

case '^': return pow(op1, op2);

default: return 0;

}

}

// Function to evaluate a postfix expression

void evaluate\_postfix() {

double s[20], res, op1, op2;

int top = -1, i;

char postfix[20], symbol;

printf("Enter the postfix expression: ");

scanf("%s", postfix);

for (i = 0; i < strlen(postfix); i++) {

symbol = postfix[i];

if (isdigit(symbol)) {

s[++top] = (symbol - '0'); // ASCII conversion to digit

} else {

op2 = s[top--];

op1 = s[top--];

res = compute(symbol, op1, op2);

s[++top] = res;

}

}

res = s[top--];

printf("The result is: %f\n", res);

}

// New Tower of Hanoi function

void tower(int n, int source, int temp, int dest) {

if (n == 1) {

printf("\nMove the disc from %c to %c", source + 'A' - 1, dest + 'A' - 1);

count++;

return;

}

tower(n - 1, source, dest, temp);

printf("\nMove the disc from %c to %c", source + 'A' - 1, dest + 'A' - 1);

count++;

tower(n - 1, temp, source, dest);

}

int main() {

int choice, n;

do {

printf("\n1. Evaluate postfix expression\n");

printf("2. Solve Tower of Hanoi problem\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

evaluate\_postfix();

break;

case 2:

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

scanf("%d", &n);

count = 0; // Reset the move count for each new problem

printf("The moves involved in solving Tower of Hanoi for %d disks are:\n", n);

tower(n, 1, 2, 3); // 1 for 'A', 2 for 'B', 3 for 'C'

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

break;

case 3:

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

break;

default:

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

}

} while (choice != 3);

return 0;

}

**OUTPUT:**

**1. Evaluate postfix expression**

**2. Solve Tower of Hanoi problem**

**3. Exit**

**Enter your choice: 1**

**Enter the postfix expression: 53+82-\***

**The result is: 48.000000**

**1. Evaluate postfix expression**

**2. Solve Tower of Hanoi problem**

**3. Exit**

**Enter your choice: 2**

**Enter the number of disks: 3**

**The moves involved in solving Tower of Hanoi for 3 disks are:**

**Move the disc from A to C**

**Move the disc from A to B**

**Move the disc from C to B**

**Move the disc from A to C**

**Move the disc from B to A**

**Move the disc from B to C**

**Move the disc from A to C**

**Total moves: 7**

**1. Evaluate postfix expression**

**2. Solve Tower of Hanoi problem**

**3. Exit**

**Enter your choice: 5**

**Invalid choice. Please try again.**

**1. Evaluate postfix expression**

**2. Solve Tower of Hanoi problem**

**3. Exit**

**Enter your choice: 3**

**Exiting the program.**

**PROGRAM 06:**

/\* Design, Develop and Implement 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

#include <stdio.h>

#define MAX\_SIZE 5

int q[MAX\_SIZE];

int f = 0, r = -1;

int count = 0;

void delete\_front() {

if (count == 0) {

printf("Underflow of queue\n");

return;

}

printf("The deleted element is %d\n", q[f]);

f = (f + 1) % MAX\_SIZE;

count--;

}

void insert\_rear(int item) {

if (count == MAX\_SIZE) {

printf("Overflow of queue\n");

return;

}

r = (r + 1) % MAX\_SIZE;

q[r] = item;

count++;

}

void display() {

if (count == 0) {

printf("Queue is empty\n");

return;

}

printf("Contents of queue are: ");

int i, temp = f;

for (i = 0; i < count; i++) {

printf("%d ", q[temp]);

temp = (temp + 1) % MAX\_SIZE;

}

printf("\n");

}

int main() {

int choice, item;

for (;;) {

printf("\nCircular Queue Operations:\n");

printf("1. Insert Rear\n");

printf("2. Delete Front\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the element to insert: ");

scanf("%d", &item);

insert\_rear(item);

break;

case 2:

delete\_front();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Please enter a valid option.\n");

}

}

}

**OUTPUT:**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 10**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 20**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 30**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 40**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 3**

**Contents of queue are: 10 20 30 40 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 1**

**Enter the element to insert: 60**

**Overflow of queue**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**The deleted element is 10**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 3**

**Contents of queue are: 20 30 40 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**The deleted element is 20**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 3**

**Contents of queue are: 30 40 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**The deleted element is 30**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 3**

**Contents of queue are: 40 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**The deleted element is 40**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**The deleted element is 50**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 3**

**Queue is empty**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 2**

**Underflow of queue**

**Circular Queue Operations:**

**1. Insert Rear**

**2. Delete Front**

**3. Display**

**4. Exit**

**Enter your choice: 4**

**Exiting...**

**Program 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>

struct student {

char usn[15];

char name[50];

char programme[50];

int sem;

char phone[15];

struct student \*link;

};

typedef struct student \*NODE;

NODE getnode() {

NODE x;

x = (NODE) malloc(sizeof(struct student));

return x;

}

NODE insert\_front(NODE first) {

NODE temp = getnode();

printf("Enter USN: ");

scanf("%s", temp->usn);

printf("Enter Name: ");

scanf("%s", temp->name);

printf("Enter Programme: ");

scanf("%s", temp->programme);

printf("Enter Semester: ");

scanf("%d", &temp->sem);

printf("Enter Phone Number: ");

scanf("%s", temp->phone);

temp->link = first;

return temp;

}

void display(NODE first) {

NODE temp = first;

int count = 0;

if (first == NULL) {

printf("List is empty\n");

return;

}

printf("Student List:\n");

while (temp != NULL) {

printf("USN: %s, Name: %s, Programme: %s, Semester: %d, Phone: %s\n",

temp->usn, temp->name, temp->programme, temp->sem, temp->phone);

temp = temp->link;

count++;

}

printf("Total number of students: %d\n", count);

}

NODE insert\_rear(NODE first) {

NODE temp = getnode();

printf("Enter USN: ");

scanf("%s", temp->usn);

printf("Enter Name: ");

scanf("%s", temp->name);

printf("Enter Programme: ");

scanf("%s", temp->programme);

printf("Enter Semester: ");

scanf("%d", &temp->sem);

printf("Enter Phone Number: ");

scanf("%s", temp->phone);

temp->link = NULL;

if (first == NULL) return temp;

NODE cur = first;

while (cur->link != NULL) {

cur = cur->link;

}

cur->link = temp;

return first;

