INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| Assignment No. | Assignment Name | Page No. | Signature |
| 1. | Assignment - 1 | 2 - 15 |  |
| 2. | Assignment – 2 | 16 – 23 |  |
| 3. | Assignment – 3 | 24 – 39 |  |
| 4. | Assignment – 4 | 40 – 49 |  |
| 5. | Assignment – 5 | 50 – 58 |  |
| 6. | Assignment - 6 | 59 - 72 |  |

Data Structure lab assignment - 1

* **Problem No: 1**
* **Problem Statement:** Write a C program to print an array.
* **Source Code:**

#include<stdio.h>

int main () {

int n, i;

printf("Enter the length of the array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the Array-->\n");

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

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

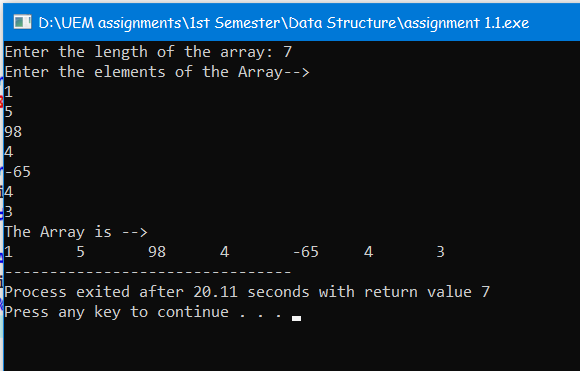
printf("The Array is -->\n");

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

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

}

* **Output:**

****

* **Problem No: 2**
* **Problem Statement:** Write a C program to check whether a given string is Palindrome or not.
* **Source Code:**

#include<stdio.h>

#include<string.h>

int main() {

char str[100];

int l = 0, h;

printf("Enter a string: ");

gets(str);

h = strlen(str) - 1;

while (h > l) {

if (str[l++] != str[h--]) {

printf("%s is not a palindrome\n", str);

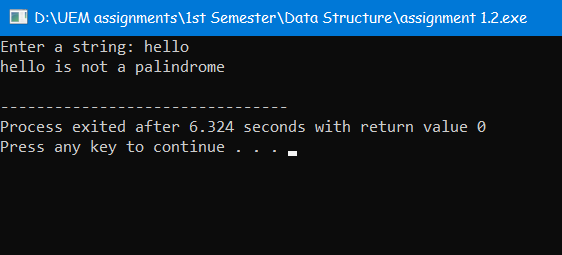
return 0;

}

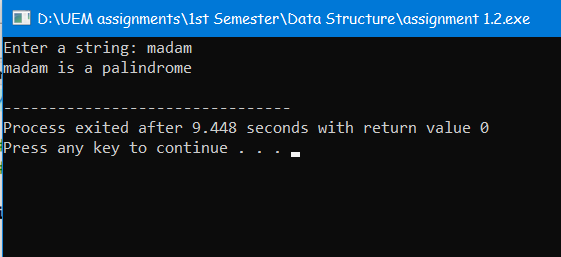
}

printf("%s is a palindrome\n", str);

return 0;

****}

**Output:**

****

* **Problem No: 3**
* **Problem Statement:** Write a C program to convert temperature from degree Centigrade to Fahrenheit.
* **Source Code:**

#include<stdio.h>

int main () {

int tc, tf;

printf("Enter the temperature in celcius: ");

scanf("%d", &tc);

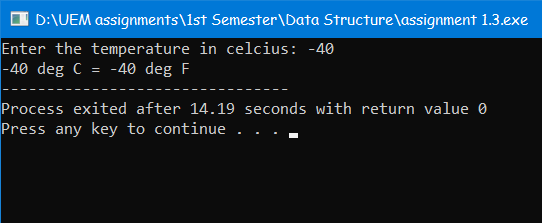
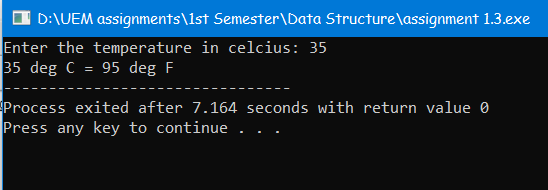
tf = (tc \* 9 / 5) + 32;

printf("%d deg C = %d deg F", tc, tf);

return 0;

}

* **Output:**

****

* **Problem No: 4**
* **Problem Statement:** Write a C program to sort an array.
* **Source Code:**

#include <stdio.h>

int main() {

int i, j, temp, len;

printf("Enter the length the array: ");

scanf("%d", &len);

int arr[len];

printf("Enter the elemets \n");

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

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

printf("The array before sort are given below \n");

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

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

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

for (j = i + 1; j < len; j++)

if (arr[i] > arr[j]) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

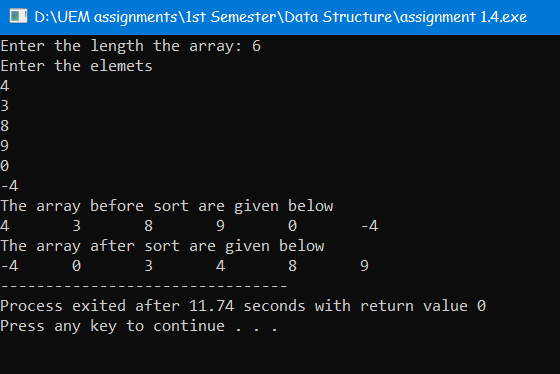
}

printf("\nThe array after sort are given below \n");

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

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

return 0;

}****

* **Output:**
* **Problem No: 5**
* **Problem Statement:** Write a C program to print the largest and second largest element of the array.
* **Source Code:**

#include <stdio.h>

int main() {

int n, max, max2, i, has\_max2 = 0;

printf("Enter the length of the array \n");

scanf("%d", &n);

int arr[n];

printf("Enter the elements \n");

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

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

printf ("The array is->\n");

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

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

max = arr[0];

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

if (max < arr[i]) {

max2 = max;

max = arr[i];

}

}

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

if (arr[i] < max) {

if (!has\_max2) {

has\_max2 = 1;

max2 = arr[i];

}

else if (arr[i] > max2)

max2 = arr[i];

}

}

if (has\_max2 == 1)

printf("\nLargest number = %d\n2nd Largest number = %d", max, max2);

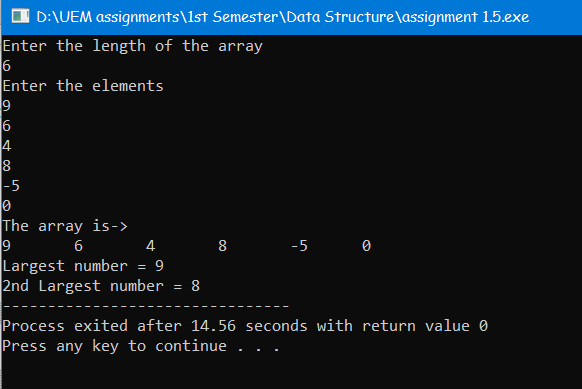
else

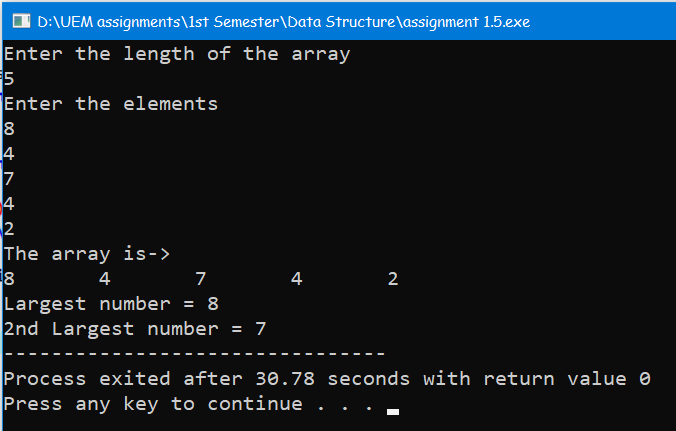
printf("\nAll values are identical to %d", max);

return 0;

}

* **Output:**

****

****

* **Problem No: 6**
* **Problem Statement:** Write a C program to display Fibonacci series.
* **Source Code:**

#include <stdio.h>

int main() {

int num, i, t1 = 0, t2 = 1, next = t1 + t2;

printf("Enter the terms of Fibonacci Series: ");

scanf("%d", &num);

printf("Fibonacci series-->\n");

printf("%d\t%d\t", t1, t2);

for (i = 2; i < num; i++) {

printf("%d\t", next);

t1 = t2;

t2 = next;

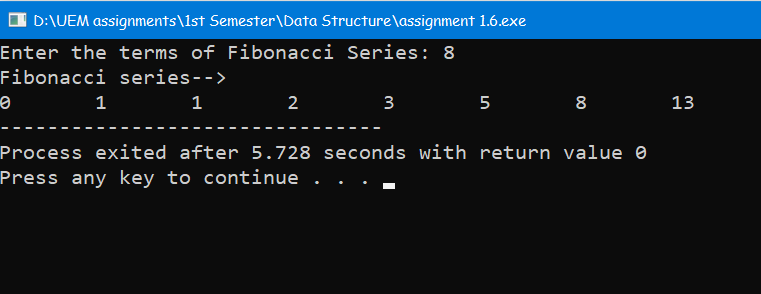
next = t1 + t2;

}

return 0;

}

* **Output:**

****

* **Problem No: 7**
* **Problem Statement:** Write a program that reads two 2D metrices from the console, verifies if metrics multiplication is possible or not. Then multiplies the metrices and prints the 3rd metrics.
* **Source Code:**

