**School of Engineering & Technology**

**Department of Computer Science & Technology**



**Data Structures Lab**

**(CSP 242)**

**Lab File**

**(2023-2024)**

**for**

**B. Tech. (CSE)**

**2nd Semester**

**Submitted By:**

Your Name

B. Tech. CSE [2nd Semester]

Your Roll No

[Your Batch]

**Submitted To:**

Dr. Pushpendra K Rajput

Associate Professor,

Department of Computer Science & Engineering

School of Engineering & Technology

Sharda University

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial No.** | **Exp. No.** | **Description** | **Page No.** |
| **1. Dynamic Memory allocation and Recursion** | | |  |
| 1. | 1.1 | Create an integer array of user defined size n1 with dynamic memory allocation. Store data after reading from keyboard. Expand the size of array with n2. Read new values (n2 values from keyboard). Print state of array with all (n1+n2) values. | -- |
| 2. | 1.2 | Write a recursive function for Tower of Hanoi Problem. |  |
| 3. | 1.3 | Write a function to find the sum of all array elements using recursion. |  |
| 4. | 1.4 | Write a recursive function to print the reverse of a string. |  |
| 5. | 1.5 | Write a program to create a 2-D array using dynamic memory allocation. Also write the code to scan the input and display element of created array. |  |
| **2. Arrays** | | |  |
| 6. | 2.1 | Write a menu driven C program to implement array operations (Insertion, Deletion, and Searching) on sorted array. Create the array dynamically with initial size n. if the array is found full increase the size of array as double (2 times of existing size) and insert the element. |  |
| 7. | 2.2 | Write a menu driven program to perform the following operations on matrix.   * 1. Addition of two matrices.   2. Subtraction of two matrices.   3. Multiplication of two matrices.   4. Transpose of a matrix. |  |
| 8. | 2.3 | Write a program to store a 2-D matrix and find the sum of each row and column. |  |
| 9 | 2.4 | Write a function that calculates the sum of both the diagonals of a given matrix. Use call by reference to update the variables for storing results. |  |
| **3. Linear Link List Data Structure and its Applications** | | |  |
| 10 | 3.1 | Implement single Linked List data structure using array. Create all necessary functions to perform operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list. |  |
| 11 | 3.2 | Implement single Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list. |  |
|  | 3.3 | Add two polynomials using Linked List. |  |
| 12 | 3.4 | Implement doubly Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list. |  |
| 13 | 3.5 | Implement circular Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list. |  |
| **4. Stack Data Structure** | | |  |
| 14 | 4.1 | Using array and functions implement Stack and its operations like push, pop, peek. |  |
| 15 | 4.2 | Use the stack operations developed in Prob 1 and reverse a string using stack |  |
| 16 | 4.3 | Using array and functions implement two Stacks and its operations (push, pop, peek). |  |
| 17 | 4.4 | Implement stack operations using linear linked list. |  |
| **5. Queue Data Structure** | | |  |
| 18 | 5.1 | Using circular array and functions implement Queue data structure and its operations like insert, delete. |  |
| 19 | 5.2 | Implement Queue data structure using linked list and its operations (Enqueue, Dequeue, Display). |  |
| 20 | 5.3 | Check whether the string is palindrome or not using Stack and Queue. |  |
| 21 | 5.4 | Implement Double Ended Queue data structure using linked list.   * 1. Input Restricted   2. Output Restricted |  |
| 22 | 5.5 | Implement Priority Queue using array where the minimum element is having highest priority |  |
| 23 | 5.6 | Implement Priority Queue using Linked list where the priority is associated with each element. |  |
| **6. Trees** | | |  |
| 24 | 6.1 | Create a binary tree using an array/linked List. Write the functions to perform Preorder, Inorder, Postorder and Level-order Traversal of constructed tree. |  |
| 25 | 6.2 | Write a Menu driven program to perform the following operations on Binary Search Tree.   * 1. Insert   2. Search   3. Delete   4. Traversals      1. Inorder      2. Preorder      3. Postorder |  |
| 26 | 6.3 | Write a program to perform the following operations on Binary Search Tree   * 1. Find Minimum Element   2. Find Maximum Element |  |
| 27 | 6.4 | Write the program that reads the random sequence of integers and prints the sorted form of given data (ascending Order) using Binary Search Tree. |  |
| **7. Sorting algorithms** | | |  |
| 28 | 7.1 | Read the numbers from a text file sort them into an array using ‘*Insertion Sort’* algorithm and write back in another text file. |  |
| 29 | 7.2 | Read the numbers from a text file sort them into an array using ‘*Bubble Sort’* algorithm and write back in another text file. |  |
| 30 | 7.3 | Read the numbers from a text file and write a function to search an element using linear search. |  |
| 31 | 7.4 | Read the numbers from a text file sort them into an array using ‘*Selection Sort’* algorithm and write back in another text file. |  |
| 32 | 7.5 | Read the numbers from a text file where numbers are stored in random manner. Write a program to search an element in the data using Binary Search. (Note: we require sorted data for applying binary Search Algorithm. |  |
| **8. Graph** | | |  |
| 33. | 8.1 | Create a graph in the memory and write a choice based program to demonstrate following two graph traversal Algorithm   * 1. DFS (Depth First Search)   2. BFS (Breadth First Search) |  |

Experiment-01

**Title: 1- Array Operation on Sorted Array.**

**Objective:** To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 1.1: Create an integer array of user defined size n1 with dynamic memory allocation. Store data after reading from keyboard. Expand the size of array with n2. Read new values (n2 values from keyboard). Print state of array with all (n1+n2) values.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

int\* firstArray(int n1); //n1 = size of array 1.

int\* secondArray(int n2); //n2 = size of array 2.

void finalArray(int\* , int\* , int , int );

int main(){

int n1;

int n2;

printf("enter the size of array\_1= "); //taking size of array 1 as input from the user.

scanf(" %d" , &n1);

int \*one= firstArray(n1); //calling the function named as firstArray.

printf("enter the size of array\_2= "); //taking size of array 2 as input from the user.

scanf("%d" , &n2);

int \*two= secondArray(n2); //calling the function named as secondArray.

finalArray(one , two , n1 , n2); //calling the function named as finalArray.

free(one); //deallocating the memory used by malloc.

free(two);

return 0;

}

int \*firstArray(int n1 ){

int\* arr1;

arr1 = (int\*)malloc(n1 \* sizeof(int)); //allocating memory of specific size by using malloc.

for(int i=0; i<n1;i++){

printf("enter the element at %d index of array\_1= " , i );

scanf("%d" ,&arr1[i]); //taking input the elements of array 1.

}

return arr1;

}

int \*secondArray(int n2 ){

int\* arr2;

arr2 = (int\*)malloc(n2 \* sizeof(int)); //allocating memory of specific size by using malloc.

for(int i=0; i<n2;i++){

printf("enter the element at %d index of array\_2= " , i);

scanf("%d" ,&arr2[i]); //taking input the elements of array 2.

}

return arr2;

}

void finalArray(int\* one, int\* two , int n1 , int n2){

int\* arr = (int\*)malloc((n1+n2) \* sizeof(int));

for(int i= 0; i<n1; i++){

arr[i]= one[i];

}

for(int i= 0; i<n2; i++){

arr[n1+i]=two[i];

}

printf("the final array is ");

for(int i=0; i< (n1+n2);i++){

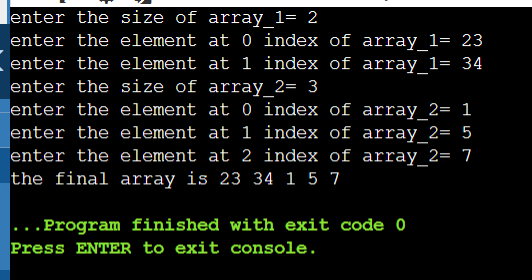
printf("%d " , arr[i]);

}

free(arr);

}

**Sample Output:**



Experiment-01

**Title: 1- Array Operation on Sorted Array.**

**Objective:** To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 1.2: Write a recursive function for Tower of Hanoi Problem.

**Source Code:**

#include <stdio.h>

void tower(int n, char from, char to, char help);

// n depicts the no. of discs ,

// "from" depicts the starting point,

// "to" depicts the destination

// and help is the auxiliary support.

int main(){

int n;

printf("enter the value of n: "); // taking input the no. of disc present.

scanf(" %d" ,&n);

char from, to , help;

printf("enter the starting point: "); //taking input the starting point.

scanf(" %c" , &from);

printf("enter the destination point: "); //taking input the ending point.

scanf(" %c" , &to);

printf("enter the name of auxiliary point: "); //taking input the help point.

scanf(" %c" , &help);

tower(n,from, to,help); //calling the function tower.

return 0;

}

void tower(int n, char from, char to, char help){

if(n==1){

printf(" %c -----> %c\n" , from, to); //base condition of the recursive function

}

else{

tower((n-1) , from,help , to);

printf(" %c -----> %c\n" , from , to); /\*condition which is going to be executed when number of

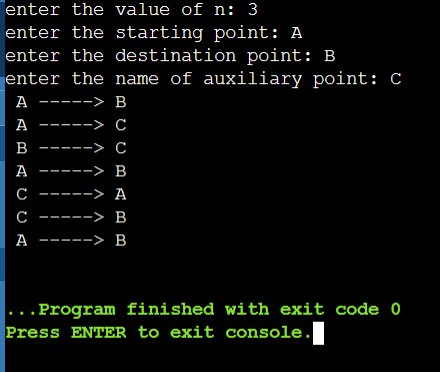
is not equal to 1\*/

tower((n-1), help , to , from);

}

}

**Sample Output:**



Experiment-01

**Title: 1- Array Operation on Sorted Array.**

**Objective:** To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 1.3: Write a function to find the sum of all array elements using recursion.

**Source Code:**

#include <stdio.h>

int sumArray(int \*, int);

int main()

{

int size;

int arr[5]= {1,4,2,5,2};

size= 5;

int \*ptr= &arr[0];

int sum;

sum= sumArray( &arr[0] , size);

printf("the sum of the element of array is %d" , sum);

return 0;

}

int sumArray( int \*ptr, int size){

if(size== 0){

return ptr[0];

}

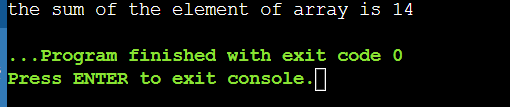
else{

return ptr[size ] + sumArray(ptr, size - 1);

}

}

**Sample Output:**



Experiment-01

**Title: 1- Array Operation on Sorted Array.**

**Objective:** To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 1.3: Write a recursive function to print the reverse of a string.