}

NODE delete\_front(NODE first) {

if (first == NULL) {

printf("List is empty\n");

return first;

}

NODE temp = first;

first = first->link;

printf("Deleted student with USN: %s\n", temp->usn);

free(temp);

return first;

}

NODE delete\_rear(NODE first) {

if (first == NULL) {

printf("List is empty\n");

return NULL;

}

if (first->link == NULL) {

printf("Deleted student with USN: %s\n", first->usn);

free(first);

return NULL;

}

NODE prev = NULL, cur = first;

while (cur->link != NULL) {

prev = cur;

cur = cur->link;

}

printf("Deleted student with USN: %s\n", cur->usn);

free(cur);

prev->link = NULL;

return first;

}

int main() {

NODE first = NULL;

int ch;

while (1) {

printf("\n---- Singly Linked List(SLL) Menu ----");

printf("\n1. Create (Insert at Front)");

printf("\n2. Display and Count");

printf("\n3. Insert at End");

printf("\n4. Delete from Front (Stack Pop)");

printf("\n5. Delete from End");

printf("\n6. Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

first = insert\_front(first);

break;

case 2:

display(first);

break;

case 3:

first = insert\_rear(first);

break;

case 4:

first = delete\_front(first);

break;

case 5:

first = delete\_rear(first);

break;

case 6:

exit(0);

default:

printf("Invalid choice!\n");

}

}

return 0;

}

**OUTPUT:**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 1**

**Enter USN: 1RV21CS001**

**Enter Name: Alice**

**Enter Programme: CSE**

**Enter Semester: 5**

**Enter Phone Number: 9876543210**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 1**

**Enter USN: 1RV21CS002**

**Enter Name: Bob**

**Enter Programme: CSE**

**Enter Semester: 6**

**Enter Phone Number: 9876543211**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 2**

**Student List:**

**USN: 1RV21CS002, Name: Bob, Programme: CSE, Semester: 6, Phone: 9876543211**

**USN: 1RV21CS001, Name: Alice, Programme: CSE, Semester: 5, Phone: 9876543210**

**Total number of students: 2**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 3**

**Enter USN: 1RV21CS003**

**Enter Name: Charlie**

**Enter Programme: CSE**

**Enter Semester: 7**

**Enter Phone Number: 9876543212**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 2**

**Student List:**

**USN: 1RV21CS002, Name: Bob, Programme: CSE, Semester: 6, Phone: 9876543211**

**USN: 1RV21CS001, Name: Alice, Programme: CSE, Semester: 5, Phone: 9876543210**

**USN: 1RV21CS003, Name: Charlie, Programme: CSE, Semester: 7, Phone: 9876543212**

**Total number of students: 3**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 4**

**Deleted student with USN: 1RV21CS002**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 5**

**Deleted student with USN: 1RV21CS003**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 2**

**Student List:**

**USN: 1RV21CS001, Name: Alice, Programme: CSE, Semester: 5, Phone: 9876543210**

**Total number of students: 1**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 4**

**Deleted student with USN: 1RV21CS001**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 2**

**List is empty**

**---- Singly Linked List(SLL) Menu ----**

**1. Create (Insert at Front)**

**2. Display and Count**

**3. Insert at End**

**4. Delete from Front (Stack Pop)**

**5. Delete from End**

**6. Exit**

**Enter your choice: 6**

**Exiting...**

**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>

struct node {

int value;

struct node \*ltree, \*rtree;

};

typedef struct node\* NODE;

NODE getnode() {

NODE x = (NODE) malloc(sizeof(struct node));

x->ltree = x->rtree = NULL;

return x;

}

NODE create(int item, NODE root) {

NODE temp = getnode();

temp->value = item;

if (root == NULL) return temp;

NODE cur = root, prev = NULL;

while (cur != NULL) {

prev = cur;

if (temp->value == cur->value) return root;

cur = (temp->value < cur->value) ? cur->ltree : cur->rtree;

}

if (temp->value < prev->value) prev->ltree = temp;

else prev->rtree = temp;

return root;

}

void inorder(NODE root) {

if (root != NULL) {

inorder(root->ltree);

printf("%d\t", root->value);

inorder(root->rtree);

}

}

void preorder(NODE root) {

if (root != NULL) {

printf("%d\t", root->value);

preorder(root->ltree);

preorder(root->rtree);

}

}

void postorder(NODE root) {

if (root != NULL) {

postorder(root->ltree);

postorder(root->rtree);

printf("%d\t", root->value);

}

}

void search(NODE root) {

int item, found = 0;

printf("Enter the element to be searched: ");

scanf("%d", &item);

NODE cur = root;

while (cur != NULL) {

if (item == cur->value) {

found = 1;

printf("Found key %d in tree\n", cur->value);

break;

}

cur = (item < cur->value) ? cur->ltree : cur->rtree;

}

if (!found) printf("Key not found\n");

}

int main() {

int choice, item, n;

NODE root = NULL;

while (1) {

printf("\n1. Create BST\n2. BST Traversal\n3. Search\n4. Exit\nEnter Your Choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter number of elements: ");

scanf("%d", &n);

for (int i = 0; i < n; i++) {

printf("Enter item to be inserted: ");

scanf("%d", &item);

root = create(item, root);

}

break;

case 2:

if (root == NULL) printf("Tree is empty\n");

else {

printf("\nPREORDER:\n");

preorder(root);

printf("\nINORDER:\n");

inorder(root);

printf("\nPOSTORDER:\n");

postorder(root);

printf("\n");

}

break;

case 3:

search(root);

break;

case 4:

return 0;

default:

printf("Invalid Choice\n");