#include<stdio.h>

int main() {

int row1, row2, column1, column2, i, j, k;

printf("Enter the row and column of the 1st Matrix-->\n");

printf("Row: ");

scanf("%d", &row1);

printf("Column: ");

scanf("%d", &column1);

printf("Enter the row and column of the 2nd Matrix-->\n");

printf("Row: ");

scanf("%d", &row2);

printf("Column: ");

scanf("%d", &column2);

if (column1 != row2) {

printf("1st matrix columns is not equal to 2nd matrix row.\nMultiplication Can't possible.");

return 0;

}

int matrix1 [row1][column1], matrix2 [row2][column2], result [row1][column2];

printf("Enter the elements of 1st Matrix-->\n");

for (i = 0; i < row1; ++i)

for (j = 0; j < column1; ++j) {

printf("Enter element at [%d] [%d]: ", i + 1, j + 1);

scanf("%d", &matrix1[i][j]);

}

printf("Enter the elements of 2nd Matrix-->\n");

for (i = 0; i < row2; ++i)

for (j = 0; j < column2; ++j) {

printf("Enter element at [%d] [%d]: ", i + 1, j + 1);

scanf("%d", &matrix2[i][j]);

}

for (i = 0; i < row1; ++i)

for (j = 0; j < column2; ++j)

result[i][j] = 0;

for (i = 0; i < row1; ++i)

for (j = 0; j < column2; ++j)

for (k = 0; k < column1; ++k)

result[i][j] += matrix1[i][k] \* matrix2[k][j];

printf("Multiplication of two matrices is-->\n");

for (i = 0; i < row1; i++) {

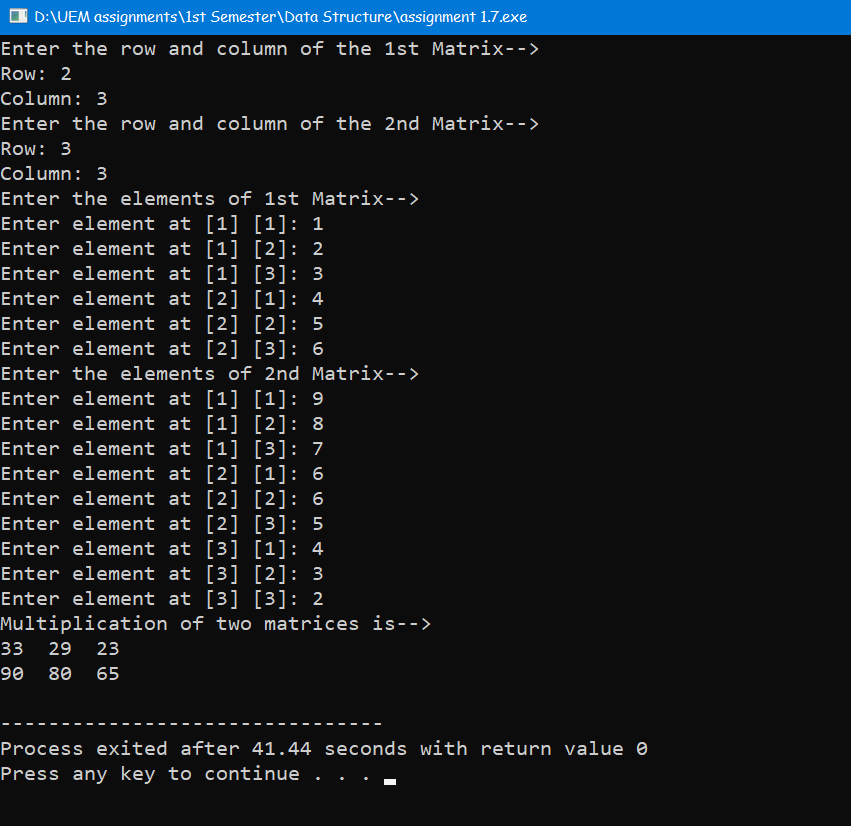
for (j = 0; j < column2; j++)

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

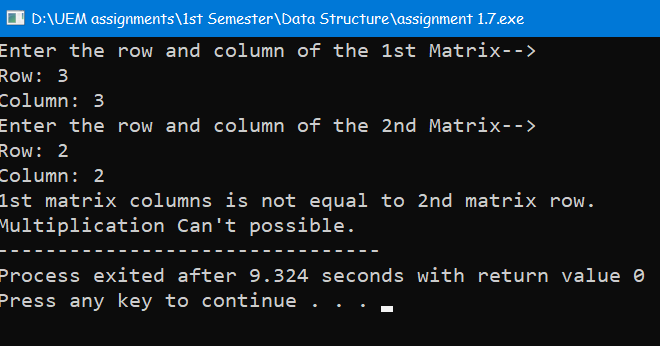
printf("\n");

}

return 0;

****}

* **Output:**

****

* **Problem No: 8**
* **Problem Statement:** Write a program that reads a 2D metrics and checks if the metrics is a symmetric metrics or not.
* **Source Code:**

#include<stdio.h>

int main() {

int row, column, i, j, flag = 0;

printf("Enter the row and column of the Matrix-->\n");

printf("Row: ");

scanf("%d", &row);

printf("Column: ");

scanf("%d", &column);

int matrix [row][column];

printf("Enter the elements of the Matrix-->\n");

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

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

printf("Enter element at [%d] [%d]: ", i + 1, j + 1);

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

}

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

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

if (matrix [j][i] != matrix [i][j]) {

flag = 1;

break;

}

if (flag == 0)

printf("The matrix is a symmetric matrix.");

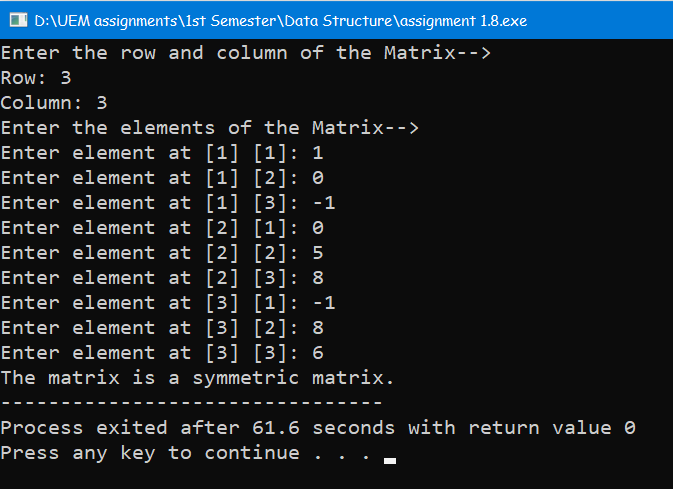
else

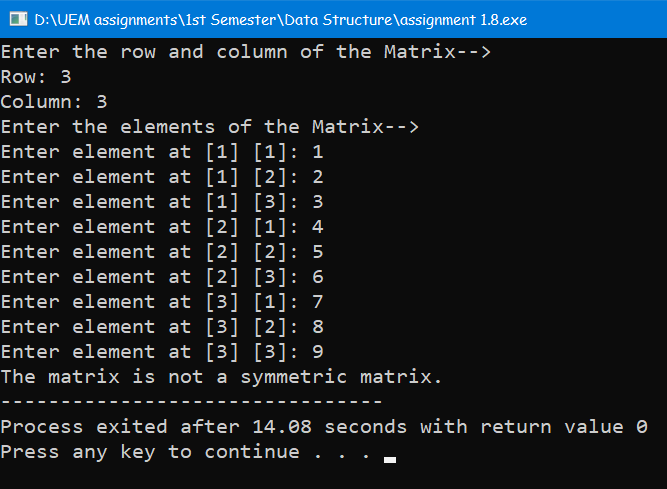
printf("The matrix is not a symmetric matrix.");

return 0;

}

* **Output:**

****

****

* **Problem No: 9**
* **Problem Statement:** Write a C program to print reverse array.
* **Source Code:**

#include<stdio.h>

int main() {

int len, i;

printf("Enter the length of the array \n");

scanf("%d", &len);

int arr[len];

printf("Enter the elements \n");

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

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

printf ("The array is->\n");

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

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

printf ("\nThe reverse of the array is->\n");

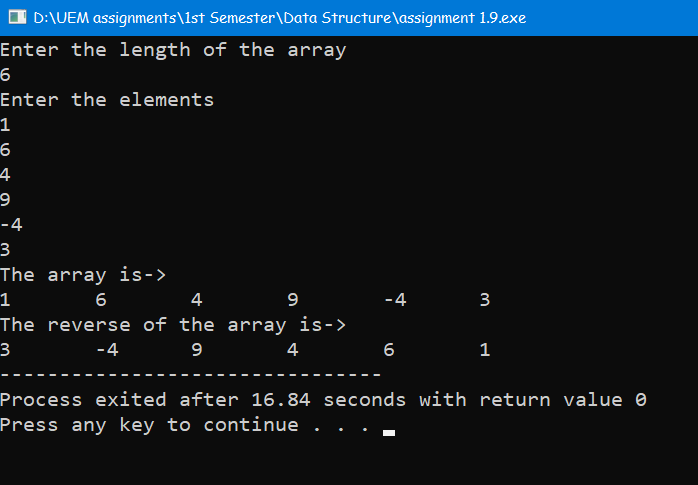
for (i = len - 1; i >= 0; i--)

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

return 0;

}

* **Output:**

****

* **Problem No: 10**
* **Problem Statement:** Write a C program to check the sum of all elements of an array.
* **Source Code:**

#include<stdio.h>

int main() {

int len, i, sum = 0;

printf("Enter the lenght of the array: ");

scanf("%d", &len);

int arr[len];

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

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

printf("%d element: ", i + 1);

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

}

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

sum += arr[i];