**Source Code:**

#include <stdio.h>

void reverse (char \*str);

int

main ()

{

char string[] = { "my name is aparna pandey" };

char \*str = &string[0];

reverse (str);

return 0;

}

void

reverse (char \*str)

{

if (\*str == '\0')

{

return;

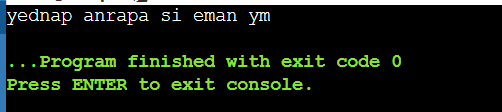
}

reverse(str + 1);

printf("%c", \*str);

}

**Sample Output:**



Experiment-01

**Title: 1- Array Operation on Sorted Array.**

**Objective:** To apply the concept of insertion, deletion, and binary search on a sorted array.

Problem – 1.3: Write a program to create a 2-D array using dynamic memory allocation. Also write the code to scan the input and display element of created array.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

void allocation\_of\_memory (int, int);

void input\_output (int \*\*p, int row, int column);

int

main ()

{

int row;

printf ("enter the number of rows: ");

scanf ("%d", &row);

int column;

printf ("enter the number of columns: ");

scanf ("%d", &column);

allocation\_of\_memory (row, column);

}

void allocation\_of\_memory (int row, int column)

{

int \*\*p = (int \*\*) malloc (row \* sizeof (int));

for (int i = 0; i < column; i++)

{

p[i] = (int \*) malloc (column \* sizeof (int));

}

input\_output (p, row, column);

}

void input\_output (int \*\*p, int row, int column)

{

printf ("Enter the elements of the array:\n");

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

scanf ("%d", &p[i][j]);

}

}

printf ("The elements of the array are:\n");

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

printf ("%d ", p[i][j]);

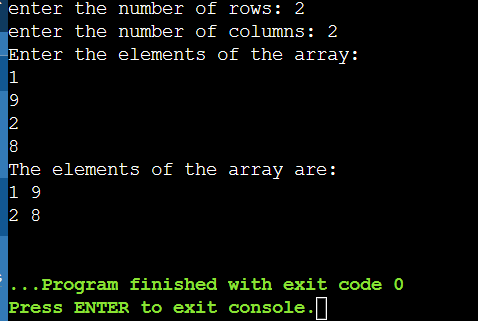
}

printf ("\n");

}

}

**Sample Output:**



Experiment-02

**Title: Arrays**

**Objective:** To learn and implement the concept of Array as ADT. Understanding of 2-D arrays.

Problem - 2.1: Write a menu driven C program to implement array operations (Insertion, Deletion, and Searching) on sorted array. Create the array dynamically with initial size n. if the array is found full increase the size of array as double (2 times of existing size) and insert the element.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

int \*resize (int \*array, int \*size)

{

\*size \*= 2;

array = (int \*) realloc (array, (\*size) \* sizeof (int));

return array;

}

void insert (int \*array, int \*count, int \*size, int element)

{

if (\*count == \*size)

{

array = resize (array, size);

}

int i = \*count - 1;

while (i >= 0 && array[i] > element)

{

array[i + 1] = array[i];

i--;

}

array[i + 1] = element;

(\*count)++;

}

void delete (int \*array, int \*count, int element)

{

int i = 0;

while (i < \*count && array[i] != element)

{

i++;

}

if (i < \*count)

{

while (i < \*count - 1)

{

array[i] = array[i + 1];

i++;

}

(\*count)--;

}

else

{

printf ("Element not found\n");

}

}

void search (int \*array, int count, int element)

{

int i = 0;

while (i < count && array[i] != element)

{

i++;

}

if (i < count)

{

printf ("Element found at position %d\n", i + 1);

}

else

{

printf ("Element not found\n");

}

}

void display (int array[], int n)

{

printf ("Sorted Array: ");

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

{

printf ("%d ", array[i]);

}

printf ("\n");

}

int main ()

{

int n;

printf ("Enter initial size of array: ");

scanf ("%d", &n);

int \*array = (int \*) malloc (n \* sizeof (int));

int count = 0;

int size = n;

int choice, element;

do

{

printf("\nMenu:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Search\n");

printf("4.display \n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf ("%d", &choice);

switch (choice)

{

case 1:

printf ("Enter element to insert: ");

scanf ("%d", &element);

insert (array, &count, &size, element);

break;

case 2:

printf ("Enter element to delete: ");

scanf ("%d", &element);

delete (array, &count, element);

break;

case 3:

printf ("Enter element to search: ");

scanf ("%d", &element);

search (array, count, element);

break;

case 4:

display (array, n);

break;

}

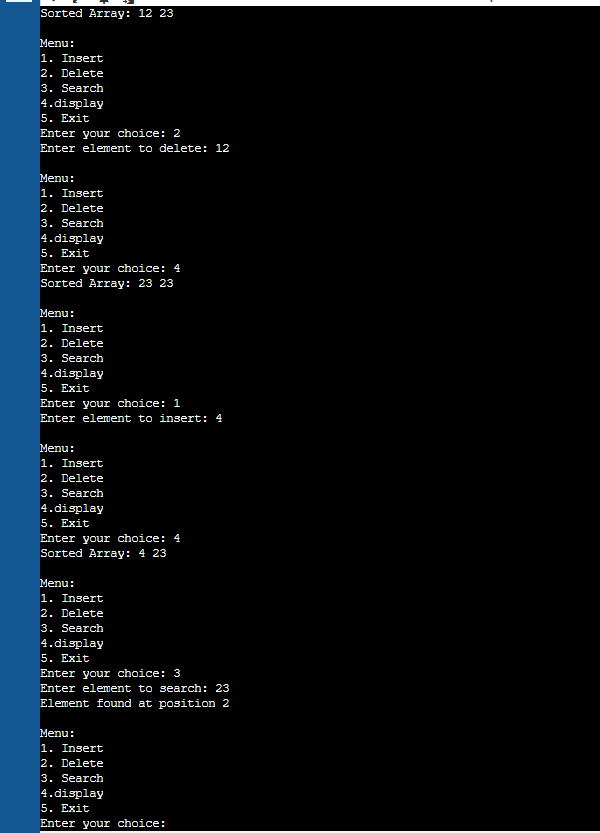
}

while (choice != 5);

free (array);

return 0;

}



**Sample Output:**

Experiment-02

**Title: Arrays**

**Objective:** To learn and implement the concept of Array as ADT. Understanding of 2-D arrays.

Problem - 2.2: Write a menu driven program to perform the following

operations on matrix.

a. Addition of two matrices.

b. Subtraction of two matrices.

c. Multiplication of two matrices.

d. Transpose of a matrix.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

// Function to add two 3x3 matrices

void add(int m[3][3], int n[3][3], int sum[3][3]) {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

sum[i][j] = m[i][j] + n[i][j];

}

void subtract(int m[3][3], int n[3][3], int result[3][3]) {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

result[i][j] = m[i][j] - n[i][j];

}

void multiply(int m[3][3], int n[3][3], int result[3][3]) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

result[i][j] = 0;

for (int k = 0; k < 3; k++)

result[i][j] += m[i][k] \* n[k][j];

}

}

}

// Function to find transpose of a 3x3 matrix

void transpose(int matrix[3][3], int trans[3][3]) {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

trans[i][j] = matrix[j][i];

}

// Function to display a 3x3 matrix

void display(int matrix[3][3]) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

printf("%d\t", matrix[i][j]);

printf("\n");

}

}

int main() {

int a[][3] = {{5, 6, 7}, {8, 9, 10}, {3, 1, 2}};

int b[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int c[3][3]; // Result matrix

printf("First Matrix:\n");

display(a);

printf("\nSecond Matrix:\n");

display(b);

int choice;

do {

printf("\nMenu:\n");

printf("1. Matrix Addition\n");

printf("2. Matrix Subtraction\n");

printf("3. Matrix Multiplication\n");

printf("4. Transpose of Matrix\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

add(a, b, c);

printf("Result of Addition:\n");

display(c);

break;

case 2:

subtract(a, b, c);

printf("Result of Subtraction:\n");

display(c);

break;

case 3:

multiply(a, b, c);

printf("Result of Multiplication:\n");

display(c);

break;

case 4:

transpose(a, c);

printf("Transpose of First Matrix:\n");

display(c);

break;

case 5:

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

exit(0);

default:

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

}

} while (1);

return 0;

}

|  |  |
| --- | --- |
|  |  |

**Sample Output:**

Experiment-02

**Title: Arrays**

**Objective:** To learn and implement the concept of Array as ADT. Understanding of 2-D arrays.

Problem - 2.3: Write a program to store a 2-D matrix and find the sum of each row and column.

**Source Code:**

#include <stdio.h>

#include<stdlib.h>

void input(int \*\*matrix, int r, int c );

void sumOfRows(int \*\*matrix , int r , int c);

void sumOfColumns(int \*\*matrix,int r, int c);

void output(int \*\*matrix, int r, int c);

int main()

{

int r,c;

printf("enter the no. of rows:");

scanf("%d" , &r);

printf("enter the no. of columns: ");

scanf("%d" , &c);

int \*\*p;

p= (int\*\*)malloc(r \* sizeof(int\*));

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

p[i]= (int\*)malloc(c\* sizeof( int));

}

input(p, r, c);

output(p, r,c);

sumOfRows(p, r,c);

sumOfColumns(p,r,c);

return 0;

}

void input(int \*\*matrix, int r, int c ){

printf("enter the elements of matrix:");

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

for(int j=0; j<c;j++){

scanf("%d" , &matrix[i][j]);

}

}

}

void output(int \*\*matrix, int r, int c){

printf("the matrix formed:\n");

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

for(int j=0; j<c;j++){

printf("%d\t" , matrix[i][j]);

}

printf("\n");

}

}

void sumOfRows(int \*\*matrix , int r , int c){

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

int sum=0;

for(int j=0; j<c;j++){

sum= sum+matrix[i][j];

}

printf("sum of row %d : %d\n" ,i+1, sum);

}

}

void sumOfColumns(int \*\*matrix,int r, int c){

for(int j=0; j<c;j++){

int sum=0;

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

sum= sum+matrix[i][j];

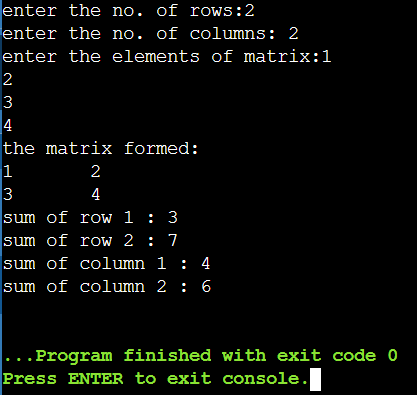
}

printf("sum of column %d : %d\n" , j+1, sum);

}

}

**Sample Output:**

Experiment-02

**Title: Arrays**

**Objective:** To learn and implement the concept of Array as ADT. Understanding of 2-D arrays.

Problem - 2.4: Write a function that calculates the sum of both the diagonals of a given matrix. Use call by reference to update the variables for storing results.