}

}

}

**OUTPUT:**

**1. Create BST**

**2. BST Traversal**

**3. Search**

**4. Exit**

**Enter Your Choice: 1**

**Enter number of elements: 12**

**Enter item to be inserted: 6**

**Enter item to be inserted: 9**

**Enter item to be inserted: 5**

**Enter item to be inserted: 2**

**Enter item to be inserted: 8**

**Enter item to be inserted: 15**

**Enter item to be inserted: 24**

**Enter item to be inserted: 14**

**Enter item to be inserted: 7**

**Enter item to be inserted: 8**

**Enter item to be inserted: 5**

**Enter item to be inserted: 2**

**1. Create BST**

**2. BST Traversal**

**3. Search**

**4. Exit**

**Enter Your Choice: 2**

**PREORDER:**

**6 5 2 9 8 7 15 14 24**

**INORDER:**

**2 5 6 7 8 9 14 15 24**

**POSTORDER:**

**2 5 7 8 9 14 24 15 6**

**1. Create BST**

**2. BST Traversal**

**3. Search**

**4. Exit**

**Enter Your Choice: 3**

**Enter the element to be searched: 15**

**Found key 15 in tree**

**1. Create BST**

**2. BST Traversal**

**3. Search**

**4. Exit**

**Enter Your Choice: 3**

**Enter the element to be searched: 20**

**Key not found**

**1. Create BST**

**2. BST Traversal**

**3. Search**

**4. Exit**

**Enter Your Choice: 4**

**PROGRAM 09:**

**/\* 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) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3**

**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 operation \*/**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include <stdio\_ext.h>

struct node // polynomial node

{

int coef;

int x,y,z;

struct node \*link;

};

typedef struct node \*NODE;

NODE getnode() // create a node

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

return x;

} // end of getnode

NODE readpoly()

{

NODE temp,head,cur;

char ch;

head=getnode(); // create a head node and set all values to -1 it is similar to FIRST in

SLL program

head->coef=-1;

head->x=-1;

head->y=-1;

head->z=-1;

head->link=head; // self reference

do

{

temp=getnode(); // create a polynomial node

printf("\nEnter the coefficient and exponent in decreasing order\n");

scanf("%d%d%d%d",&temp->coef,&temp->x,&temp->y,&temp->z );

cur=head;

while(cur->link!=head) // find the last node

cur=cur->link;

cur->link=temp; // connect new node to the last node

temp->link=head; // point back to head

printf("\nDo you want to enter more coefficients(y/n)");

\_\_fpurge(stdin); // to clear the stdin buffer

scanf("%c",&ch);

} while(ch =='y' || ch == 'Y');

return head; // return the polynomial list

} // end of readpoly

int compare(NODE a,NODE b) // function to compare the A and B polynomial nodes

{

if(a->x > b->x)

return 1;

else if(a->x < b->x)

return -1;

else if(a->y > b->y)

return 1;

else if(a->y < b->y)

return -1;

else if(a->z > b->z)

return 1;

else if(a->z < b->z)

return -1;

return 0;

} // end of compare

void attach(int cf,int x1,int y1, int z1, NODE \*ptr) // function to attach the A and B

polynomial node to C Polynomial

{

NODE temp;

temp=getnode();

temp->coef=cf;

temp->x=x1;

temp->y=y1;

temp->z=z1;

(\*ptr)->link=temp;

\*ptr=temp;

} // end of attach

NODE addpoly(NODE a,NODE b) // function to add polynomial A and B i.e, C=A+B

{

scanf("%d", &z);

cur=ptr->link; // start with HEAD

while(cur!=ptr) // Repeat until the end of list

{

ex=cur->x; // exponent of x

ey=cur->y; // exponent of y

ez=cur->z; // exponent of z

cof=cur->coef; // coefficient

res+=cof\*pow(x,ex)\*pow(y,ey)\*pow(z,ez); // compute result for each

polynomial

cur=cur->link; // move to next node

}

printf("\nresult: %d",res);

} // end of evaluate

void main(void)

{

int i, ch;

NODE a=NULL,b,c;

while(1)

{

printf("\n1: Represent first polynomial A");

printf("\n2: Represent Second polynomial B");

printf("\n3: Display the polynomial A");

printf("\n4: Display the polynomial B");

printf("\n5: Add A & B polynomials"); // C=A+B

printf("\n6: Evaluate polynomial C");

printf("\n7: Exit");

printf("\n Enter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the elements of the polynomial A");

a=readpoly();

break;

case 2:printf("\nEnter the elements of the polynomial B");

b= readpoly();

break;

case 3: print(a); // display polynomial A

break;

case 4:print(b); // display polynomial A

break;

case 5: c=addpoly(a,b); // C=A+B

printf("\nThe sum of two polynomials is: ");

print(c); // display polynomial C

printf("\n");

break;

case 6:evaluate(c); // Evaluate polynomial C

break;

case 7: return;

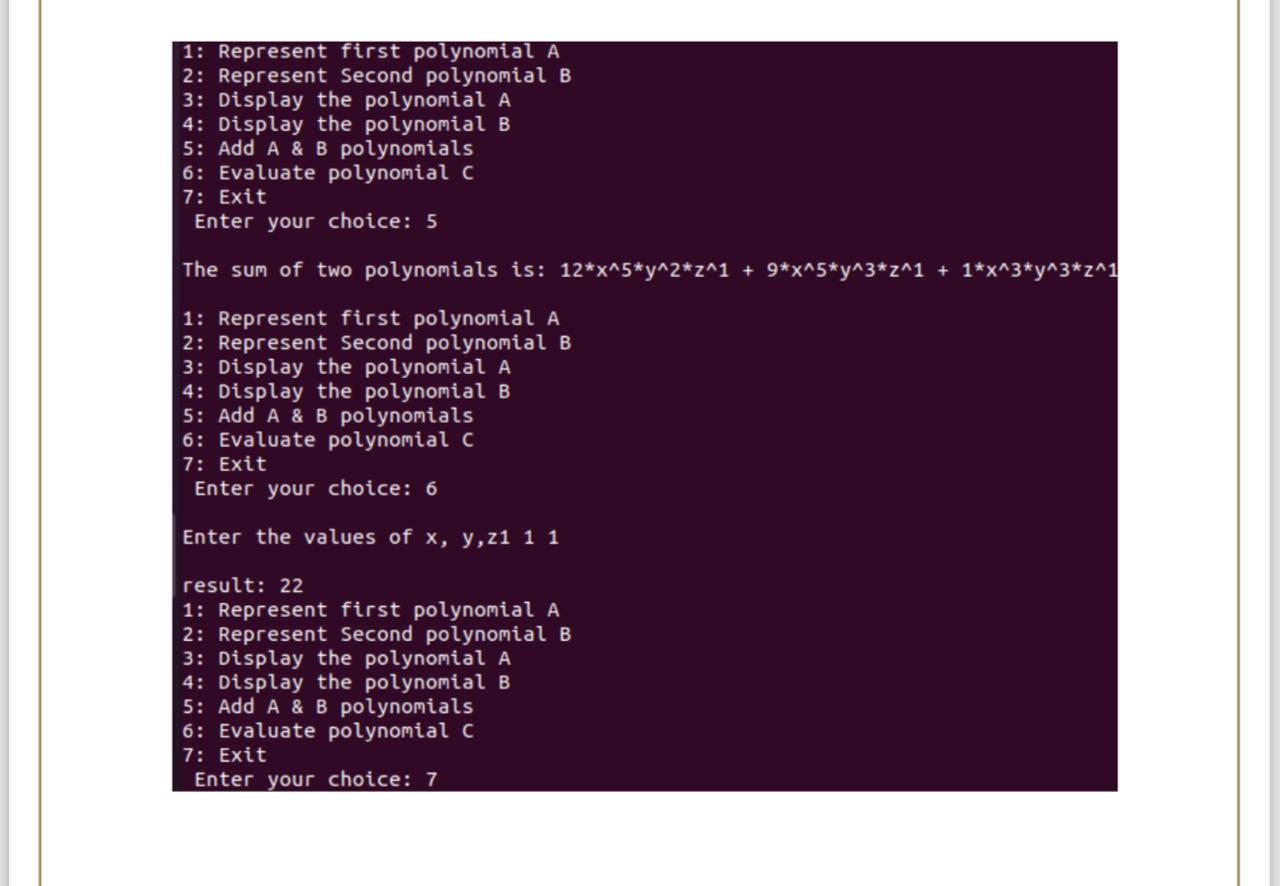
default: printf("\nInvalid choice!\n");