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

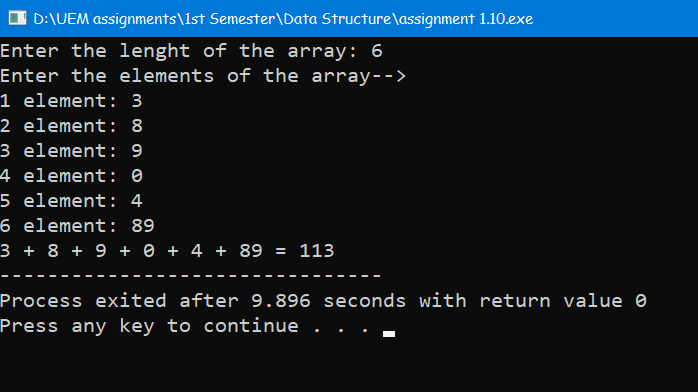
}

printf("\b\b= %d", sum);

return 0;

}

* **Output:**

****

* **Problem No: 11**
* **Problem Statement:** Write a C program to check duplicate number in an array.
* **Source Code:**

#include <stdio.h>

int main() {

int len, i, j, count = 0;

printf("Enter the lenght of the array: ");

scanf("%d", &len);

int arr[len];

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

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

printf("%d element: ", i + 1);

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

}

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

for (j = i + 1; j < len; j++)

if (arr[i] == arr[j]) {

count++;

break;

}

if (count == 0)

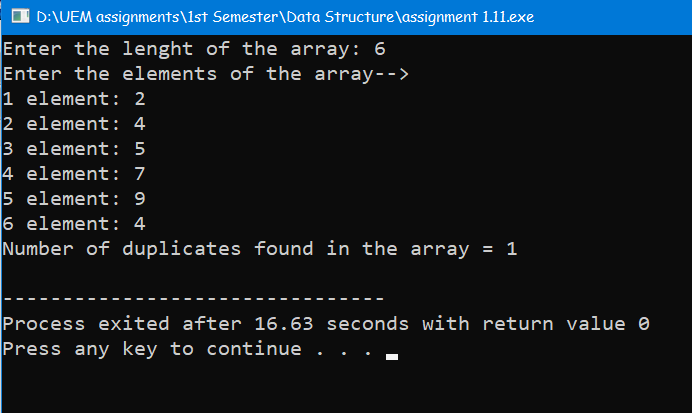
printf("No duplicates found in the array.\n");

else

printf("Number of duplicates found in the array = %d\n", count);

return 0;

}

* **Output:**

Data Structure lab assignment 2

* **Problem No: 1**
* **Problem Statement:** Write a C program to read a 2D array (with most of the elements as 0s) and then represent the same array as Sparse Metrics.
* **Source Code:**

#include <stdio.h>

#define MAX\_ROWS 50

#define MAX\_COLS 50

#define MAX\_ELEMENTS 1000

struct Element {

int row; int col; int value; };

void convertToSparse(int matrix[MAX\_ROWS][MAX\_COLS], int rows, int cols) {

struct Element sparse[MAX\_ELEMENTS];

int sparseIndex = 0;

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

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

if (matrix[i][j] != 0) {

sparse[sparseIndex].row = i;

sparse[sparseIndex].col = j;

sparse[sparseIndex].value = matrix[i][j];

sparseIndex++;

} } }

printf("Sparse Matrix Representation:\n");

printf("Row Col Value\n");

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

printf("%3d %3d %4d\n", sparse[i].row, sparse[i].col, sparse[i].value);

}}

int main() {

int rows, cols; int matrix[MAX\_ROWS][MAX\_COLS];

printf("Enter the number of rows and columns for the 2D array: ");

scanf("%d %d", &rows, &cols);

printf("Enter the elements of the 2D array:\n");

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

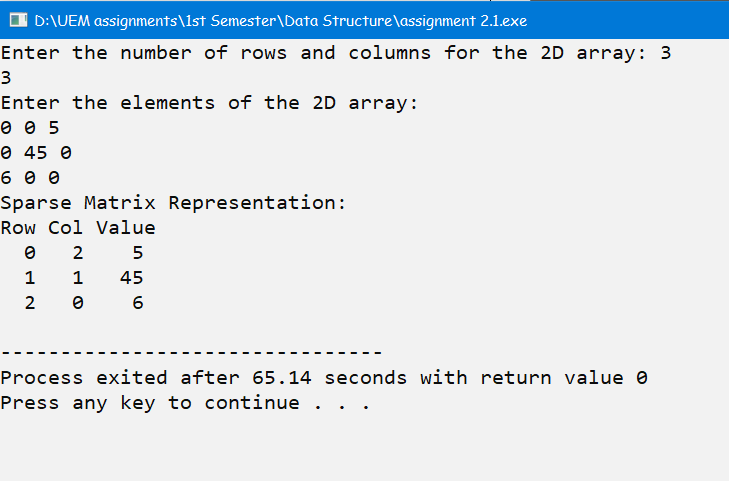
for (int j = 0; j < cols; j++) {

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

convertToSparse(matrix, rows, cols);

return 0;

}

* **Output:**
* **Problem No: 2 & 3**
* **Problem Statement**: Write a C program to pass an array to a function using Call by Value and Call by reference, update the array values in the function, print the array elements both in the function and in the calling function**.**
* **Source Code:**

#include <stdio.h>

void modifyByValue(int arr[], int size) {

int i;

printf("Array elements in the function (Call by Value):\n");

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

arr[i] += 10;

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

printf("\n");}

void modifyByReference(int \*arr, int size) {

int i;

printf("Array elements in the function (Call by Reference):\n");

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

\*arr += 5;

arr++;

printf("%d ", \*(arr - 1)); } printf("\n");}

void displayArray(int arr[], int size, const char \*message) {

int i;

printf("%s\n", message);

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

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

printf("\n"); }

int main() {

int n, i;

printf("Enter the number of elements of array: ");

scanf("%d", &n);

int arr[n] ;

printf("Enter the elements of the 2D array:\n");

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

scanf("%d", &arr[i]); }

int size = sizeof(arr) / sizeof(arr[0]);

printf("%d",size);

displayArray(arr, size, "Array elements in the main function:");

modifyByValue(arr, size);

displayArray(arr, size, "Array elements after Call by Value:");

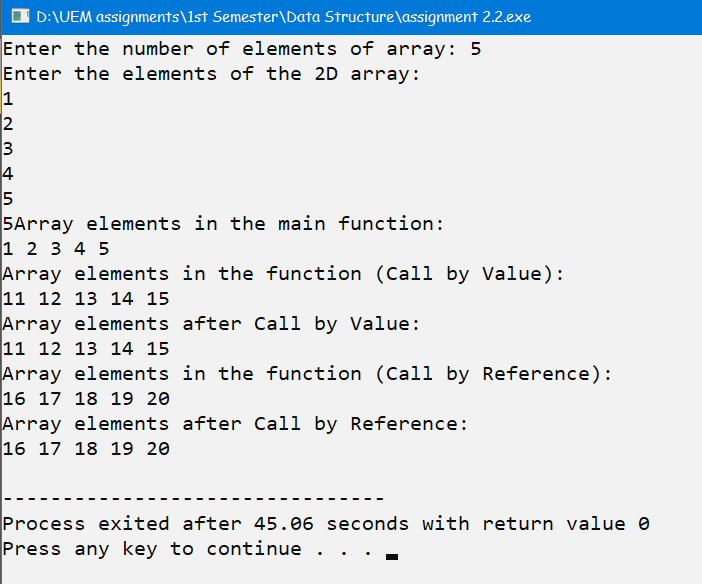
modifyByReference(arr, size);

displayArray(arr, size, "Array elements after Call by Reference:");

return 0;

}

* **Output:**

****

* **Problem No: 4**
* **Problem Statement:** Write a program to display n number of elements. Memory should be allocated dynamically using malloc( ).
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i;

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

scanf("%d", &n);

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

if (arr == NULL) {

printf("Memory allocation failed. Exiting...");

return 1;

}

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

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

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

}

printf("Elements you entered:\n");

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

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

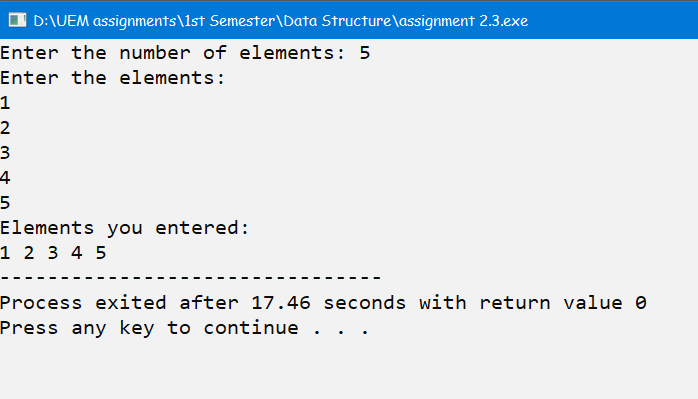
}

free(arr);

return 0;

}

* **Output:**



* **Problem No: 5**
* **Problem Statement:** Write a program to display n number of elements. Memory should be allocated dynamically using calloc( ).
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i;

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

scanf("%d", &n);

int \*arr = (int \*)calloc(n, sizeof(int));

if (arr == NULL) { printf("Memory allocation failed. Exiting...");

return 1; }

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

for (i = 0; i < n; i++) { scanf("%d", &arr[i]); }

printf("Elements you entered:\n");

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

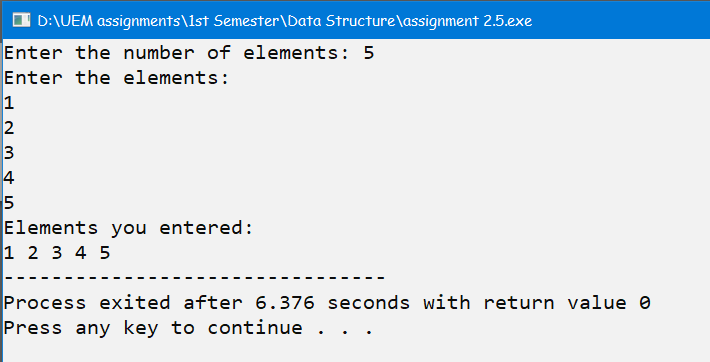
printf("%d ", arr[i]); }

free(arr);

return 0;