**Source Code:**

#include <stdio.h>

void sumOfDiagonals(int matrix[][100], int size, int \*sumDiagonal1, int \*sumDiagonal2) {

\*sumDiagonal1 = 0;

\*sumDiagonal2 = 0;

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

\*sumDiagonal1 += matrix[i][i];

\*sumDiagonal2 += matrix[i][size - 1 - i];

}

}

int main() {

int matrix[100][100];

int size;

printf("Enter the no. of elements in each row of the square matrix: ");

scanf("%d", &size);

printf("Enter the elements of the matrix:\n");

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

for (int j = 0; j < size; j++) {

scanf("%d", &matrix[i][j]);

}

}

int sumDiagonal1, sumDiagonal2;

sumOfDiagonals(matrix, size, &sumDiagonal1, &sumDiagonal2);

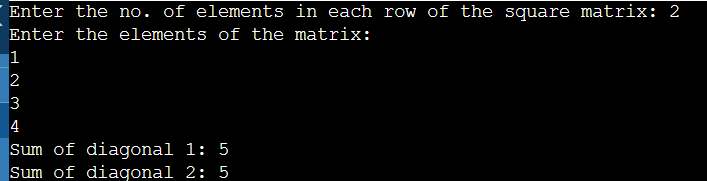
printf("Sum of diagonal 1: %d\n", sumDiagonal1);

printf("Sum of diagonal 2: %d\n", sumDiagonal2);

return 0;

}

**Sample Output:**



Experiment-03

**Title: Linear Link List Data Structure and its Applications**

**Objective:** To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.1: Implement single Linked List data structure using array. Create all necessary functions to perform operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

void traversing(int \*data, int \*link, int start);

int count(int \*data, int \*link, int start);

int search(int \*data, int \*link, int start);

void insert\_specific(int \*data, int \*link, int start, int avail, int loc, int item);

void insert\_at\_first(int \*data, int \*link, int \*start, int avail, int item);

void insert\_at\_last(int \*data, int \*link, int start, int avail, int item);

void delete\_specific(int \*data, int \*link, int \*start, int \*avail, int loc, int \*ploc);

void delete\_at\_first(int \*data, int \*link, int \*start, int \*avail);

void delete\_at\_last(int \*data, int \*link, int \*start, int \*avail);

void display(int \*data, int \*link, int start);

void search\_ploc(int \*data ,int\* link ,int start,int item ,int\* loc ,int\* ploc);

int main() {

int data[10], link[10];

int start = 0;

data[0] = 10;

link[0] = 4;

data[4] = 80;

link[4] = 6;

data[6] = 100;

link[6] = -1;

for (int i = 1; i < 10; i++) {

link[i] = i + 1;

}

link[9] = -1;

int avail = 1;

while (true) {

printf("1. Traverse the linked list.\n");

printf("2. Count the elements of the list.\n");

printf("3. Search for a given element.\n");

printf("4. Insert an element in the list.\n");

printf("5. Insert an element at the first node.\n");

printf("6. Insert an element at the last node.\n");

printf("7. Delete a specific element.\n");

printf("8. Delete the first element.\n");

printf("9. Delete the last element.\n");

printf("10. Search for ploc.\n");

printf("11. Display the list.\n");

printf("Make your choice: ");

int choice;

scanf("%d", &choice);

switch (choice) {

case 1: {

traversing(data, link, start);

break;

}

case 2: {

int total\_elements = count(data, link, start);

printf("Total elements: %d\n", total\_elements);

break;

}

case 3: {

int found\_at = search(data, link, start);

if (found\_at != -1) {

printf("Element is at index: %d\n", found\_at);

} else {

printf("Element not found!\n");

}

break;

}

case 4: {

int item, loc;

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

scanf("%d", &item);

printf("Enter the location: ");

scanf("%d", &loc);

insert\_specific(data, link, start, avail, loc, item);

break;

}

case 5: {

int item\_first;

printf("Enter the element you want to insert at the beginning: ");

scanf("%d", &item\_first);

insert\_at\_first(data, link, &start, avail, item\_first);

break;

}

case 6: {

int last\_item;

printf("Enter the element you want to insert at the end: ");

scanf("%d", &last\_item);

insert\_at\_last(data, link, start, avail, last\_item);

break;

}

case 7: {

int loc;

printf("Enter the location of node you want to delete from the list: ");

scanf("%d", &loc);

delete\_specific(data, link, &start, &avail, loc, NULL);

break;

}

case 8: {

delete\_at\_first(data, link, &start, &avail);

break;

}

case 9: {

delete\_at\_last(data, link, &start, &avail);

break;

}

case 10: {

int item;

printf("Enter the item to search for ploc: ");

scanf("%d", &item);

int loc = -1;

int ploc = -1;

search\_ploc(data ,link , start, item , &loc , &ploc);

if (ploc != -1) {

printf("ploc is %d\n", ploc);

} else {

printf("Item not found!\n");

}

break;

}

case 11:{

display(data, link, start);

break;

}

default: {

exit(0);

}

}

}

return 0;

}

void traversing(int \*data, int \*link, int start) {

int ptr = start;

while (ptr != -1) {

printf("%d ", data[ptr]);

ptr = link[ptr];

}

printf("\n");

}

int count(int \*data, int \*link, int start) {

int ptr = start;

int count = 0;

while (ptr != -1) {

count++;

ptr = link[ptr];

}

return count;

}

int search(int \*data, int \*link, int start) {

int element;

printf("Enter the element you need to search: ");

scanf("%d", &element);

int loc = start;

while (loc != -1) {

if (data[loc] == element) {

return loc;

}

loc = link[loc];

}

return -1; // Element not found

}

void insert\_at\_first(int \*data, int \*link, int \*start, int avail, int item) {

if (avail == -1) {

printf("Overflowed!\n");

return;

}

int new\_node = avail;

avail = link[avail];

data[new\_node] = item;

link[new\_node] = \*start;

\*start = new\_node;

printf("Element %d inserted at the beginning.\n", data[\*start]);

}

void insert\_at\_last(int \*data, int \*link, int start, int avail, int item) {

if (avail == -1) {

printf("Overflowed!\n");

return;

}

int new\_node = avail;

avail = link[avail];

data[new\_node] = item;

if (start == -1) {

start = new\_node;

link[start] = -1;

} else {

int ptr = start;

while (link[ptr] != -1) {

ptr = link[ptr];

}

link[ptr] = new\_node;

link[new\_node] = -1;

}

printf("Element %d inserted at the end.\n", data[new\_node]);

}

void insert\_specific(int \*data, int \*link, int start, int avail, int loc, int item) {

if (avail == -1) {

printf("Overflowed!\n");

return;

}

int new\_node = avail;

avail = link[avail];

data[new\_node] = item;

if (loc == -1) {

link[new\_node] = start;

start = new\_node;

} else {

link[new\_node] = link[loc];

link[loc] = new\_node;

}

printf("Element %d inserted at index %d.\n", data[new\_node], loc);

}

void delete\_specific(int \*data, int \*link, int \*start, int \*avail, int loc, int \*ploc) {

if (\*start == -1) {

printf("Underflowed!\n");

return;

}

if (\*ploc == -1) {

\*start = link[\*start];

} else {

link[\*ploc] = link[loc];

}

link[loc] = \*avail;

\*avail = loc;

printf("Location %d deleted.\n", loc);

}

void delete\_at\_first(int \*data, int \*link, int \*start, int \*avail) {

if (\*start == -1) {

printf("Underflowed!\n");

return;

}

int ptr = \*start;

\*start = link[\*start];

link[ptr] = \*avail;

\*avail = ptr;

printf("First node is deleted!\n");

}

void delete\_at\_last(int \*data, int \*link, int \*start, int \*avail) {

if (\*start == -1) {

printf("Underflowed!\n");

return;

}

int loc = \*start;

int ploc = -1;

while (link[loc] != -1) {

ploc = loc;

loc = link[loc];

}

if (ploc == -1) {

\*start = -1;

} else {

link[ploc] = -1;

}

link[loc] = \*avail;

\*avail = loc;

printf("Last node is deleted!\n");

}

void display(int \*data, int \*link, int start) {

int ptr = start;

while (ptr != -1) {

printf("%d ", data[ptr]);

ptr = link[ptr];

}

printf("\n");

}

void search\_ploc(int \*data ,int\* link ,int start,int item ,int\* loc ,int\* ploc){

\*ploc= -1;

\*loc= start;

while(\*loc!=-1){

if(data[\*loc]== item){

return;

}

else{

\*ploc= \*loc;