} //end of switch

} // end of while

} // end of main





**PROGRAM 11:**

/\* Design, Develop and Implement 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>

int a[20][20], q[20], visited[20];

int n, i, j, f=0, r=-1;

void create\_graph() // Create the Digraph using Adjacency matrix

{

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

scanf("%d",&n);

printf("\n Enter graph data in matrix form:\n");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

scanf("%d",&a[i][j]); // read adjacency matrix

return;

}

void bfs(int v) // Reachability using Breadth First Search

{

for(i=1;i<=n;i++)

if(a[v][i] && !visited[i]) // check weather node is visited

q[++r]=i; // if not add it Queue

if(f<=r) // check for non empty Queue

{

visited[q[f]]=1; // set visited status for front node of Queue

bfs(q[f++]); // recursive call BSF

}

}// end of BSF

void main()

{

int v, choice;

while(1)

{

printf("\n1. Create a Digraph of N cities using Adjacency Matrix");

printf("\n2. Print all the nodes reachable from a given starting node in a digraph using

BFS method") ;

printf("\n3. Exit");

printf("\n Enter Your Choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: create\_graph();

break;

case 2: printf("Enter the source vertex: ");

scanf("%d",&v);

if((v<1)||(v>n)) // check for valid source entry

printf("\nEnter a valid source vertex");

else // if valid begin test for reachability

{

for(i=1;i<=n;i++) // begin with assuming all cities are not visited

visited[i]=0;

visited[v]=1; // source is visited

bfs(v); // cal BFS to check reachability

printf("The reachable nodes from node %d:\n",v);

for(i=1;i<=n;i++) // display reachable cities from the source city

if(visited[i] && i !=v)

printf("node %d\n",i);

}

break;

case 3:return;

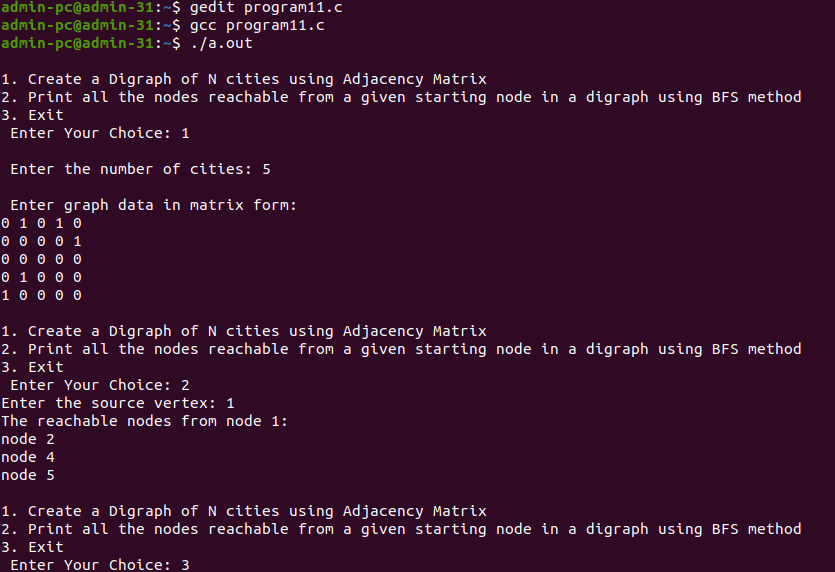
default:printf("\nInvalid Choice");

} // end of switch

} // end of while

} // end of main

Output:



**PROGRAM 12:**

/\* Given a File of N employee records with a set K of Keys(4-digit) which uniquely determinethe 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. Design and 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>

#define MAX 10

void linear\_prob(int a[MAX])

{

int flag, key, i, addrs, count;

char ans;

do

{

flag=0;

count=0;

printf("\n Enter 4 digit Key : ");

scanf("%4d", &key); // read a key

addrs=key%10; // generate single digit key for given key

if(a[addrs]== -1) // check for empty entry in Hash table

a[addrs] = key; // if yes the add to HT

else // if entry exits then avoid collision

{

printf("\nCollision Detected...!!!\n");

i=0;

while(i<MAX) // check for next available empty location in HT

{

if (a[i]!=-1)

count++; // count empty location in HT

i++;

} // end of while

if(count == MAX) // if HT is full then display HT and return

{

printf("\n Hash table is full \n");

display(a); // Display HT

return;

} // end of if

printf("\nCollision avoided successfully using LINEAR PROBING\n");

for(i=addrs+1; i<MAX; i++) // if there is empty space after key in HT then

//make a entry in HT

if(a[i] == -1)

{

a[i] = key;

flag =1;

break;

} // end of if

i=0;

while((i<addrs) && (flag==0)) // check for empty space before key in HT then

//make a entry in HT

{

if(a[i] == -1)

{

a[i] = key;

flag=1;

break;

} // end of if

i++;

} // end of while

} // end of else

printf("\n Do you wish to continue ? (y/n) ");

fflush(stdin);

scanf("%c",&ans);