}

* **Output:**

****

* **Problem No: 6**
* **Problem Statement:** Write a program to allocate memory using malloc( ) and then reallocate the previously allocated memory using realloc( ). Display the elements which have been taken after reallocation.
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i;

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

scanf("%d", &n);

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

if (arr == NULL) { printf("Memory allocation failed. Exiting...");

return 1; }

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

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

scanf("%d", &arr[i]); }

printf("Elements before reallocation:\n");

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

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

int newSize;

printf("\nEnter the new size for reallocation: ");

scanf("%d", &newSize);

int \*newArr = (int \*)realloc(arr, newSize \* sizeof(int));

if (newArr == NULL) {

printf("Memory reallocation failed. Exiting...");

free(arr);

return 1;

}

printf("\nEnter additional elements:\n");

for (i = n; i < newSize; i++) {

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

}

printf("Elements after reallocation:\n");

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

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

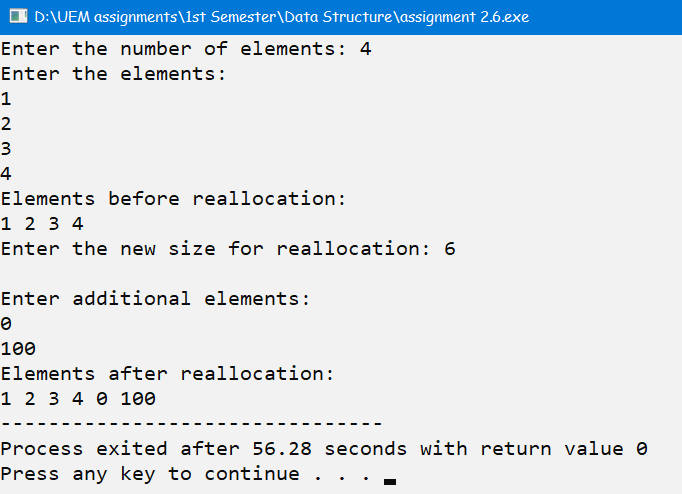
}

free(newArr);

return 0;

}

* **Output:**

****

* **Problem No: 7**
* **Problem Statement:** Write a program to allocate memory using calloc( ) and then reallocate the previously allocated memory using realloc( ). Display the elements which have been taken after reallocation.
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i;

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

scanf("%d", &n);

int \*arr = (int \*)calloc(n, sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed. Exiting...");

return 1;

}

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

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

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

}

printf("Elements before reallocation:\n");

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

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

}

int newSize;

printf("\nEnter the new size for reallocation: ");

scanf("%d", &newSize);

int \*newArr = (int \*)realloc(arr, newSize \* sizeof(int));

if (newArr == NULL) {

printf("Memory reallocation failed. Exiting...");

free(arr);

return 1;

}

printf("\nEnter additional elements:\n");

for (i = n; i < newSize; i++) {

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

}

printf("Elements after reallocation:\n");

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

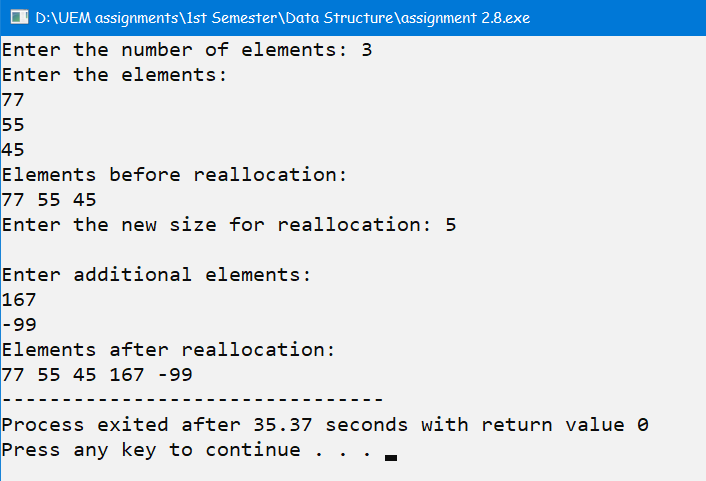
printf("%d ", newArr[i]);

}

free(newArr);

return 0;

}

* **Output:**
* **Problem No: 8**
* **Problem Statement:** Write a C program to search an element in an Array using dynamic memory allocation.
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

int searchElement(int \*arr, int size, int key) {

int i;

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

if (arr[i] == key) {

return i;

}

}

return -1;

}

int main() {

int n, key, i;

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

scanf("%d", &n);

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

if (arr == NULL) {

printf("Memory allocation failed. Exiting...");

return 1;

}

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

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

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

}

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

scanf("%d", &key);

int index = searchElement(arr, n, key);

if (index != -1) {

printf("%d found at index %d in the array.\n", key, index);

} else {

printf("%d not found in the array.\n", key);

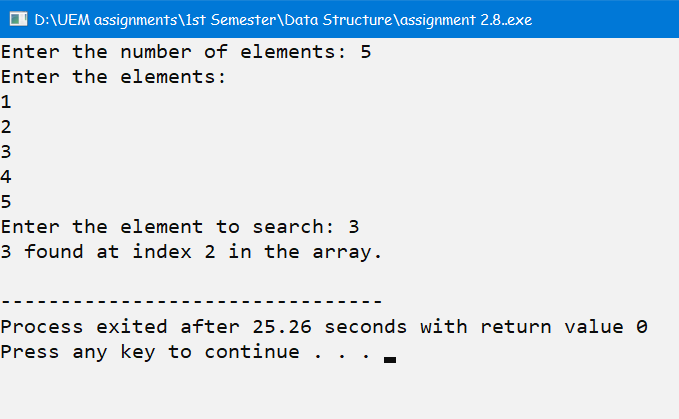
}

free(arr);

return 0;

}

* **Output:**

****

Data Structure lab assignment 3

* **Problem No: 1**
* **Problem Statement:**

Write a Menu driven C program to accomplish the following functionalities in single

linked list.

a) Create a single linked list.

b) Display the elements of a single linked list.

c) Insert a node at the beginning of a single linked list.

d) Insert a node at the end of a single linked list.

e) Insert a node before a given node of a single linked list.

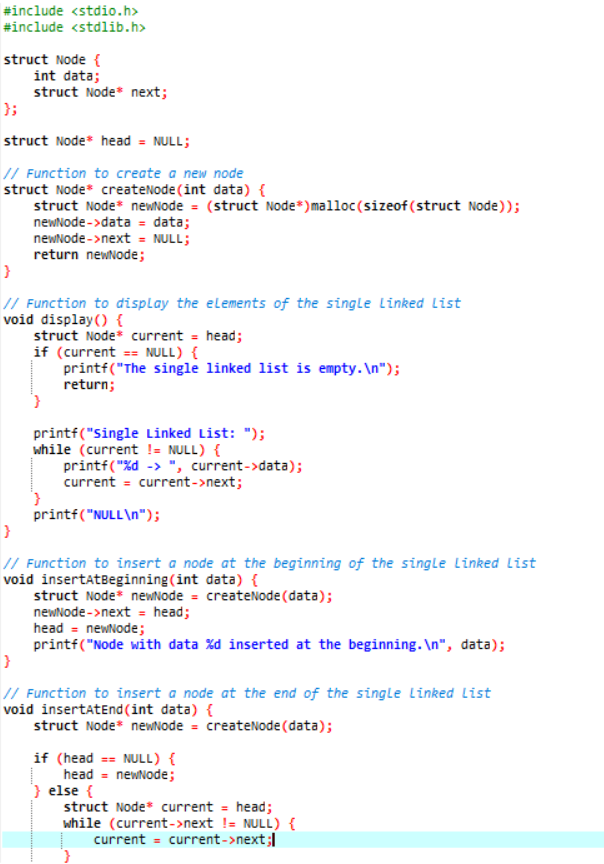
f) Insert a node after a given node of a single linked list.

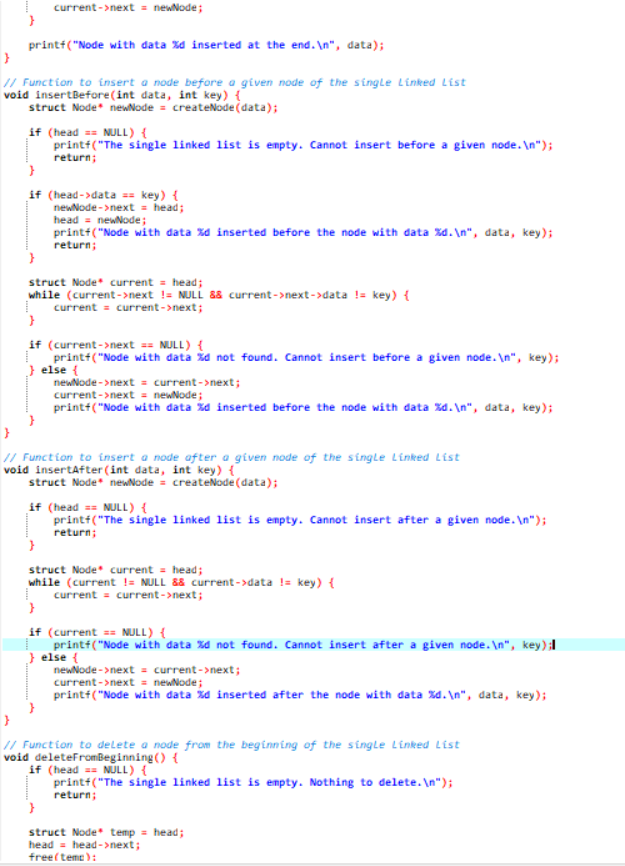
g) Delete a node from the beginning of a single linked list.