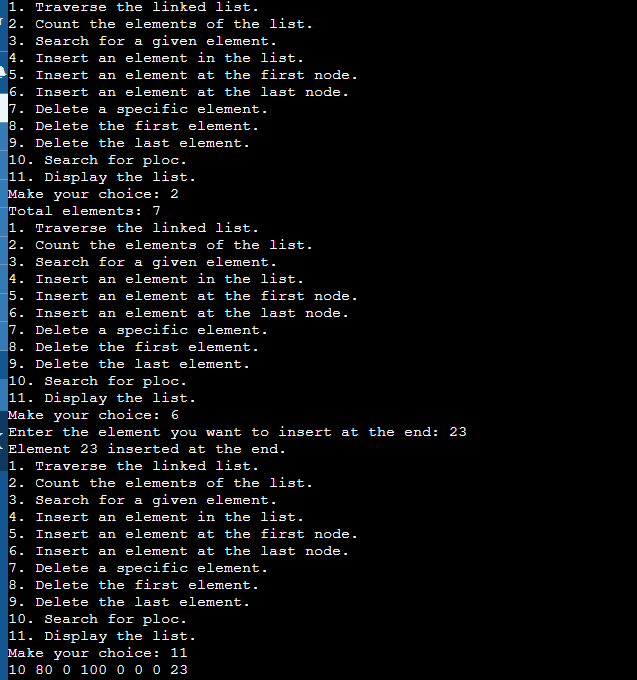
\*loc= link[\*loc];

}

}

}

**Sample Output:**



Experiment-03

**Title: Linear Link List Data Structure and its Applications**

**Objective:** To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.2: Implement single Linked List data structure and its operations like insert and delete in the beginning/end and n th position of the list, and display the items stored in the linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*link;

};

struct node \*create\_node() {

return (struct node\*)malloc(sizeof(struct node));

}

void traverse(struct node \*start) {

struct node \*ptr;

ptr = start;

while (ptr != NULL) {

printf("%d \t", ptr->data);

ptr = ptr->link;

}

printf("\n");

}

int count(struct node \*start) {

struct node \*ptr;

ptr = start;

int count = 0;

while (ptr != NULL) {

count++;

ptr = ptr->link;

}

return count;

}

void insert\_specific(struct node \*\*start, int item, int position) {

if(position <= 0 || position > count(\*start)+1){

printf("Invalid position\n");

return ;

}

struct node \*newNode=create\_node();

newNode->data=item;

if(position==1){

newNode->link=\*start;

\*start=newNode;

}

else{

struct node \*temp=\*start;

for(int i=1;i<position-1;i++){

temp=temp->link;

}

newNode->link=temp->link;

temp->link=newNode;

}

traverse(\*start);

}

void insert\_first(struct node \*\*start, int item) {

struct node \*newNode=create\_node();

newNode->data=item;

newNode->link=\*start;

\*start=newNode;

traverse(\*start);

}

void insert\_last(struct node \*\*start, int item) {

struct node \*newNode=create\_node();

newNode->data=item;

newNode->link=NULL;

if(\*start==NULL){

\*start=newNode;

}

else{

struct node \*temp=\*start;

while(temp->link!=NULL){

temp=temp->link;

}

temp->link=newNode;

}

traverse(\*start);

}

void delete\_specific(struct node \*\*start, int item) {

if (\*start == NULL) {

printf("The list is empty.\n");

return;

}

struct node \*temp = \*start;

struct node \*prev = NULL;

if (temp != NULL && temp->data == item) {

\*start = temp->link;

free(temp);

printf("Item deleted successfully.\n");

return;

}

while (temp != NULL && temp->data != item) {

prev = temp;

temp = temp->link;

}

if (temp == NULL) {

printf("Item not found in the list.\n");

return;

}

prev->link = temp->link;

free(temp);

printf("Item deleted successfully.\n");

}

void delete\_first(struct node \*\*start) {

if (\*start == NULL) {

printf("The list is empty.\n");

return;

}

struct node\* temp = \*start;

\*start = (\*start)->link;

free(temp);

printf("First item deleted successfully.\n");

}

void delete\_last(struct node \*\*start) {

if (\*start == NULL) {

printf("The list is empty.\n");

return;

}

struct node\* temp = \*start;

struct node\* prev = NULL;

while (temp->link != NULL) {

prev = temp;

temp = temp->link;

}

if (prev != NULL)

prev->link = NULL;

free(temp);

printf("Last item deleted successfully.\n");

}

int find(struct node \*start, int item) {

struct node \*ptr = start;

int index = 0;

while (ptr != NULL) {

if (ptr->data == item) {

return index;

}

ptr = ptr->link;

index++;

}

return -1;

}

void remove\_item(struct node \*\*start, int item) {

if (\*start == NULL) {

printf("The list is empty.\n");

return;

}

int position = find(\*start, item);

if (position == -1) {

printf("Item not found in the list.\n");

return;

}

delete\_specific(start, item);

printf("Item removed successfully.\n");

}

int main() {

struct node \*start = NULL;

int choice, item, position;

while(1){

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

printf("1. Display the list\n");

printf("2. Length of the list\n");

printf("3. Insert an element at a specific position\n");

printf("4. Insert an element at the beginning\n");

printf("5. Insert an element at the end\n");

printf("6. Delete a specific element\n");

printf("7. Delete the first element\n");

printf("8. Delete the last element\n");

printf("9. Find an element and print its index\n");

printf("10. Remove an element from the list\n");

printf("11. Exit\n");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch(choice){

case 1:

traverse(start);

break;

case 2:

int total\_elements = count(start);

printf("The total number of elements in the list is %d\n", total\_elements);

break;

case 3:

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

scanf("%d", &item);

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

scanf("%d", &position);

insert\_specific(&start, item, position);

break;

case 4:

printf("Enter the item to insert at the beginning: ");

scanf("%d", &item);

insert\_first(&start, item);

break;

case 5:

printf("Enter the item to insert at the end: ");

scanf("%d", &item);

insert\_last(&start, item);

break;

case 6:

printf("Enter the item to delete: ");

scanf("%d", &item);

delete\_specific(&start, item);

break;

case 7:

delete\_first(&start);

break;

case 8:

delete\_last(&start);

break;

case 9:

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

scanf("%d", &item);

int index = find(start, item);

if (index == -1) {

printf("Item not found in the list.\n");

} else {

printf("Item found at index %d.\n", index);

}

break;

case 10:

printf("Enter the element to remove from the list: ");

scanf("%d", &item);

remove\_item(&start, item);

break;

case 11:

exit(0);

default:

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

}

}

return 0;

}

**Sample Output:**

|  |  |
| --- | --- |
|  |  |

Experiment-03

**Title: Linear Link List Data Structure and its Applications**

**Objective:** To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.3: Add two polynomials using Linked List.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int coefficient;

int power;

struct node \*link;

};

struct node \*create\_node() {

return (struct node\*)malloc(sizeof(struct node));

}

void display\_polynomial(struct node \*p) {

printf("Resulting polynomial: ");

while (p != NULL) {

printf("%dx^%d ", p->coefficient, p->power);

p = p->link;

if(p!=NULL){

printf("+");

}

else{

printf("\n");

}

}

}

void insert\_term(struct node \*\*p, int coefficient, int power) {

struct node \*temp = create\_node();

temp->coefficient = coefficient;

temp->power = power;

temp->link = \*p;

\*p = temp;

printf("Data is inserted.\n");

}

struct node\* add\_polynomial(struct node \*p1, struct node \*p2) {

struct node \*result = NULL;

while (p1 != NULL && p2 != NULL) {

struct node \*temp = create\_node();

if (p1->power > p2->power) {

temp->power = p1->power;

temp->coefficient = p1->coefficient;

p1 = p1->link;

} else if (p1->power < p2->power) {

temp->power = p2->power;

temp->coefficient = p2->coefficient;

p2 = p2->link;

} else {

temp->coefficient = p1->coefficient + p2->coefficient;

temp->power = p1->power;

p1 = p1->link;

p2 = p2->link;

}

temp->link = result;

result = temp;

}

while (p1 != NULL) {

struct node \*temp = create\_node();

temp->coefficient = p1->coefficient;

temp->power = p1->power;

temp->link = result;

result = temp;

p1 = p1->link;

}

while (p2 != NULL) {

struct node \*temp = create\_node();

temp->coefficient = p2->coefficient;

temp->power = p2->power;

temp->link = result;

result = temp;

p2 = p2->link;

}

return result;

}

int main() {

struct node \*p1, \*p2, \*r;

p1 = p2 = r = NULL;

int coefficient;

int power;

int n;

printf("Enter the number of terms in polynomial\_1: ");

scanf("%d", &n);

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

printf("Enter the coefficient: ");

scanf("%d", &coefficient);

printf("Enter the power: ");

scanf("%d", &power);

insert\_term(&p1, coefficient, power);

}

int coefficient\_;

int power\_;

int n2;

printf("Enter the number of terms in polynomial\_2: ");

scanf("%d", &n2);

for(int i=0; i<n2; i++){

printf("Enter the coefficient: ");

scanf("%d", &coefficient\_);

printf("Enter the power: ");

scanf("%d", &power\_);

insert\_term(&p2, coefficient\_, power\_);

}

printf("Adding polynomials\n");

r = add\_polynomial(p1, p2);

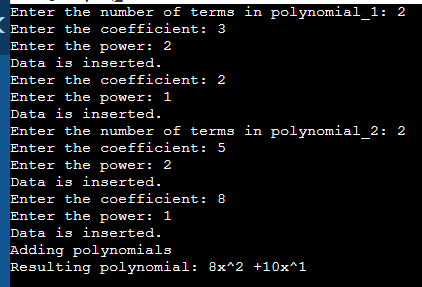
display\_polynomial(r);

return 0;

}

**Sample Output:**

­­



Experiment-03

**Title: Linear Link List Data Structure and its Applications**

**Objective:** To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.4: Implement doubly Linked List data structure and its

operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* head = NULL;

// Function to insert a node at the beginning of the list

void insertAtFirst(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->prev = NULL;

if (head == NULL) {

newNode->next = NULL;

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

printf("Node inserted successfully at the beginning.\n");

}

// Function to insert a node at the end of the list

void insertAtLast(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if (head == NULL) {

newNode->prev = NULL;

head = newNode;

} else {

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

printf("Node inserted successfully at the end.\n");

}

// Function to insert a node at a specific position in the list

void insertAtPosition(int value, int position) {

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

if(newNode==NULL){

printf("Memory allocation failed\n");

return ;

}

int count=1;

struct Node \*temp=head,\*prev=NULL;

while(temp!=NULL && count<position){

prev=temp;

temp=temp->next;

count++;

}

if(count!=position){

printf("Invalid position\n");

free(newNode);

return ;

}

newNode->data=value;

if(prev==NULL){

newNode->prev=NULL;

newNode->next=head;

head=newNode;