} while(ans=='y' || ans == 'Y') ;// end of do while

} // end of linear probe

void display(int a[MAX]) // display the HT

{

int i;

printf("\n the HASH TABLE is\n Addrs \t Key");

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

printf("\n %d \t %d ", i, a[i]);

}// end of display

void main()

{

int a[MAX], i, choice;

for (i=0;i<MAX;i++) // initialize HT with no entries

a[i] = -1;

while(1)

{

printf("\n Collision handling by Linear Probing ");

printf("\n1. Insert into Hash table");

printf("\n2. Display Hash table");

printf("\n3. Exit");

printf("\n Enter your Choice : ");

scanf("%d",&choice);

switch (choice)

{

case 1: linear\_prob(a);

break;

case 2: display(a);

break;

case 3: return;

default: printf("\n Invalid Choice");

} // end of switch

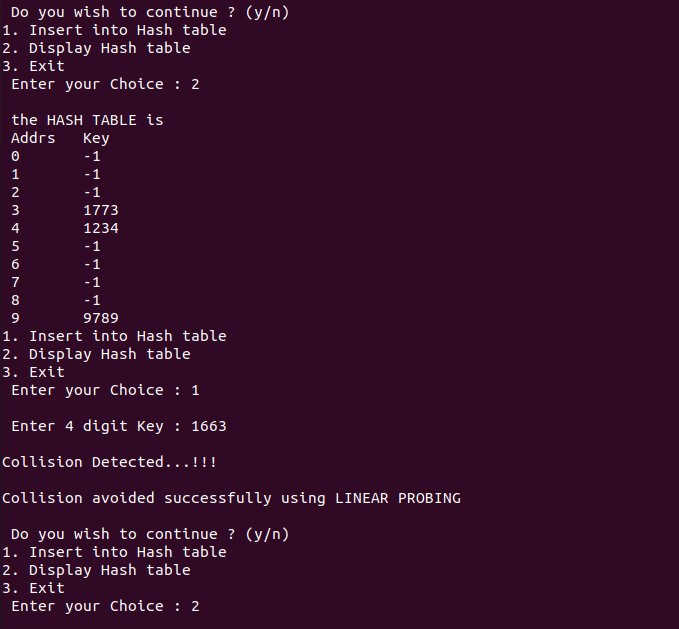
} // end of while

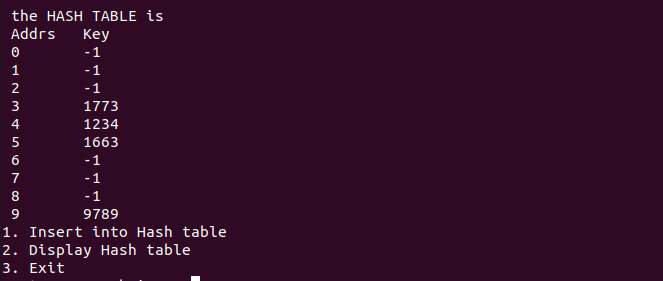
} // end of main

OUTPUT:









**PROGRAM 08:**

/\* Design, Develop and Implement 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>**