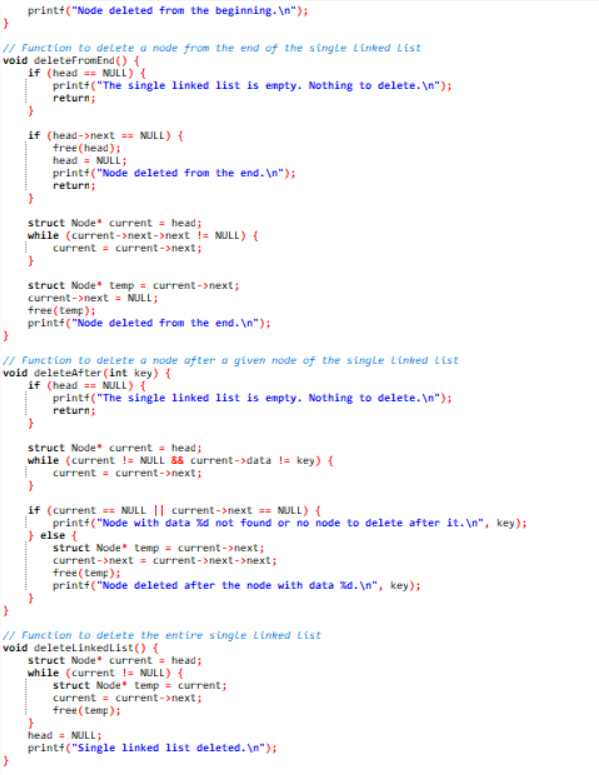
h) Delete a node from the end of a single linked list.

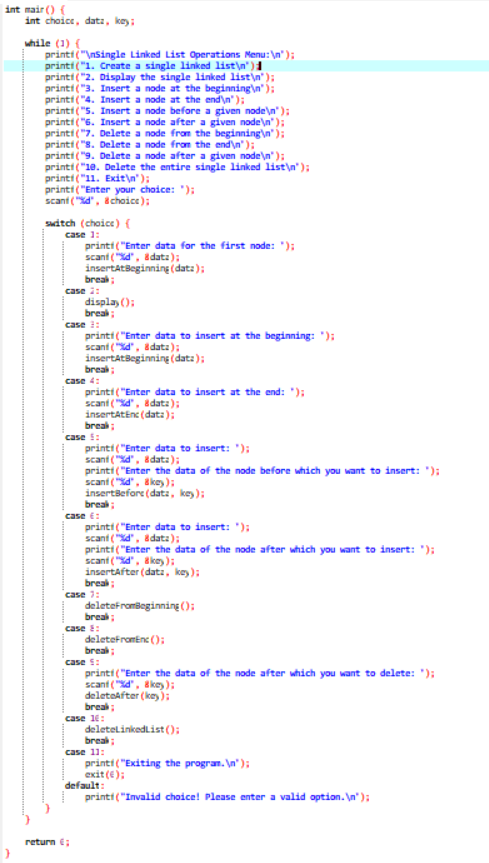
i) Delete a node after a given node of a single linked list.

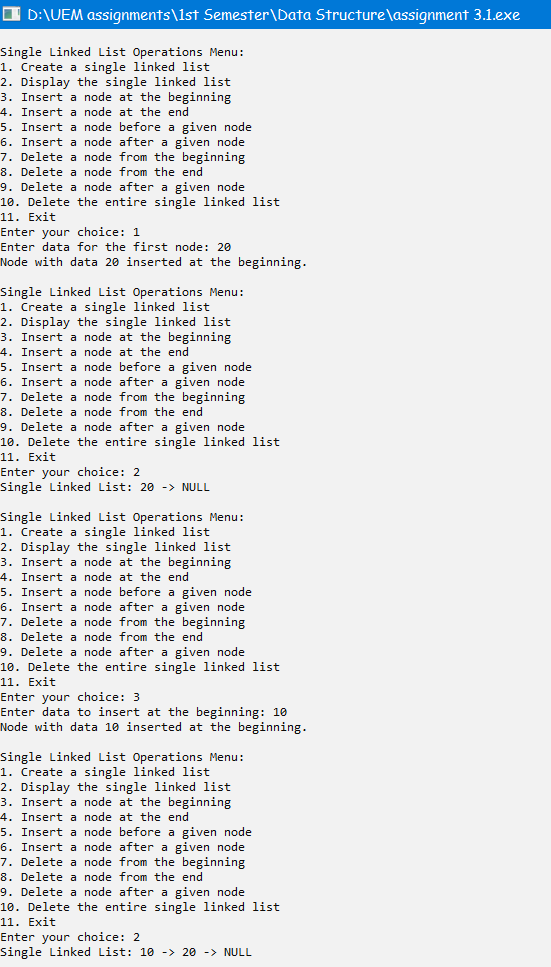
j) Delete the entire single linked list.

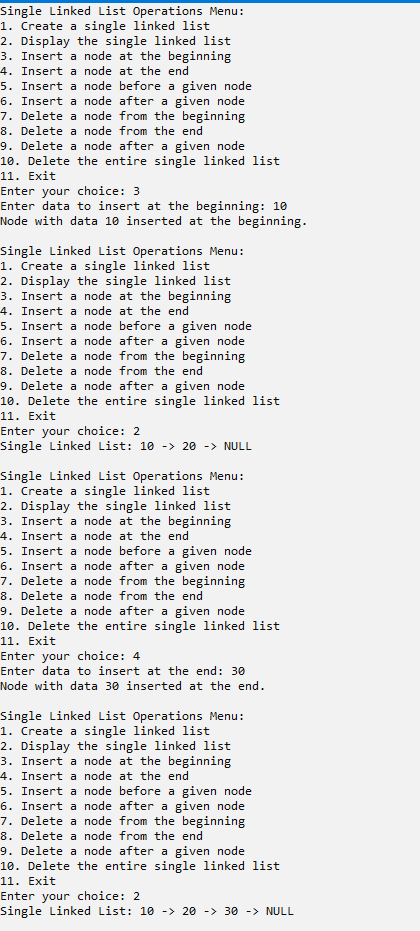
* **Source Code:**

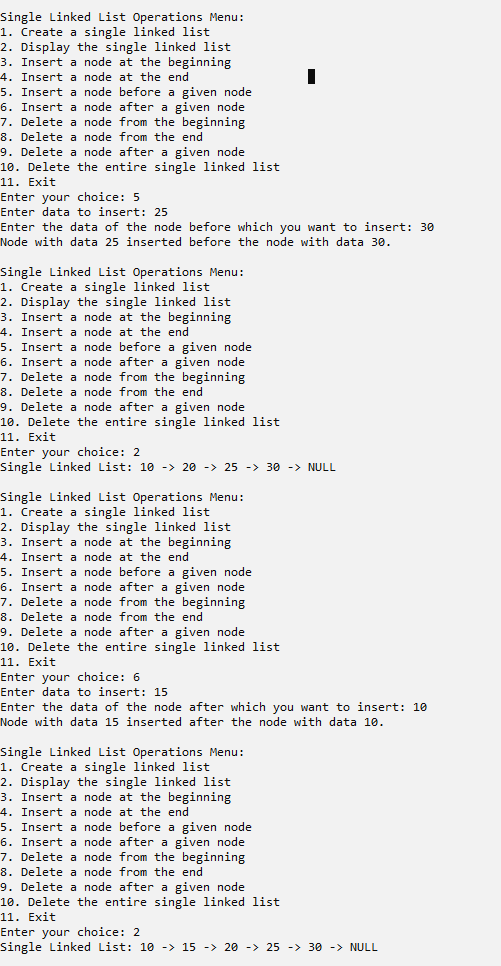


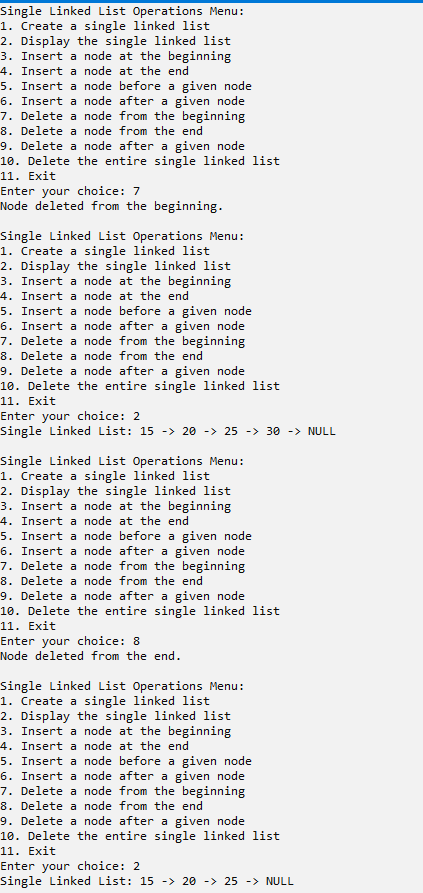


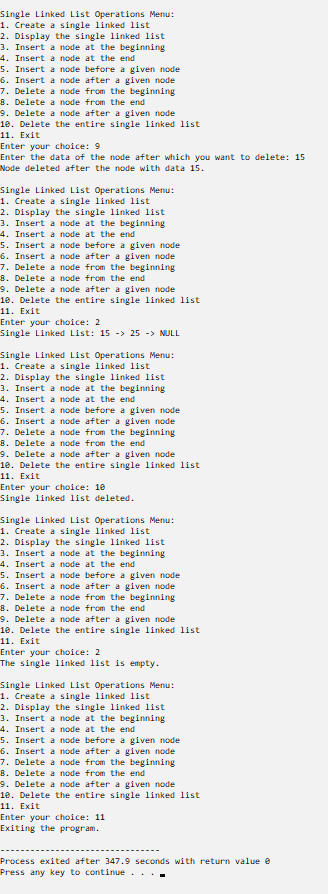


* **Output:**









* **Problem No: 2**
* **Problem Statement:** Write a Menu driven C program to accomplish the following functionalities in circular linked list.

a) Create a circular linked list.

b) Display the elements of a circular linked list.

c) Insert a node at the beginning of a circular linked list.