}

else{

newNode->prev=prev;

newNode->next=prev->next;

prev->next=newNode;

if(newNode->next!=NULL)

newNode->next->prev=newNode;

}

printf("Node inserted successfully at position %d.\n", position);

}

// Function to delete a node from the list by its position

void deleteByPosition(int position) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

if (position == 1) {

head = temp->next;

if (head != NULL)

head->prev = NULL;

free(temp);

printf("Node deleted successfully from position %d.\n", position);

return;

}

int count = 1;

while (temp != NULL && count < position) {

temp = temp->next;

count++;

}

if (temp == NULL || count != position) {

printf("Invalid position.\n");

return;

}

temp->prev->next = temp->next;

if (temp->next != NULL)

temp->next->prev = temp->prev;

free(temp);

printf("Node deleted successfully from position %d.\n", position);

}

// Function to delete the first node from the list

void deleteFirstNode() {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

head = temp->next;

if (head != NULL)

head->prev = NULL;

free(temp);

printf("First node deleted successfully.\n");

}

// Function to delete the last node from the list

void deleteLastNode() {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

if (temp == head) {

head = NULL;

free(temp);

printf("Last node deleted successfully.\n");

return;

}

temp->prev->next = NULL;

free(temp);

printf("Last node deleted successfully.\n");

}

// Function to search for an element in the list

void searchElement(int value) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

int position = 1, found = 0;

while (temp != NULL) {

if (temp->data == value) {

found = 1;

break;

}

temp = temp->next;

position++;

}

if (found)

printf("Element %d found at position %d.\n", value, position);

else

printf("Element %d not found in the list.\n", value);

}

// Function to count the length of the list

int countListLength() {

int count = 0;

struct Node\* temp = head;

while (temp != NULL) {

count++;

temp = temp->next;

}

return count;

}

// Function to display the elements of the list

void displayList() {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

struct Node\* temp = head;

printf("Elements of the list: ");

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

int main() {

int choice, value, position;

do {

printf("\n--- Doubly Linked List Menu ---\n");

printf("1. Insert at beginning\n");

printf("2. Insert at end\n");

printf("3. Insert at a specific position\n");

printf("4. Delete by position\n");

printf("5. Delete first node\n");

printf("6. Delete last node\n");

printf("7. Search element\n");

printf("8. Count list length\n");

printf("9. Display list\n");

printf("0. Exit\n");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nEnter the value to insert: ");

scanf("%d", &value);

insertAtFirst(value);

break;

case 2:

printf("\nEnter the value to insert: ");

scanf("%d", &value);

insertAtLast(value);

break;

case 3:

printf("\nEnter the value to insert: ");

scanf("%d", &value);

printf("Enter the position: ");

scanf("%d", &position);

insertAtPosition(value, position);

break;

case 4:

if (head == NULL) {

printf("The list is empty.\n");

break;

}

displayList();

printf("\nEnter the position of the node to delete: ");

scanf("%d", &position);

deleteByPosition(position);

break;

case 5:

deleteFirstNode();

break;

case 6:

deleteLastNode();

break;

case 7:

if (head == NULL) {

printf("The list is empty.\n");

break;

}

displayList();

printf("\nEnter the element to search: ");

scanf("%d", &value);

searchElement(value);

break;

case 8:

printf("Length of the list is %d\n", countListLength());

break;

case 9:

displayList();

break;

case 0:

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

exit(0);

default:

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

}

} while (1);

return 0;

}

|  |  |
| --- | --- |
|  |  |

**Sample Output:**

Experiment-03

**Title: Linear Link List Data Structure and its Applications**

**Objective:** To implement linear linked list data structure in C using structures, pointers, and dynamic memory allocation.

Problem - 3.5: Implement circular Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

void insertAtBeginning(struct Node \*\*head\_ref, int data);

void insertAtEnd(struct Node \*\*head\_ref, int data);

void insertAtNthPosition(struct Node \*\*head\_ref, int data, int position);

void deleteAtBeginning(struct Node \*\*head\_ref);

void deleteAtEnd(struct Node \*\*head\_ref);

void deleteAtNthPosition(struct Node \*\*head\_ref, int position);

void display(struct Node \*head);

int main() {

struct Node \*head = NULL;

int choice, data, position;

while (1) {

// Menu

printf("\nCircular Linked List Operations:\n");

printf("1. Insert at beginning\n");

printf("2. Insert at end\n");

printf("3. Insert at nth position\n");

printf("4. Delete at beginning\n");

printf("5. Delete at end\n");

printf("6. Delete at nth position\n");

printf("7. Display\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert at: ");

scanf("%d", &position);

insertAtNthPosition(&head, data, position);

break;

case 4:

deleteAtBeginning(&head);

break;

case 5:

deleteAtEnd(&head);

break;

case 6:

printf("Enter position to delete: ");

scanf("%d", &position);

deleteAtNthPosition(&head, position);

break;

case 7:

display(head);

break;

case 8:

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

exit(0);

default:

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

}

}

return 0;

}

void insertAtBeginning(struct Node \*\*head\_ref, int data) {

struct Node \*new\_node = (struct Node \*)malloc(sizeof(struct Node));

struct Node \*temp = \*head\_ref;

new\_node->data = data;

new\_node->next = new\_node;

if (\*head\_ref != NULL) {

while (temp->next != \*head\_ref) {

temp = temp->next;

}

temp->next = new\_node;

new\_node->next = \*head\_ref;

} else {

new\_node->next = new\_node;

}

\*head\_ref = new\_node;

}

void insertAtEnd(struct Node \*\*head\_ref, int data) {

struct Node \*new\_node = (struct Node \*)malloc(sizeof(struct Node));

struct Node \*temp = \*head\_ref;

new\_node->data = data;

new\_node->next = new\_node;

if (\*head\_ref != NULL) {

while (temp->next != \*head\_ref) {

temp = temp->next;

}

temp->next = new\_node;

new\_node->next = \*head\_ref;

} else {

new\_node->next = new\_node;

\*head\_ref = new\_node;

}

}

void insertAtNthPosition(struct Node \*\*head\_ref, int data, int position) {

if (position == 1) {

insertAtBeginning(head\_ref, data);

return;

}

struct Node \*new\_node = (struct Node \*)malloc(sizeof(struct Node));

struct Node \*temp = \*head\_ref;

new\_node->data = data;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position!\n");

return;

}

new\_node->next = temp->next;

temp->next = new\_node;

}

void deleteAtBeginning(struct Node \*\*head\_ref) {

if (\*head\_ref == NULL) {

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

return;

}

struct Node \*temp = \*head\_ref;

if (temp->next == \*head\_ref) {

\*head\_ref = NULL;

free(temp);

} else {

while (temp->next != \*head\_ref) {

temp = temp->next;

}

temp->next = (\*head\_ref)->next;

free(\*head\_ref);

\*head\_ref = temp->next;

}

}

void deleteAtEnd(struct Node \*\*head\_ref) {

if (\*head\_ref == NULL) {

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

return;

}

struct Node \*temp = \*head\_ref, \*prev;

if (temp->next == \*head\_ref) {

\*head\_ref = NULL;

free(temp);

} else {

while (temp->next != \*head\_ref) {

prev = temp;

temp = temp->next;

}

prev->next = temp->next;

free(temp);

}

}

void deleteAtNthPosition(struct Node \*\*head\_ref, int position) {

if (\*head\_ref == NULL) {

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

return;

}

struct Node \*temp = \*head\_ref, \*prev;

if (position == 1) {

deleteAtBeginning(head\_ref);

return;

}

for (int i = 1; i < position && temp != NULL; i++) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position!\n");

return;

}

prev->next = temp->next;

free(temp);

}

void display(struct Node \*head) {

if (head == NULL) {

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

return;

}

struct Node \*temp = head;

printf("Circular Linked List: ");

do {

printf("%d ", temp->data);

temp = temp->next;

} while (temp != head);

printf("\n");

}

Sample Output:

|  |  |
| --- | --- |
|  |  |

Experiment-04

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Stack operations and applications of Stack

Problem - 4.1: Using array and functions implement Stack and its operations like push, pop, peek.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void push(int \*stack , int \*top ,int item){

if(isfull(stack , &top)){

printf("Overflow!");

return ;

}

\*top= \*top+1;

stack[\*top]= item;

}

int pop(int \*stack , int \*top){

if(isempty(stack , &top)){

printf("underflow");

return;

}

int item = stack[\*top];

\*top = \*top-1;

return item;

}

int peek(int \*stack , int \*top){

if(isempty(stack , &top)){

printf("underflow!");

}

return stack[\*top];

}

void print\_stack(int \*stack , int top){

printf("\n current stack is \n");

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

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

}

}

int isempty(int \*stack , int \*top){

if(\*top==-1){

return 1;

}

else{

return 0;

}

}

int isfull(int \*stack , int \*top){

if(\*top== MAX-1){

return 1;

}

else{

return 0 ;

}

}

int main(){

int stack[MAX] , top = -1;

int choice , item;

while(1){

printf("1.push\n");

printf("2.pop\n ");

printf("3.Peek\n ");

printf("4.EXIT\n");

printf("make your choice: ");

scanf("%d" , &choice);

switch(choice){

case 1:

printf("Enter the item you want to insert: ");

scanf("%d" , &item);

push(stack, &top , item);

print\_stack(stack , top);

break;

case 2:

item = pop(stack , &top );

printf("\ndeleted item is %d\n" , item);

print\_stack(stack , top);

break;

case 3:

item= peek(stack , &top);

printf("the top- most item is %d\n" , item);

break;

case 4:

exit(0);

default:

printf("invalid choice! \n");