**struct node // structure to store employee details**

**{**

**char ssn[12],name[20],dept[25],desig[20];**

**unsigned long long int phno;**

**float sal;**

**struct node \*prev;**

**struct node \*next;**

**};**

**typedef struct node \*NODE; // renaming node as NODE**

**NODE temp, FIRST=NULL,END=NULL;**

**NODE getnode() // Create node**

**{**

**NODE x;**

**x=(NODE)malloc(sizeof(struct node));**

**x->prev=NULL;**

**x->next=NULL;**

**return x;**

**}**

**void read() // read details of employee**

**{**

**temp=getnode();**

**printf("Enter SSN:");**

**scanf("%s",temp->ssn);**

**printf("Enter Name:");**

**scanf("%s",temp->name);**

**printf("Enter Dept:");**

**scanf("%s",temp->dept);**

**printf("Enter Designation:");**

**scanf("%s",temp->desig);**

**printf("Enter Phno:");**

**scanf("%llu",&temp->phno);**

**printf("Enter Salary:");**

**scanf("%f",&temp->sal);**

**return;**

**}**

**void Create\_DLL() // Create a DLL of N Employees Data by using end insertion.**

**{**

**int n,i=1;**

**printf("Enter the number of Employees \n");**

**scanf("%d",&n);**

**while(i<=n)**

**{**

**printf("Enter the details of the %d employee\n", i++);**

**read();**

**if(FIRST==NULL) // if empty list new node will be the first node**

**{**

**FIRST=temp;**

**END=temp;**

**}**

**else //otherwise find the last node and insert the new node**

**{**

**END->next=temp;**

**temp->prev=END;**

**END=temp;**

**}**

**} //e nd of while**

**} // end of create()**

**void display\_count() //Display the status of DLL and count the number of nodes in it**

**{**

**temp=FIRST;**

**int count=0;**

**if(FIRST==NULL) // check for empty list**

**printf("the Employee detail is NULL and count is %d\n", count);**

**else**

**{**

**printf("Employee details:\n");**

**printf("SSN \t EMPLOYEE NAME \t DEPARTMENT \t DESIGNATION \t**

**PHONE NUMBER \t SALARY");**

**while(temp!=NULL) // display all nodes in the list**

**{**

**count++;**

**printf("\n%s\t%s\t%s\t%s\t%llu\t%0.2f",temp->ssn, temp->name, temp-**

**>dept,temp->desig,temp->phno,temp->sal);**

**temp=temp->next;**

**}**

**printf("\n Employee count is %d\n",count);**

**} // end of else**

**return;**

**} // end of display()**

**void Insertionfront() //Perform Insertion at front of DLL**

**{**

**printf("Enter the details of the employee\n");**

**read();**

**if(FIRST==NULL) // if empty list new node will be the first node**

**{**

**FIRST=temp;**

**END=temp;**

**}**

**else // otherwise insert the new node at front**

**{**

**temp->next=FIRST;**

**FIRST->prev=temp;**

**FIRST=temp;**

**}**

**} // end of insert front**

**void Insertionend() //Perform Insertion at End of DLL**

**{**

**printf("Enter the details of the new employee\n");**

**read();**

**if(FIRST==NULL) // check for empty list**

**{**

**FIRST=temp;**

**END=temp;**

**}**

**else // otherwise find the last node and insert the new node**

**{**

**END->next=temp;**

**temp->prev=END;**

**END=temp;**

**}**

**return ;**

**} // end of insert end**

**void Deletionfront() //Delete node from front of DLL**

**{**

**temp=FIRST;**

**if(FIRST==NULL) // check for empty list**

**printf("List is empty\n");**

**else if(FIRST==END) // otherwise check for single node in list**

**{**

**printf("deleted element is %s\n", temp->ssn);**

**FIRST=NULL;**

**END=NULL;**

**free(temp);**

**}**

**else // otherwise delete node from front of DLL**

**{**

**printf("deleted element is %s\n", temp->ssn);**

**FIRST =FIRST->next;**

**FIRST->prev=NULL;**

**free(temp);**

**}**

**return;**

**} // end of deletefront**

**void Deletionend() // delete node at end of DLL**

**{**

**temp = END;**

**if(FIRST==NULL) // check for empty list**

**printf("List is empty\n");**

**else if(FIRST==END) // otherwise check for single node in list**

**{**

**printf("deleted element is %s\n", temp->ssn);**

**FIRST=NULL;**

**END=NULL;**

**free(temp);**

**}**

**else // otherwise delete end node from DLL**

**{**

**printf("deleted element is %s\n", temp->ssn);**

**END=END->prev;**

**END->next=NULL;**

**free(temp);**

**}**

**return ;**

**} // end of deletionend**

**void main()**

**{**

**int choice;**

**while(1)**

**{**

**printf(" \n 1 - Create DLL of N Employees \n 2 - Display DLL \n 3 - Insertion**

**at front \n 4 - Insertion at end");**

**printf("\n 5 - Deletion at front \n 6 - Deletion at end \n 7 - Exit\n");**

**printf("Enter Your Choice: ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1 : Create\_DLL();**

**break;**

**case 2 : display\_count();**

**break;**

**case 3 : Insertionfront();**

**break;**

**case 4 : Insertionend();**

**break;**

**case 5 : Deletionfront();**

**break;**

**case 6 : Deletionend();**

**break;**

**case 7 : return;**

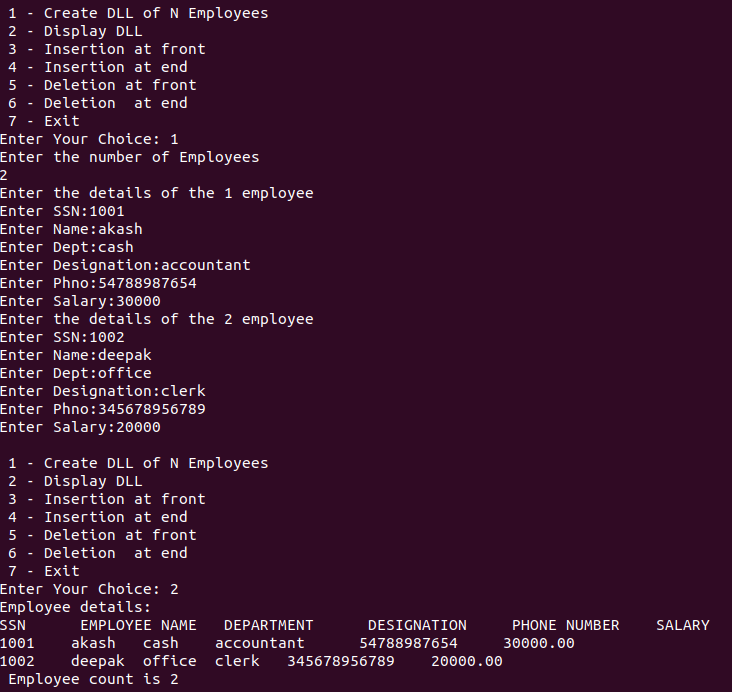
**default: printf("Invalid Choice\n");**