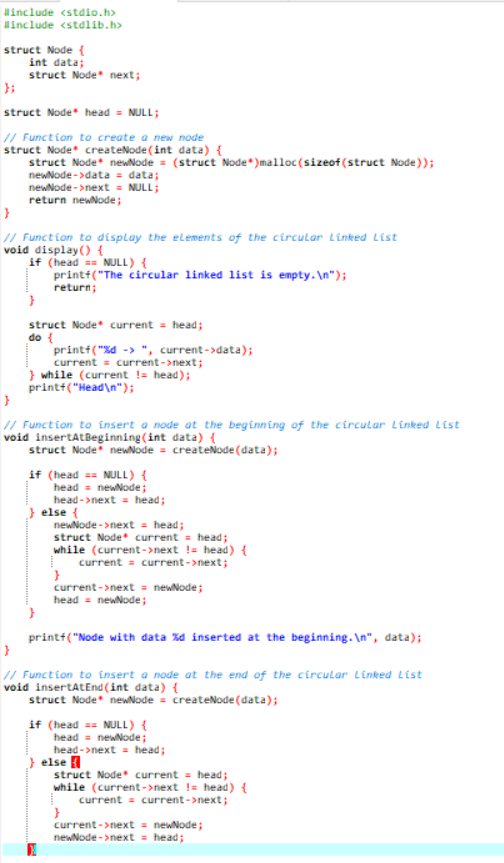
d) Insert a node at the end of a circular linked list.

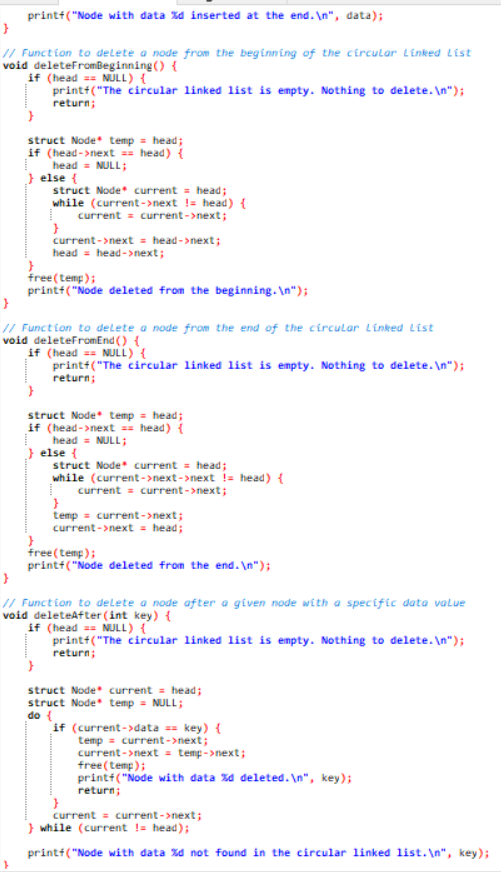
e) Delete a node from the beginning of a circular linked list.

f) Delete a node from the end of a circular linked list.

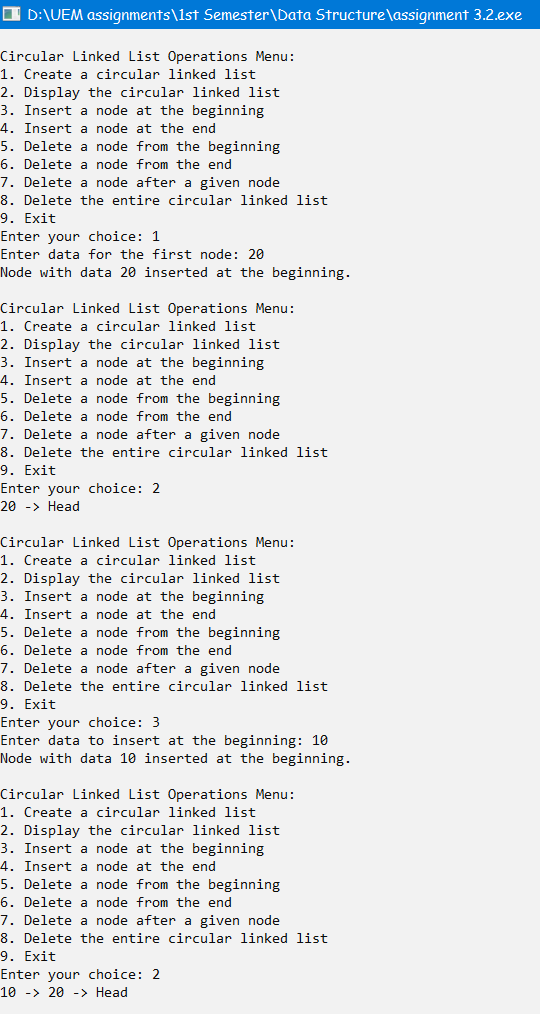
g) Delete a node after a given node of a circular linked list.

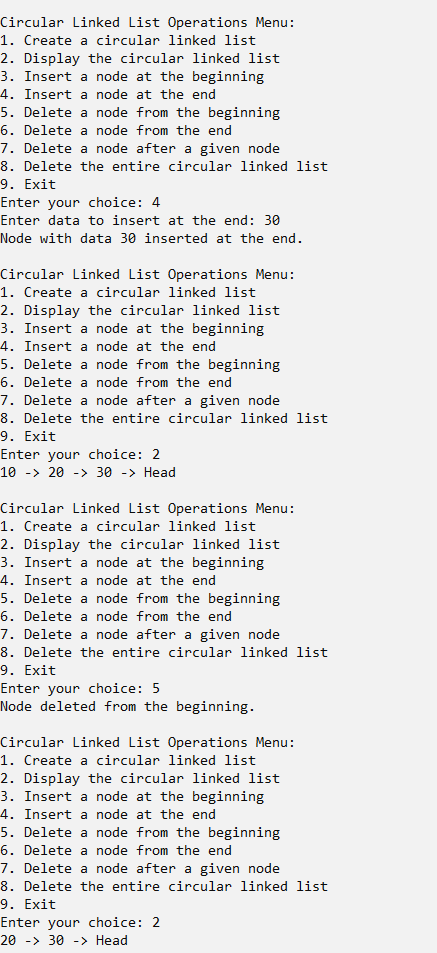
h) Delete the entire circular linked list.

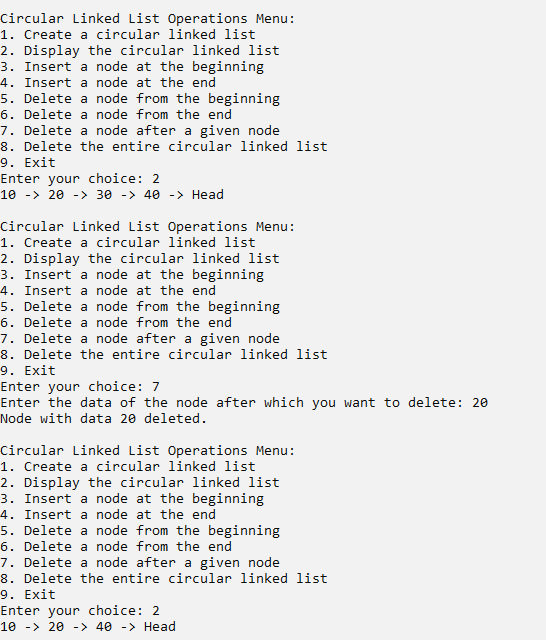
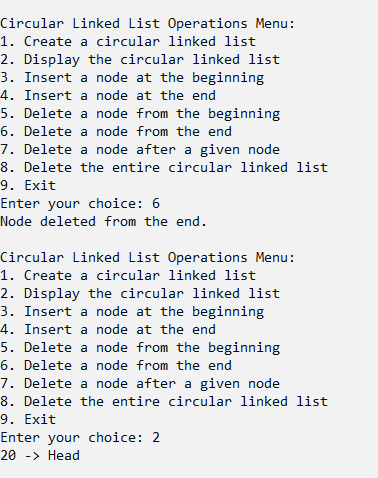
* **Source Code:**

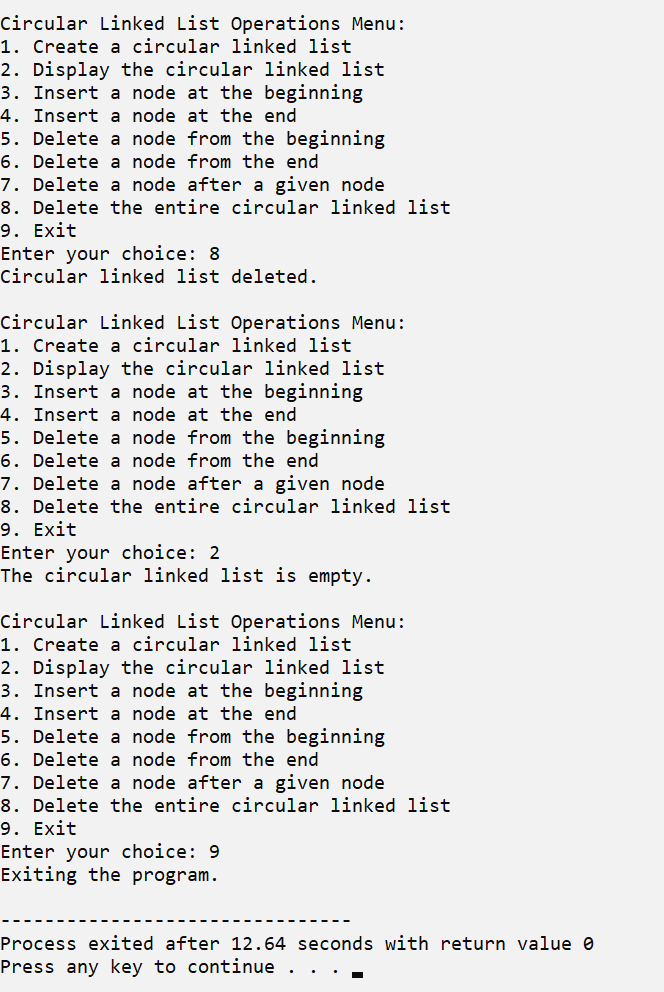
****

****

*****    **Output:**

****

****

****

Data Structure lab assignment 4

* **Problem No: 1**
* **Problem Statement: Write a Menu driven C program to accomplish the following functionalities in doubly linked list.**

a) Create a doubly linked list.