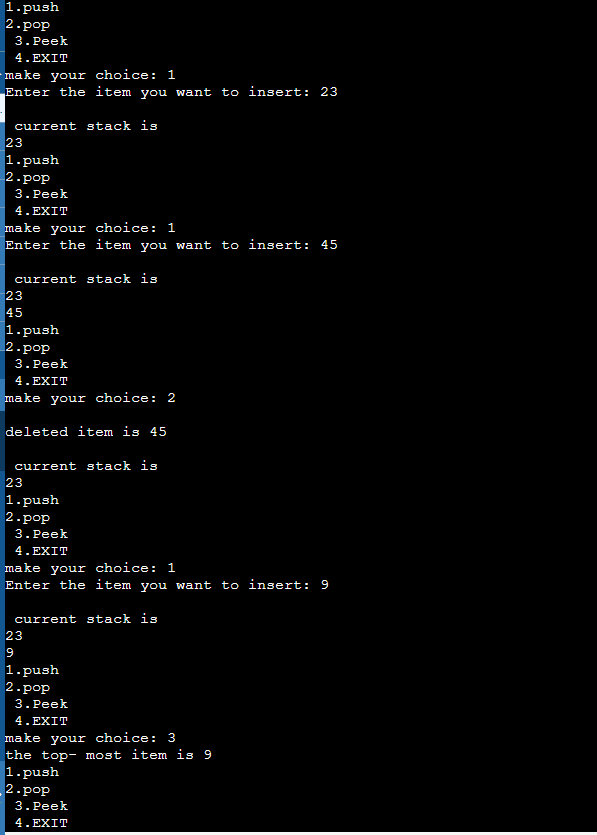
break;

}

}

}

**Sample Output:**



Experiment-04

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Stack operations and applications of Stack

Problem - 4.2: Use the stack operations developed in Prob 1 and reverse a string using stack

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

char data[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*stack);

int isEmpty(struct Stack \*stack);

int isFull(struct Stack \*stack);

void push(struct Stack \*stack, char item);

char pop(struct Stack \*stack);

char peek(struct Stack \*stack);

void display(struct Stack \*stack);

void reverseString(char \*str);

int main() {

struct Stack stack;

initialize(&stack);

int choice;

char item, str[MAX\_SIZE];

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. Reverse a string\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to push: ");

scanf(" %c", &item);

push(&stack, item);

break;

case 2:

if (!isEmpty(&stack)) {

printf("Popped element: %c\n", pop(&stack));

} else {

printf("Stack is empty!\n");

}

break;

case 3:

if (!isEmpty(&stack)) {

printf("Top element: %c\n", peek(&stack));

} else {

printf("Stack is empty!\n");

}

break;

case 4:

display(&stack);

break;

case 5:

printf("Enter a string to reverse: ");

scanf(" %[^\n]", str);

reverseString(str);

printf("Reversed string: %s\n", str);

break;

case 6:

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

exit(0);

default:

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

}

}

return 0;

}

void initialize(struct Stack \*stack) {

stack->top = -1;

}

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

int isFull(struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

void push(struct Stack \*stack, char item) {

if (isFull(stack)) {

printf("Stack overflow!\n");

return;

}

stack->data[++stack->top] = item;

}

char pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack underflow!\n");

exit(1);

}

return stack->data[stack->top--];

}

char peek(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty!\n");

exit(1);

}

return stack->data[stack->top];

}

void display(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty!\n");

return;

}

printf("Stack: ");

for (int i = 0; i <= stack->top; i++) {

printf("%c ", stack->data[i]);

}

printf("\n");

}

void reverseString(char \*str) {

struct Stack stack;

initialize(&stack);

for (int i = 0; str[i] != '\0'; i++) {

push(&stack, str[i]);

}

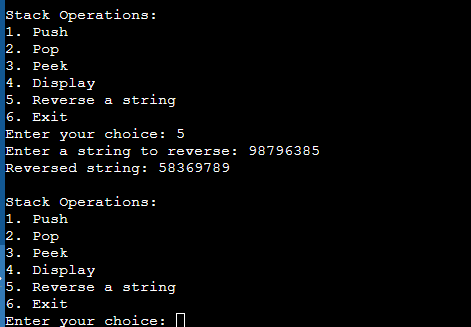
for (int i = 0; str[i] != '\0'; i++) {

str[i] = pop(&stack);

}

}

**Sample Output:**

****

Experiment-04

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Stack operations and applications of Stack

Problem - 4.3: Using array and functions implement two Stacks and its operations (push, pop, peek).

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

// Structure for a stack

struct Stack {

int data[MAX\_SIZE];

int top1;

int top2;

};

void initialize(struct Stack \*stack);

int isFull(struct Stack \*stack, int stackNumber);

int isEmpty(struct Stack \*stack, int stackNumber);

void push(struct Stack \*stack, int stackNumber, int item);

int pop(struct Stack \*stack, int stackNumber);

int peek(struct Stack \*stack, int stackNumber);

void display(struct Stack \*stack, int stackNumber);

int main() {

struct Stack stack;

initialize(&stack);

int choice, stackNumber, item;

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. Exit\n");

printf("Enter stack number (1 or 2): ");

scanf("%d", &stackNumber);

if (stackNumber != 1 && stackNumber != 2) {

printf("Invalid stack number! Please enter 1 or 2.\n");

continue;

}

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to push: ");

scanf("%d", &item);

push(&stack, stackNumber, item);

break;

case 2:

if (!isEmpty(&stack, stackNumber)) {

printf("Popped element: %d\n", pop(&stack, stackNumber));

} else {

printf("Stack is empty!\n");

}

break;

case 3:

if (!isEmpty(&stack, stackNumber)) {

printf("Top element: %d\n", peek(&stack, stackNumber));

} else {

printf("Stack is empty!\n");

}

break;

case 4:

display(&stack, stackNumber);

break;

case 5:

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

exit(0);

default:

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

}

}

return 0;

}

void initialize(struct Stack \*stack) {

stack->top1 = -1;

stack->top2 = MAX\_SIZE;

}

int isFull(struct Stack \*stack, int stackNumber) {

if (stackNumber == 1) {

return stack->top1 == stack->top2 - 1;

} else {

return stack->top2 == stack->top1 + 1;

}

}

int isEmpty(struct Stack \*stack, int stackNumber) {

if (stackNumber == 1) {

return stack->top1 == -1;

} else {

return stack->top2 == MAX\_SIZE;

}

}

void push(struct Stack \*stack, int stackNumber, int item) {

if (isFull(stack, stackNumber)) {

printf("Stack %d overflow!\n", stackNumber);

return;

}

if (stackNumber == 1) {

stack->data[++stack->top1] = item;

} else {

stack->data[--stack->top2] = item;

}

}

int pop(struct Stack \*stack, int stackNumber) {

if (isEmpty(stack, stackNumber)) {

printf("Stack %d underflow!\n", stackNumber);

exit(1);

}

if (stackNumber == 1) {

return stack->data[stack->top1--];

} else {

return stack->data[stack->top2++];

}

}

int peek(struct Stack \*stack, int stackNumber) {

if (isEmpty(stack, stackNumber)) {

printf("Stack %d is empty!\n", stackNumber);

exit(1);

}

if (stackNumber == 1) {

return stack->data[stack->top1];

} else {

return stack->data[stack->top2];

}

}

void display(struct Stack \*stack, int stackNumber) {

if (isEmpty(stack, stackNumber)) {

printf("Stack %d is empty!\n", stackNumber);

return;

}

printf("Stack %d: ", stackNumber);

if (stackNumber == 1) {

for (int i = stack->top1; i >= 0; i--) {

printf("%d ", stack->data[i]);

}

} else {

for (int i = stack->top2; i < MAX\_SIZE; i++) {

printf("%d ", stack->data[i]);

}

}

printf("\n");

}

|  |  |
| --- | --- |
|  |  |

**Sample Output:**

Experiment-04

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Stack operations and applications of Stack

Problem - 4.4: Implement stack operations using linear linked list.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct Node {

int data;

struct Node \*next;

};

struct Stack {

struct Node \*top;

};

void initialize(struct Stack \*stack);

bool isEmpty(struct Stack \*stack);

void push(struct Stack \*stack, int data);

int pop(struct Stack \*stack);

int peek(struct Stack \*stack);

void display(struct Stack \*stack);

int main() {

struct Stack stack;

initialize(&stack);

int choice, data;

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. IsEmpty\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to push: ");

scanf("%d", &data);

push(&stack, data);

break;

case 2:

if (!isEmpty(&stack)) {

printf("Popped element: %d\n", pop(&stack));

} else {

printf("Stack is empty!\n");

}

break;

case 3:

if (!isEmpty(&stack)) {

printf("Top element: %d\n", peek(&stack));

} else {

printf("Stack is empty!\n");

}

break;

case 4:

display(&stack);

break;

case 5:

if (isEmpty(&stack)) {

printf("Stack is empty\n");

} else {

printf("Stack is not empty\n");

}

break;

case 6:

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

exit(0);

default:

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

}

}

return 0;

}

void initialize(struct Stack \*stack) {

stack->top = NULL;

}

bool isEmpty(struct Stack \*stack) {

return stack->top == NULL;

}

void push(struct Stack \*stack, int data) {

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = stack->top;

stack->top = newNode;

}

int pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack underflow!\n");

exit(1);

}

struct Node \*temp = stack->top;

int data = temp->data;

stack->top = temp->next;

free(temp);

return data;

}

int peek(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty!\n");

exit(1);

}

return stack->top->data;

}

void display(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty!\n");

return;

}

struct Node \*current = stack->top;

printf("Stack: ");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

|  |  |
| --- | --- |
|  |  |

**Sample Output:**

Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.1: Using circular array and functions implement Queue data structure and its operations like insert, delete.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 10

struct Queue {

int data[MAX\_SIZE];

int front, rear;

int size;

};

void initialize(struct Queue \*queue);

int isFull(struct Queue \*queue);

int isEmpty(struct Queue \*queue);

void enqueue(struct Queue \*queue, int item);

int dequeue(struct Queue \*queue);

void display(struct Queue \*queue);

int main() {

struct Queue queue;

initialize(&queue);

int choice, item;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (!isFull(&queue)) {

printf("Enter element to enqueue: ");

scanf("%d", &item);

enqueue(&queue, item);

} else {

printf("Queue is full!\n");

}

break;

case 2:

if (!isEmpty(&queue)) {

printf("Dequeued element: %d\n", dequeue(&queue));

} else {

printf("Queue is empty!\n");

}

break;

case 3:

display(&queue);

break;

case 4:

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

exit(0);

default:

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

}

}

return 0;

}

void initialize(struct Queue \*queue) {

queue->front = queue->rear = -1;

queue->size = 0;

}

int isFull(struct Queue \*queue) {

return (queue->size == MAX\_SIZE);

}

int isEmpty(struct Queue \*queue) {

return (queue->size == 0);