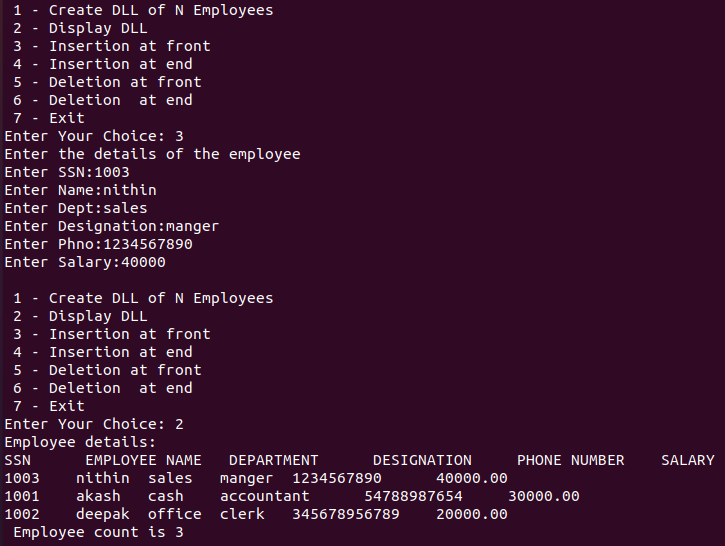
**} //end of switch**

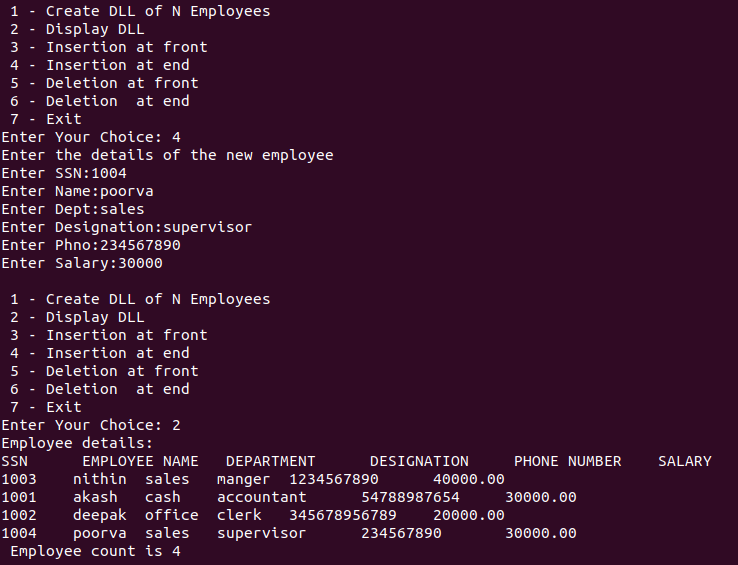
**} // end of while**

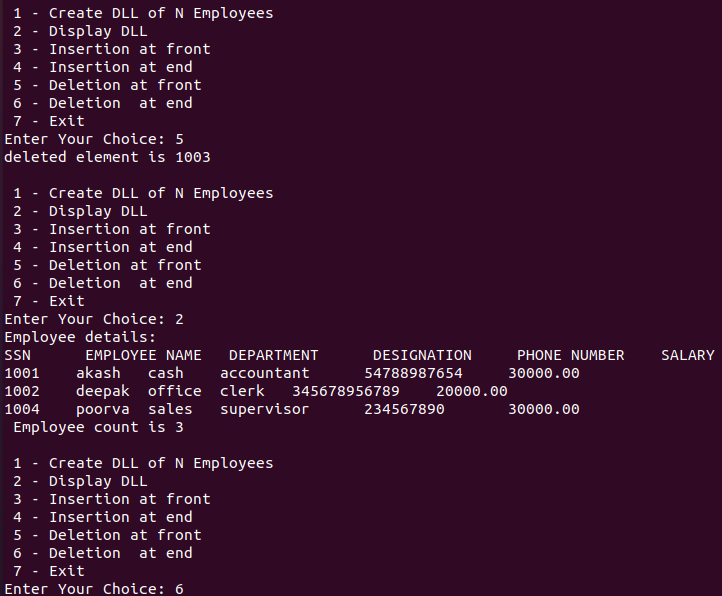
**} // end of main**

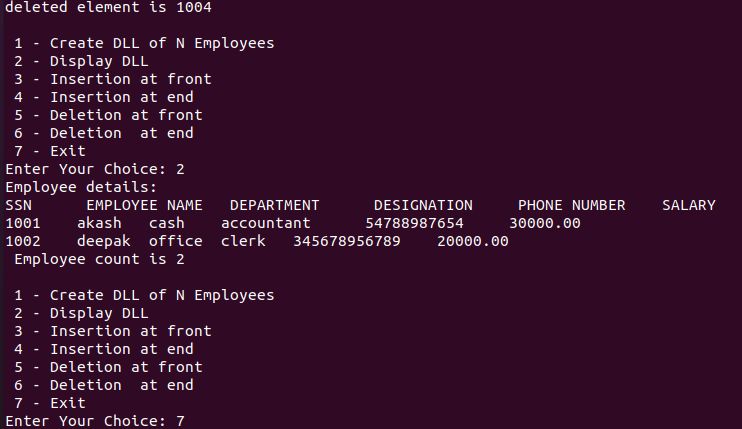
**OUTPUT:**











VIVA QUESTIONS

PROGRAM 01:

1. What is a structure in C? How do you define a structure in C? Provide an example.

2. What is the difference between a structure and a union in C?

3. How do you access members of a structure?

4. Can a structure have another structure as a member? Explain with an example.

5. How do you pass a structure to a function? Can you pass it by value or do you have to pass it by reference?

6. What is the difference between an array of structures and a structure of arrays?

7. How can you dynamically allocate memory for a structure in C?

8. Explain the concept of bit-fields in structures.

9. What is the significance of the dot operator (.) when working with structures?

10. How do you define a pointer to a structure? How is it different from a pointer to a simple data

type?

11. Explain the concept of dynamic memory allocation in C and how it relates to data structures?

12. How do you free memory allocated dynamically in C to avoid memory leaks?

13. How is dynamic memory allocated in C? Explain the functions used for allocation.

14. What is the role of the malloc function, and how is it different from calloc and realloc?

15. What is the purpose of the sizeof operator when working with dynamic memory allocation?

PROGRAM 02:

1. What is pattern matching in the context of string processing?

2. How can you perform a basic pattern matching operation in C without using any built-in

functions or libraries?

3. How can you handle case-insensitive pattern matching when replacing strings in C?

4. What are the applications of pattern matching and string replacement in real-world

programming scenarios?

5. Explain the difference between string comparison using == and strcmp()

6. What is the significance of the null-terminator ('\0') in string comparison?

7. Can you compare two strings using the less than (<) or greater than (>) operators directly? Why or why not?

PROGRAM 03:

1. What is a stack, and how is it implemented using an array in C?

2. Explain the basic operations of a stack (push and pop) and how they are performed in an array based stack.

3. What is the role of the top pointer (or index) in an array-based stack? How do you initialize an empty stack using an array?

4. How do you check if a stack is empty or full when using an array-based implementation?

5. Discuss the advantages and limitations of using an array to implement a stack.

6. Can you implement a dynamic array-based stack that can resize itself when needed? If so, how?

7. Explain how to handle stack overflow and underflow in an array-based stack.8. How can you retrieve the top element of the stack without popping it?

9. Describe a real-world scenario where you might use an array-based stack in a C program.

10. Explain how to implement multiple stacks using a single array.

PROGRAM 04:

1. What is infix notation, and what is postfix notation?

2. Explain the process of converting an infix expression to postfix notation using a stack data

structure.

3. How do you handle operators of different precedence levels during the conversion process?

4. What should be done when encountering an open parenthesis "(" during the conversion? How

do you handle the close parenthesis ")" during the infix to postfix conversion?

5. What is the significance of the right-associative operators in the conversion process?

6. How can you handle errors or invalid expressions during the conversion process?

7. What are the applications or use cases for infix to postfix conversion in programming?

8. How does the infix to postfix conversion algorithm handle unary operators and functions?

9. Can you convert complex infix expressions with nested parentheses to postfix notation usingthe same algorithm?

10. What are some advantages of postfix notation over infix notation for expression evaluation?

PROGRAM 05:

Postfix Expression Evaluation:

1. Explain the process of evaluating a postfix expression.

2. How do you handle operands and operators in the postfix evaluation algorithm? Can you

provide a step-by-step example of evaluating a postfix expression?

3. What is the role of a stack in the postfix expression evaluation?

4. What happens if the postfix expression is invalid or incomplete?

5. Can you evaluate postfix expressions with multi-digit operands? If so, how?

6. How do you handle division by zero or other potential errors during postfix evaluation?

7. Discuss the advantages of postfix notation in mathematical expressions.

Tower of Hanoi:

8. What is the Tower of Hanoi problem, and what are its rules?

9. Explain the recursive approach to solving the Tower of Hanoi problem.

10. How many moves are required to solve the Tower of Hanoi problem with N disks?

11. How does the number of moves required to solve the Tower of Hanoi increase with the number

of disks?