b) Display the elements of a doubly linked list.

c) Insert a node at the beginning of a doubly linked list.

d) Insert a node at the end of a doubly linked list.

e) Insert a node before a given node of a doubly linked list.

f) Insert a node after a given node of a doubly linked list.

g) Delete a node from the beginning of a doubly linked list.

h) Delete a node from the end of a doubly linked list.

i) Delete a node after a given node of a doubly linked list.

j) Delete the entire doubly linked list.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

// Node structure for a doubly linked list

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* head = NULL;

// Function to create a new node

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Function to display the doubly linked list

void displayList() {

struct Node \* current = head;

printf("Doubly Linked List:\nNULL <-> ");

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

// Function to insert a node at the beginning

void insertAtBeginning(int data) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

return;

}

newNode->next = head;

head->prev = newNode;

}

// Function to insert a node at the end

void insertAtEnd(int data) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

return;

}

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

// Function to insert a node before a given node

void insertBeforeNode(int data, int key) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

return;

}

if (head->data == key) {

newNode->next = head;

head->prev = newNode;

return;

}

struct Node\* temp = head;

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

temp = temp->next;

}

if (temp->next == NULL) {

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

free(newNode);

return;

}

newNode->next = temp->next;

newNode->prev = temp;

temp->next->prev = newNode;

temp->next = newNode;

}

// Function to insert a node after a given node

void insertAfterNode(int data, int key) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

return;

}

struct Node\* temp = head;

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

temp = temp->next;

}

if (temp == NULL) {

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

free(newNode);

return;

}

newNode->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = newNode;

}

newNode->prev = temp;

temp->next = newNode;

}

// Function to delete a node from the beginning

void deleteFromBeginning() {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return;

}

if (head -> next == NULL) {

free(head);

head = NULL;

}

struct Node\* temp = head;

head = head -> next;

head -> prev = NULL;

free(temp);

}

// Function to delete a node from the end

void deleteFromEnd() {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return;

}

if (head->next == NULL) {

free(head);

return;

}

struct Node\* temp = head;

while (temp->next->next != NULL) {

temp = temp->next;

}

free(temp->next);

temp->next = NULL;

}

// Function to delete a node after a given node

void deleteAfterNode(int key) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return;

}

struct Node\* temp = head;

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

temp = temp->next;

}

if (temp == NULL || temp->next == NULL) {

printf("Key not found or no node after the key to delete.\n");

return;

}

struct Node\* nodeToDelete = temp->next;

temp->next = nodeToDelete->next;

if (nodeToDelete->next != NULL) {

nodeToDelete->next->prev = temp;

}

free(nodeToDelete);

}

// Function to delete the entire list

void deleteList() {

while (head != NULL) {

struct Node\* temp = head;

head = head->next;

free(temp);

}

printf("Entire list deleted.\n");

}

int main() {

int choice, data, key;

do {

printf("\nDoubly Linked List Menu:\n");

printf("1. Create a doubly linked list\n");

printf("2. Display the list\n");

printf("3. Insert at the beginning\n");

printf("4. Insert at the end\n");

printf("5. Insert before a given node\n");

printf("6. Insert after a given node\n");

printf("7. Delete from the beginning\n");

printf("8. Delete from the end\n");

printf("9. Delete after a given node\n");

printf("10. Delete the entire list\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

// Create a doubly linked list

printf("Enter data for the first node: ");

scanf("%d", &data);

head = createNode(data);

break;

case 2:

// Display the elements of the doubly linked list

displayList();

break;

case 3:

// Insert at the beginning

printf("Enter data for the new node: ");

scanf("%d", &data);

insertAtBeginning(data);

break;

case 4:

// Insert at the end

printf("Enter data for the new node: ");

scanf("%d", &data);

insertAtEnd(data);

break;

case 5:

// Insert before a given node

printf("Enter data for the new node: ");

scanf("%d", &data);

printf("Enter the key value before which to insert: ");

scanf("%d", &key);

insertBeforeNode(data, key);

break;

case 6:

// Insert after a given node

printf("Enter data for the new node: ");

scanf("%d", &data);

printf("Enter the key value after which to insert: ");

scanf("%d", &key);

insertAfterNode(data, key);

break;

case 7:

// Delete from the beginning

deleteFromBeginning();

break;

case 8:

// Delete from the end

deleteFromEnd();

break;

case 9:

// Delete after a given node

printf("Enter the key value after which to delete: ");

scanf("%d", &key);

deleteAfterNode(key);

break;

case 10:

// Delete the entire list

deleteList();

break;

case 0:

// Exit the program

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

break;

default:

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

}

} while (choice != 0);

return 0;

}

* **Output:**

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 1

Enter data for the first node: 10

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 10 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 3

Enter data for the new node: 45

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 45 <-> 10 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 4

Enter data for the new node: 50

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 45 <-> 10 <-> 50 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 5

Enter data for the new node: 34

Enter the key value before which to insert: 10

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 45 <-> 34 <-> 10 <-> 50 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 6

Enter data for the new node: 22

Enter the key value after which to insert: 50

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 45 <-> 34 <-> 10 <-> 50 <-> 22 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 7

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 34 <-> 10 <-> 50 <-> 22 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 8

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 34 <-> 10 <-> 50 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 9

Enter the key value after which to delete: 10

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: 34 <-> 10 <-> NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 10

Entire list deleted.

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 2

Doubly Linked List: NULL

Doubly Linked List Menu:

1. Create a doubly linked list

2. Display the list

3. Insert at the beginning

4. Insert at the end

5. Insert before a given node

6. Insert after a given node

7. Delete from the beginning

8. Delete from the end

9. Delete after a given node

10. Delete the entire list

0. Exit

Enter your choice: 0

Exiting the program.

* **Problem No: 2**
* **Problem Statement: Write a Menu driven C program to accomplish the following functionalities in circular doubly linked list.**

a) Create a circular doubly linked list.

b) Display the elements of a circular doubly linked list.

c) Insert a node at the beginning of a circular doubly linked list.

d) Insert a node at the end of a circular doubly linked list.

e) Delete a node from the beginning of a circular doubly linked list.

f) Delete a node from the end of a circular doubly linked list.

g) Delete a node after a given node of a circular doubly linked list.

h) Delete the entire circular doubly linked list.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*prev;

struct Node \*next;

};

struct Node \*head = NULL;

void createList() {

int n, data;

struct Node \*newNode, \*temp;

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

scanf("%d", &n);

if (n <= 0) {

printf("Invalid number of nodes.\n");

return;

}

printf("Enter data for node 1: ");

scanf("%d", &data);

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

head->data = data;

head->next = head;

head->prev = head;

temp = head;

for (int i = 2; i <= n; i++) {

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

printf("Enter data for node %d: ", i);

scanf("%d", &data);

newNode->data = data;

newNode->next = head;

newNode->prev = temp;

temp->next = newNode;

head->prev = newNode;

temp = newNode;

}

printf("Circular doubly linked list created successfully.\n");

}

void displayList() {

struct Node \*temp;

if (head == NULL) {

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

return;

}

temp = head;

printf("Circular doubly linked list elements: ");

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

void insertAtBeginning() {

int data;

struct Node \*newNode, \*last;

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

scanf("%d", &data);

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

newNode->data = data;

if (head == NULL) {

head = newNode;

head->next = head;

head->prev = head;

} else {

last = head->prev;

newNode->next = head;

newNode->prev = last;

head->prev = newNode;

last->next = newNode;

head = newNode;

}

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

}

void insertAtEnd() {

int data;

struct Node \*newNode, \*last;

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

scanf("%d", &data);

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

newNode->data = data;

if (head == NULL) {

head = newNode;

head->next = head;

head->prev = head;

} else {

last = head->prev;

newNode->next = head;

newNode->prev = last;

head->prev = newNode;

last->next = newNode;

}

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

}

void deleteFromBeginning() {

struct Node \*temp;

if (head == NULL) {

printf("List is empty, nothing to delete.\n");

return;

} else if (head->next == head) {

free(head);

head = NULL;

} else {

temp = head;

head = head->next;

head->prev = temp->prev;

temp->prev->next = head;

free(temp);

}

printf("Node deleted from the beginning successfully.\n");

}

void deleteFromEnd() {

struct Node \*temp;

if (head == NULL) {

printf("List is empty, nothing to delete.\n");

return;

} else if (head->next == head) {

free(head);

head = NULL;

} else {

temp = head->prev;

head->prev = temp->prev;

temp->prev->next = head;

free(temp);

}

printf("Node deleted from the end successfully.\n");

}

void deleteAfterNode(int key) {

struct Node \*temp, \*toDelete;

temp = head;

while (temp->data != key) {

temp = temp->next;

if (temp == head) {

printf("Node with key %d not found.\n", key);

return;