}

void enqueue(struct Queue \*queue, int item) {

if (isFull(queue)) {

printf("Queue overflow!\n");

return;

}

if (isEmpty(queue)) {

queue->front = queue->rear = 0;

} else {

queue->rear = (queue->rear + 1) % MAX\_SIZE;

}

queue->data[queue->rear] = item;

queue->size++;

}

int dequeue(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue underflow!\n");

exit(1);

}

int item = queue->data[queue->front];

if (queue->front == queue->rear) {

queue->front = queue->rear = -1;

} else {

queue->front = (queue->front + 1) % MAX\_SIZE;

}

queue->size--;

return item;

}

void display(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return;

}

printf("Queue: ");

int i = queue->front;

do {

printf("%d ", queue->data[i]);

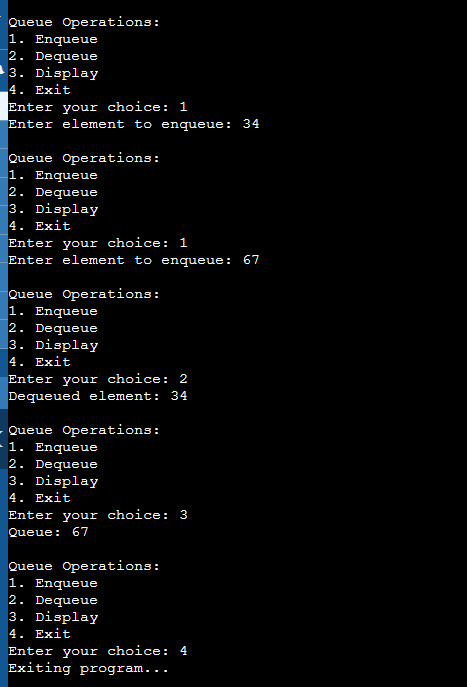
i = (i + 1) % MAX\_SIZE;

} while (i != (queue->rear + 1) % MAX\_SIZE);

printf("\n");

}

**Sample Output:**



Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.2: Implement Queue data structure using linked list and its operations (Enqueue, Dequeue, Display).

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

struct Queue {

struct Node \*front;

struct Node \*rear;

};

void initialize(struct Queue \*queue);

void enqueue(struct Queue \*queue, int data);

int dequeue(struct Queue \*queue);

int isEmpty(struct Queue \*queue);

void display(struct Queue \*queue);

int main() {

struct Queue queue;

initialize(&queue);

int choice, data;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to enqueue: ");

scanf("%d", &data);

enqueue(&queue, data);

break;

case 2:

if (!isEmpty(&queue)) {

printf("Dequeued element: %d\n", dequeue(&queue));

} else {

printf("Queue is empty!\n");

}

break;

case 3:

display(&queue);

break;

case 4:

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

exit(0);

default:

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

}

}

return 0;

}

void initialize(struct Queue \*queue) {

queue->front = NULL;

queue->rear = NULL;

}

void enqueue(struct Queue \*queue, int data) {

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

if (queue->rear == NULL) {

queue->front = newNode;

queue->rear = newNode;

} else {

queue->rear->next = newNode;

queue->rear = newNode;

}

}

int dequeue(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue underflow!\n");

exit(1);

}

int data = queue->front->data;

struct Node \*temp = queue->front;

queue->front = queue->front->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return data;

}

int isEmpty(struct Queue \*queue) {

return queue->front == NULL;

}

void display(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return;

}

struct Node \*current = queue->front;

printf("Queue: ");

while (current != NULL) {

printf("%d ", current->data);

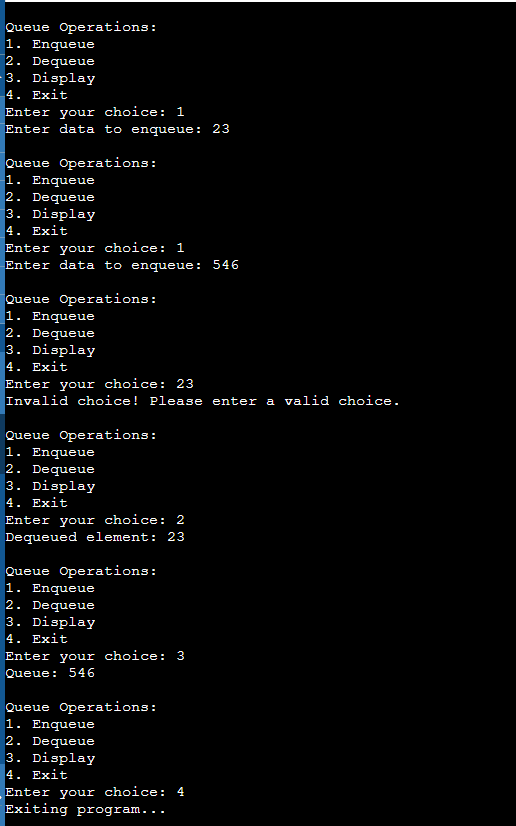
current = current->next;

}

printf("\n");

}

**Sample Output:**



Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.3 Check whether the string is palindrome or not using Stack and Queue.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#define MAX\_SIZE 100

// Node structure for stack and queue

struct Node {

char data;

struct Node\* next;

};

// Structure for stack

struct Stack {

struct Node\* top;

};

// Structure for queue

struct Queue {

struct Node\* front;

struct Node\* rear;

};

// Function prototypes

void initializeStack(struct Stack\* stack);

void push(struct Stack\* stack, char data);

char pop(struct Stack\* stack);

bool isStackEmpty(struct Stack\* stack);

void initializeQueue(struct Queue\* queue);

void enqueue(struct Queue\* queue, char data);

char dequeue(struct Queue\* queue);

bool isQueueEmpty(struct Queue\* queue);

bool isPalindrome(char\* str);

int main() {

char str[MAX\_SIZE];

// Read string from user

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

// Remove newline character from fgets

size\_t len = strlen(str);

if (len > 0 && str[len - 1] == '\n') {

str[len - 1] = '\0';

}

if (isPalindrome(str)) {

printf("The string is a palindrome.\n");

} else {

printf("The string is not a palindrome.\n");

}

return 0;

}

// Function to initialize the stack

void initializeStack(struct Stack\* stack) {

stack->top = NULL;

}

// Function to push an element onto the stack

void push(struct Stack\* stack, char data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = stack->top;

stack->top = newNode;

}

// Function to pop an element from the stack

char pop(struct Stack\* stack) {

if (isStackEmpty(stack)) {

printf("Stack underflow!\n");

exit(1);

}

struct Node\* temp = stack->top;

char data = temp->data;

stack->top = temp->next;

free(temp);

return data;

}

// Function to check if the stack is empty

bool isStackEmpty(struct Stack\* stack) {

return stack->top == NULL;

}

// start the queue

void initializeQueue(struct Queue\* queue) {

queue->front = NULL;

queue->rear = NULL;

}

// Function to enqueue an element into the queue

void enqueue(struct Queue\* queue, char data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

if (queue->rear == NULL) {

queue->front = newNode;

queue->rear = newNode;

} else {

queue->rear->next = newNode;

queue->rear = newNode;

}

}

// Function to dequeue an element from the queue

char dequeue(struct Queue\* queue) {

if (isQueueEmpty(queue)) {

printf("Queue underflow!\n");

exit(1);

}

struct Node\* temp = queue->front;

char data = temp->data;

queue->front = temp->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return data;

}

// to check if the queue is empty

bool isQueueEmpty(struct Queue\* queue) {

return queue->front == NULL;

}

// check if a string is a palindrome using stack and queue

bool isPalindrome(char\* str) {

struct Stack stack;

struct Queue queue;

initializeStack(&stack);

initializeQueue(&queue);

// Push each character onto the stack and enqueue into the queue

for (int i = 0; str[i] != '\0'; i++) {

char ch = str[i];

if (ch != ' ') {

push(&stack, ch);

enqueue(&queue, ch);

}

}

// Pop and dequeue characters from stack and queue and compare

while (!isStackEmpty(&stack) && !isQueueEmpty(&queue)) {

char stackChar = pop(&stack);

char queueChar = dequeue(&queue);

if (stackChar != queueChar) {

return false;

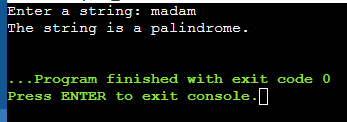
}

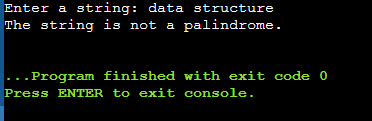
}

return true;

}

**Sample Output:**





Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.4: Implement Double Ended Queue data structure using linked list.

a. Input Restricted

b. Output Restricted

**Source Code:**

// input restricted queue

#include <stdio.h>

void deleteLastElement(int \*queue, int \*front, int \*rear, int maxSize) {

if (\*front == -1) {

printf("Queue is empty\n");

return;

}

if (\*front == \*rear) {

\*front = -1;

\*rear = -1;

} else if (\*rear == 0) {

\*rear = maxSize - 1;

} else {

\*rear = \*rear - 1;

}

traverse\_circular\_queue(queue, \*front, \*rear, maxSize);

}

void traverse\_circular\_queue(int queue[], int front, int rear, int size) {

if (front == -1) {

printf("\nQueue is empty\n");

return;

}

printf("\nQueue elements: ");

if (rear >= front) {

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

} else {

for (int i = front; i < size; i++) {

printf("%d ", queue[i]);

}

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

printf("%d ", queue[i]);

}

}

printf("\n");

}

void enqueue\_rear(int \*queue, int \*front, int \*rear, int item, int max\_size) {

if ((\*front == 0 && \*rear == max\_size - 1) || (\*front == \*rear + 1)) {

printf("OVERFLOW!");

return;

}

if (\*rear == max\_size - 1) {

\*rear = 0;

} else if (\*rear == -1 && \*front == -1) {

\*front = \*rear = 0;

queue[\*rear] = item;

} else {

\*rear += 1;

queue[\*rear] = item;

}

printf("Item inserted successfully.");

traverse\_circular\_queue(queue, \*front, \*rear, max\_size);

}

void dequeue\_front(int \*queue , int front, int rear , int item , int max\_size ){

if(front==-1){

printf("underflowed");

return;

}

if(front==max\_size-1){

front = front+1;