12. Are there alternative methods to solve the Tower of Hanoi problem besides recursion?

PROGRAM 06:

1. What is a circular queue, and how does it differ from a regular queue?

2. Explain the concept of front and rear pointers in a circular queue.

3. How is memory allocated for a circular queue in an array-based implementation?

4. What is the maximum size (MAX) of the circular queue in this program?

5. Describe the basic operations of a circular queue, including enqueuing and dequeuing elements.

6. How do you determine if the circular queue is empty or full during enqueue and dequeue

operations?

7. What are the advantages of using a circular queue over a regular queue in certain scenarios?

8. What happens when you attempt to enqueue an element into a full circular queue?

9. How do you handle dequeuing from an empty circular queue?

10. Can you demonstrate the concept of circular queue traversal?

11. How can you implement additional operations like checking the size of the circular queue andclearing it?

12. What is the importance of using a circular queue in situations where data continuously cycles

through a fixed-size buffer?

13. Discuss potential issues or challenges that may arise when working with circular queues

PROGRAM 07:

1. What is a singly linked list, and how does it differ from other data structures like arrays?

2. Describe the structure of a node in a singly linked list.

3. How do you represent an empty singly linked list? How do you check if a linked list is empty

in C?

4. What is the significance of the "head" pointer in a singly linked list?

5. How can you insert a new node at the beginning of a singly linked list?

6. Describe the process of inserting a new node at the end of a singly linked list.

7. What is the difference between inserting a node in the middle of the list and at a specific

position in the list?

8. How do you delete a node from a singly linked list?

9. Discuss the importance of properly updating pointers when inserting or deleting nodes from a

linked list.

10. Explain how to traverse a singly linked list from the beginning to the end.

11. What is a singly linked list's linear data structure and how is it useful in problem-solving?

12. What is a self-referential structure, and how is it used in a singly linked list node?

13. Discuss potential issues or challenges that may arise when working with singly linked lists.

14. Can you provide an example of a real-world application where a singly linked list is used in C?

PROGRAM 08:

1. What is a doubly linked list, and how does it differ from a singly linked list?

2. Explain the structure of a doubly linked list node. What information does it typically contain?

3. Describe the advantages of using a doubly linked list in comparison to a singly linked list.

4. What is the purpose of having both "next" and "previous" pointers in a doubly linked list?

5. Explain the process of inserting a new node at the beginning of a doubly linked list.

6. How do you insert a new node at the end of a doubly linked list?

7. Discuss the steps to insert a new node at a specific position within a doubly linked list.

8. How can you delete a node from a doubly linked list?

9. What precautions should be taken when deleting nodes from a doubly linked list?

10. How do you traverse a doubly linked list in both forward and reverse directions?

PROGRAM 09:

1. What is a singly circular linked list, and how does it differ from a singly linked list and a

doubly circular linked list?

2. Describe the structure of a singly circular linked list node. What information does it typically

contain?

3. Explain the concept of a circular linked list. How does it differ from a linear linked list?

4. How does a singly circular linked list ensure that there is no "end" or "beginning" of the list?

5. What are the advantages of using a circular linked list in comparison to a linear linked list?

6. Explain the process of inserting a new node at the beginning of a singly circular linked list.

7. How do you insert a new node at the end of a singly circular linked list?

8. Discuss the steps to insert a new node at a specific position within a singly circular linked list.

9. How can you delete a node from a singly circular linked list?

10. What precautions should be taken when deleting nodes from a singly circular linked list?

11. How do you traverse a singly circular linked list? What is the exit condition for traversal?

12. What are the potential challenges or drawbacks of using a singly circular linked list in C?

13. How can you represent a polynomial using a linked list in C?

14. Explain the structure of a node in a linked list representation of a polynomial?

15. How do you handle sparse polynomials where many coefficients are zero?

PROGRAM 10:

1. What is a binary search tree (BST), and how is it structured?

2. Explain the fundamental properties of a BST that differentiate it from other binary tree

structures.

3. How does the BST property ensure efficient searching of elements in a BST?

4. What is the significance of the left and right sub trees in a BST?

5. Can a binary search tree contain duplicate values? Why or why not?

6. Describe the process of inserting a new element into a BST while maintaining the BST

property.

7. How do you delete a node from a BST while preserving the BST property?

8. What is the difference between in-order, pre-order, and post-order traversal of a BST, and

what are their use cases?

9. How do you find the minimum and maximum elements in a BST?

10. What is the height (depth) of a binary search tree, and how does it relate to the efficiency ofoperations?

PROGRAM 11:

1. What is a graph, and how is it represented using an adjacency matrix?

2. Explain the concept of directed and undirected graphs in the context of adjacency matrices.

3. How do you represent weighted edges in an adjacency matrix?

4. What is Breadth-First Search (BFS), and what is its primary purpose in graph traversal?

5. Explain the process of performing BFS on a graph using an adjacency matrix.

6. What data structure is commonly used to implement the BFS algorithm?

7. How do you find the shortest path between two nodes in an unweighted graph using BFS?

8. What is Depth-First Search (DFS), and how does it differ from BFS in terms of traversal

strategy?

9. Explain the process of performing DFS on a graph using an adjacency matrix.

10. What data structure is commonly used to implement the DFS algorithm?

11. Compare and contrast the time complexities of BFS and DFS on graphs represented by

adjacency matrices.

PROGRAM 12

1. What is hashing, and how does it relate to the mapping of keys to addresses?

2. Explain the concept of a hash function. What does it do, and why is it important in hashing?

3. What is a collision in the context of hashing, and why does it occur?

4. Can you provide examples of situations where collisions might occur in hash tables?

5. What is linear probing, and how does it work to resolve collisions in a hash table?

6. What is the basic idea behind linear probing when a collision occurs?

7. How do you search for a key in a hash table using linear probing?

8. When inserting a key-value pair with linear probing, what steps do you follow to find the

appropriate location in the hash table?

9. What are the advantages and disadvantages of using linear probing as a collision resolution

method?

10. Discuss potential scenarios in which linear probing may not be an ideal choice for collision

resolution.

11. Can you explain any issues or challenges that may arise when implementing a hash table with

linear probing?

12. What are the applications of hashing techniques with linear probing in real-world scenarios?