}

else if(front==rear){

front= rear= -1;

}

else{

front+=1;

}

traverse\_circular\_queue(queue , front , rear , max\_size);

}

int main(){

int max\_size;

int queue[max\_size] ;

int front= -1;

int rear=-1;

max\_size= 5 ;

int \*queue\_= &queue[0];

int choice;

while(1){

printf("1.insertion\n");

printf("2.deletion\n");

printf("3. delete at last.\n");

printf("make your choice: ");

scanf("%d" , &choice);

switch(choice){

case 1:

int item\_insert;

printf("enter the item you want to insert= ");

scanf("%d" , &item\_insert);

enqueue\_rear(queue\_ , &front , &rear , item\_insert , max\_size);

traverse\_circular\_queue(queue\_,front,rear,max\_size);

break;

case 2:

int item\_delete;

printf("enter the item you want to delete= ");

scanf("%d" , &item\_delete);

dequeue\_front(queue\_ , front , rear , item\_delete , max\_size);

break;

case 3:

deleteLastElement(queue\_,&front, &rear, max\_size);

break;

default:

printf("invalid input");

}

}

}

//OUTPUT RESTRICTED QUEUE

#include <stdio.h>

void enque\_front(int \*queue, int \*front, int rear, int item ,int max\_size) {

if ((\*front == 0 && rear == max\_size - 1) || (\*front == rear + 1)) {

printf("Overflow!");

return;

}

if (\*front == -1) {

\*front = 0;

}

if (\*front == 0) {

\*front = max\_size - 1;

} else {

\*front = (\*front - 1 + max\_size) % max\_size;

}

queue[\*front] = item;

traverse\_circular\_queue(queue, \*front, rear, max\_size);

}

void traverse\_circular\_queue(int queue[], int front, int rear, int size) {

if (front == -1) {

printf("\nQueue is empty\n");

return;

}

printf("\nQueue elements: ");

if (rear >= front) {

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

} else {

for (int i = front; i < size; i++) {

printf("%d ", queue[i]);

}

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

printf("%d ", queue[i]);

}

}

printf("\n");

}

void enqueue\_rear(int \*queue, int \*front, int \*rear, int item, int max\_size) {

if ((\*front == 0 && \*rear == max\_size - 1) || (\*front == \*rear + 1)) {

printf("OVERFLOW!");

return;

}

if (\*rear == max\_size - 1) {

\*rear = 0;

} else if (\*rear == -1 && \*front == -1) {

\*front = \*rear = 0;

queue[\*rear] = item;

} else {

\*rear += 1;

queue[\*rear] = item;

}

printf("Item inserted successfully.");

traverse\_circular\_queue(queue, \*front, \*rear, max\_size);

}

void dequeue\_front(int \*queue , int front, int rear , int item , int max\_size ){

if(front==-1){

printf("underflowed");

return;

}

if(front==max\_size-1){

front = front+1;

}

else if(front==rear){

front= rear= -1;

}

else{

front+=1;

}

traverse\_circular\_queue(queue , front , rear , max\_size);

}

int main(){

int max\_size;

int queue[max\_size] ;

int front= -1;

int rear=-1;

max\_size= 5 ;

int \*queue\_= &queue[0];

int choice;

while(1){

printf("1.insertion\n");

printf("2.deletion\n");

printf("3.insert at front\n");

printf("make your choice: ");

scanf("%d" , &choice);

switch(choice){

case 1:

int item\_insert;

printf("enter the item you want to insert= ");

scanf("%d" , &item\_insert);

enqueue\_rear(queue\_ , &front , &rear , item\_insert , max\_size);

traverse\_circular\_queue(queue\_,front,rear,max\_size);

break;

case 2:

int item\_delete;

printf("enter the item you want to delete= ");

scanf("%d" , &item\_delete);

dequeue\_front(queue\_ , front , rear , item\_delete , max\_size);

break;

case 3:

int item\_insert\_front;

printf("enter the item you want to insert at front= ");

scanf("%d" , &item\_insert\_front);

enque\_front(queue\_ , &front , rear , item\_insert\_front , max\_size);

break;

default:

printf("invalid input");

}

}

}

**Sample Output:**

|  |  |
| --- | --- |
| INPUT RESTRICTED QUEUE OUTPUT | OUTPUT RESTRICTED QUEUE OUTPUT |
|  |  |

Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.5: Implement Priority Queue using array where the minimum element is having highest priority

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

struct PriorityQueue {

int data[MAX\_SIZE];

int size;

};

void initializePriorityQueue(struct PriorityQueue \*pq);

void insert(struct PriorityQueue \*pq, int element);

int removeMin(struct PriorityQueue \*pq);

int isEmpty(struct PriorityQueue \*pq);

int isFull(struct PriorityQueue \*pq);

int main() {

struct PriorityQueue pq;

initializePriorityQueue(&pq);

insert(&pq, 10);

insert(&pq, 5);

insert(&pq, 20);

insert(&pq, 3);

while (!isEmpty(&pq)) {

printf("Minimum element: %d\n", removeMin(&pq));

}

return 0;

}

void initializePriorityQueue(struct PriorityQueue \*pq) {

pq->size = 0;

}

void insert(struct PriorityQueue \*pq, int element) {

if (isFull(pq)) {

printf("Priority queue overflow!\n");

exit(1);

}

int i = pq->size;

while (i > 0 && element < pq->data[i - 1]) {

pq->data[i] = pq->data[i - 1];

i--;

}

pq->data[i] = element;

pq->size++;

}

int removeMin(struct PriorityQueue \*pq) {

if (isEmpty(pq)) {

printf("Priority queue underflow!\n");

exit(1);

}

int min = pq->data[0];

for (int i = 0; i < pq->size - 1; i++) {

pq->data[i] = pq->data[i + 1];

}

pq->size--;

return min;

}

int isEmpty(struct PriorityQueue \*pq) {

return pq->size == 0;

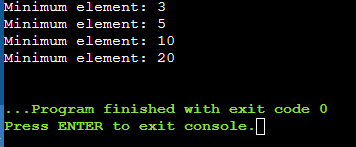
}

int isFull(struct PriorityQueue \*pq) {

return pq->size == MAX\_SIZE;

}

**Sample Output:**



Experiment-05

**Title: Stack Data Structures**

**Objective:** To demonstrate use of arrays and linked list to implement Queue operations and types of Queues.

Problem - 5.6: Implement Priority Queue using Linked list where the priority is associated with each element.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

int priority;

struct Node\* next;

};

struct PriorityQueue {

struct Node\* front;

};

void initializePriorityQueue(struct PriorityQueue \*pq);

void insert(struct PriorityQueue \*pq, int data, int priority);

int removeMin(struct PriorityQueue \*pq);

int isEmpty(struct PriorityQueue \*pq);

int main() {

struct PriorityQueue pq;

initializePriorityQueue(&pq);

insert(&pq, 910, 2);

insert(&pq, 4, 1);

insert(&pq, 1720, 3);

insert(&pq, 137, 1);

while (!isEmpty(&pq)) {

printf("Minimum element: %d\n", removeMin(&pq));

}

return 0;

}

void initializePriorityQueue(struct PriorityQueue \*pq) {

pq->front = NULL;

}

void insert(struct PriorityQueue \*pq, int data, int priority) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->priority = priority;

newNode->next = NULL;

if (isEmpty(pq) || priority < pq->front->priority) {

newNode->next = pq->front;

pq->front = newNode;

} else {

struct Node\* current = pq->front;

while (current->next != NULL && current->next->priority <= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

}

int removeMin(struct PriorityQueue \*pq) {

if (isEmpty(pq)) {

printf("Priority queue underflow!\n");

exit(1);

}

struct Node\* temp = pq->front;

int minData = temp->data;

pq->front = pq->front->next;

free(temp);

return minData;

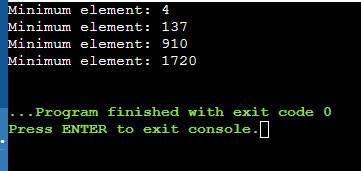
}

int isEmpty(struct PriorityQueue \*pq) {

return pq->front == NULL;

}

**Sample Output:**



Experiment-06

**Title: Trees**

**Objective:** To demonstrate the creation of a binary tree using arrays/linked lists and working with tree traversal and heap sorting algorithms.

Problem - 6.1:

**Source Code:**

**Sample Output:**

Experiment-06

**Title: Trees**

**Objective:** To demonstrate the creation of a binary tree using arrays/linked lists and working with tree traversal and heap sorting algorithms.

Problem - 6.2:

**Source Code:**

**Sample Output:**

Experiment-06

**Title: Trees**

**Objective:** To demonstrate the creation of a binary tree using arrays/linked lists and working with tree traversal and heap sorting algorithms.

Problem - 6.3:

**Source Code:**

**Sample Output:**

Experiment-06

**Title: Trees**

**Objective:** To demonstrate the creation of a binary tree using arrays/linked lists and working with tree traversal and heap sorting algorithms.

Problem - 6.4:

**Source Code:**

**Sample Output:**

Experiment-07

**Title: Searching and Sorting**

**Objective:** To implement various searching & sorting algorithms.

Problem - 7.1:

**Source Code:**

**Sample Output:**

Experiment-07

**Title: Searching and Sorting**

**Objective:** To implement various searching & sorting algorithms.

Problem - 7.2:

**Source Code:**

**Sample Output:**

Experiment-07

**Title: Searching and Sorting**

**Objective:** To implement various searching & sorting algorithms.

Problem - 7.3:

**Source Code:**

**Sample Output:**

Experiment-07

**Title: Searching and Sorting**

**Objective:** To implement various searching & sorting algorithms.

Problem - 7.4:

**Source Code:**

**Sample Output:**

Experiment-07

**Title: Searching and Sorting**

**Objective:** To implement various searching & sorting algorithms.

Problem - 7.5:

**Source Code:**

**Sample Output:**

Experiment-08

**Title: Graph Data Structure**

**Objective:** To demonstrate the creation of graph using adjacency Matrix/Adjacency Lists and working with graph traversal algorithms.

Problem - 8.1:

**Source Code:**

<Paste your code with single line spacing, Font: Courier New, Font Size: 12pt>

#include<stdio.h>

#include<-->

**Sample Output:**

<Paste captured screenshot>