}

}

toDelete = temp->next;

temp->next = toDelete->next;

toDelete->next->prev = temp;

free(toDelete);

printf("Node after key %d deleted successfully.\n", key);

}

void deleteList() {

struct Node \*current, \*temp;

if (head == NULL) {

printf("List is already empty.\n");

return;

}

current = head;

while (current->next != head) {

temp = current->next;

free(current);

current = temp;

}

free(head);

head = NULL;

printf("Circular doubly linked list deleted successfully.\n");

}

int main() {

int choice, key;

do {

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

printf("1. Create a circular doubly linked list\n");

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

printf("3. Insert a node at the beginning\n");

printf("4. Insert a node at the end\n");

printf("5. Delete a node from the beginning\n");

printf("6. Delete a node from the end\n");

printf("7. Delete a node after a given node\n");

printf("8. Delete the entire list\n");

printf("9. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

createList();

break;

case 2:

displayList();

break;

case 3:

insertAtBeginning();

break;

case 4:

insertAtEnd();

break;

case 5:

deleteFromBeginning();

break;

case 6:

deleteFromEnd();

break;

case 7:

printf("Enter the key after which the node should be deleted: ");

scanf("%d", &key);

deleteAfterNode(key);

break;

case 8:

deleteList();

break;

case 9:

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

break;

default:

printf("Invalid choice, please enter a valid option.\n");

}

} while (choice != 9);

return 0;

}

* **Output:**

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 1

Enter the number of nodes: 3

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 30

Circular doubly linked list created successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 10 20 30

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 3

Enter data to insert at the beginning: 50

Node inserted at the beginning successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 50 10 20 30

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 4

Enter data to insert at the end: 60

Node inserted at the end successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 50 10 20 30 60

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 5

Node deleted from the beginning successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 10 20 30 60

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 6

Node deleted from the end successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 10 20 30

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 7

Enter the key after which the node should be deleted: 20

Node after key 20 deleted successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

Circular doubly linked list elements: 10 20

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 8

Circular doubly linked list deleted successfully.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 2

List is empty.

Circular Doubly Linked List Operations:

1. Create a circular doubly linked list

2. Display the elements of the list

3. Insert a node at the beginning

4. Insert a node at the end

5. Delete a node from the beginning

6. Delete a node from the end

7. Delete a node after a given node

8. Delete the entire list

9. Exit

Enter your choice: 9

Exiting the program.

Data Structure lab assignment 5

* **Problem No: 1**
* **Problem Statement: Write a Menu driven C program to accomplish the following functionalities in Queue using an Array:**

a. Insert an element into the queue using an array (Enqueue Operation).

b. Delete an element from the queue using an array (Dequeue Operation).

c. Return the value of the FRONT element of the queue (without deleting it from the

queue) using an array (Peep operation).

d. Display the elements of a queue using an array.

* **Source Code:**

#include <stdio.h>

#define MAX\_SIZE 10

void enqueue(int element);

void dequeue();

void peep();

void display();

int queue[MAX\_SIZE];

int front = -1, rear = -1;

int main() {

int choice, element;

do {

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

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Peep\n");

printf("4. Display\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to enqueue: ");

scanf("%d", &element);

enqueue(element);

break;

case 2:

dequeue();

break;

case 3:

peep();

break;

case 4:

display();

break;

case 5:

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

break;

default:

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

} while (choice != 5);

return 0;

}

void enqueue(int element) {

if (rear == MAX\_SIZE - 1) {

printf("Queue is full. Cannot enqueue element.\n");

} else {

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = element;

printf("Enqueued %d\n", element);

}}

void dequeue() {

if (front == -1) {

printf("Queue is empty. Cannot dequeue element.\n");

} else {

printf("Dequeued %d\n", queue[front]);

if (front == rear) {

front = rear = -1;

} else {

front++;

}}}

void peep() {

if (front == -1) {

printf("Queue is empty. Peep operation not possible.\n");

} else {

printf("Front element: %d\n", queue[front]);

}}

void display() {

if (front == -1) {

printf("Queue is empty. Nothing to display.\n");

} else {

printf("Queue elements: ");

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

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

}

printf("\n");

}}

* **Output:**

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 1

Enter element to enqueue: 10

Enqueued 10

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 1

Enter element to enqueue: 20

Enqueued 20

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 4

Queue elements: 10 20

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 3

Front element: 10

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 2

Dequeued 10

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 4

Queue elements: 20

Queue Menu:

1. Enqueue

2. Dequeue

3. Peep

4. Display

5. Exit

Enter your choice: 5

Exiting program. Goodbye!

* **Problem No: 2**
* **Problem Statement: Write a Menu driven C program to accomplish the following functionalities in Queue using Linked List:**

e. Insert an element into the queue using a Linked List (Enqueue Operation).

f. Delete an element from the queue using a Linked List (Dequeue Operation).

g. Return the value of the FRONT element of the queue (without deleting it from the

queue) using a Linked List (Peep operation).

h. Display the elements of a queue using a Linked List.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

struct Queue {

struct Node \*front, \*rear;

};

struct Queue \*createQueue() {

struct Queue \*queue = (struct Queue \*)malloc(sizeof(struct Queue));

queue->front = queue->rear = NULL;

return queue;

}

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

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

newNode->data = data;

newNode->next = NULL;

if (queue->rear == NULL) {

queue->front = queue->rear = newNode;

return;

}

queue->rear->next = newNode;

queue->rear = newNode;

}

void dequeue(struct Queue \*queue) {

if (queue->front == NULL) {

printf("Queue is empty. Cannot dequeue.\n");

return;

}

struct Node \*temp = queue->front;

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

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

}

int peek(struct Queue \*queue) {

if (queue->front == NULL) {

printf("Queue is empty. Peek operation cannot be performed.\n");

return -1;

}

return queue->front->data;

}

void displayQueue(struct Queue \*queue) {

struct Node \*temp = queue->front;

if (temp == NULL) {

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

return;

}

printf("Queue elements: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct Queue \*queue = createQueue();

int choice, element;

do {

printf("\nQueue Operations using Linked List:\n");

printf("1. Insert element into the queue (Enqueue)\n");

printf("2. Delete element from the queue (Dequeue)\n");

printf("3. Return value of the FRONT element (Peek)\n");

printf("4. Display elements of the queue\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &element);

enqueue(queue, element);

printf("Element enqueued successfully.\n");

break;

case 2:

dequeue(queue);

printf("Element dequeued successfully.\n");

break;

case 3:

element = peek(queue);

if (element != -1) {

printf("Front element of the queue: %d\n", element);

}

break;

case 4:

displayQueue(queue);

break;

case 5:

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

break;

default:

printf("Invalid choice, please enter a valid option.\n");

}

} while (choice != 5);

return 0;

}

* **Output:**

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 1

Enter the element to enqueue: 34

Element enqueued successfully.

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 1

Enter the element to enqueue: 45

Element enqueued successfully.

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 1

Enter the element to enqueue: 23

Element enqueued successfully.

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 2

Element dequeued successfully.

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 3

Front element of the queue: 45

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 4

Queue elements: 45 23

Queue Operations using Linked List:

1. Insert element into the queue (Enqueue)

2. Delete element from the queue (Dequeue)

3. Return value of the FRONT element (Peek)

4. Display elements of the queue

5. Exit

Enter your choice: 5

Exiting the program

* **Problem No: 3**
* **Problem Statement: Write a Menu driven C program to accomplish the following functionalities in Circular Queue using Array:**

i. Insert an element into the circular queue.

j. Delete an element from the circular queue.

k. Return the value of the FRONT element of the circular queue (without deleting it

from the queue).

l. Display the elements of a circular queue using the circular queue.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 10

struct CircularQueue {

int items[MAX\_SIZE];

int front, rear;

};

struct CircularQueue \*createQueue() {

struct CircularQueue \*queue = (struct CircularQueue \*)malloc(sizeof(struct CircularQueue));

queue->front = -1;

queue->rear = -1;

return queue;

}

int isFull(struct CircularQueue \*queue) {

if ((queue->front == 0 && queue->rear == MAX\_SIZE - 1) || (queue->front == queue->rear + 1)) {

return 1;

}

return 0;

}

int isEmpty(struct CircularQueue \*queue) {

if (queue->front == -1) {

return 1;

}

return 0;

}

void enqueue(struct CircularQueue \*queue, int value) {

if (isFull(queue)) {

printf("Queue is full. Cannot enqueue.\n");

return;

}

if (queue->front == -1) {

queue->front = 0;

}

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

queue->items[queue->rear] = value;

printf("Element enqueued successfully.\n");

}

void dequeue(struct CircularQueue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty. Cannot dequeue.\n");

return;

}

printf("Dequeued element: %d\n", queue->items[queue->front]);

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

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

} else {

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

}}

int peek(struct CircularQueue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty. Peek operation cannot be performed.\n");

return -1;

}

return queue->items[queue->front];

}

void displayQueue(struct CircularQueue \*queue) {

int i;

if (isEmpty(queue)) {

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

return;

}

printf("Circular Queue elements: ");

i = queue->front;

do {

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

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

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

printf("\n");

}

int main() {

struct CircularQueue \*queue = createQueue();

int choice, element;

do {

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

printf("i. Insert element into the queue (Enqueue)\n");

printf("j. Delete element from the queue (Dequeue)\n");

printf("k. Return value of the FRONT element (Peek)\n");

printf("l. Display elements of the queue\n");

printf("m. Exit\n");

printf("Enter your choice: ");

scanf(" %c", &choice);

switch (choice) {

case 'i':

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

scanf("%d", &element);

enqueue(queue, element);

break;

case 'j':

dequeue(queue);

break;

case 'k':

element = peek(queue);

if (element != -1) {

printf("Front element of the queue: %d\n", element);

}

break;

case 'l':

displayQueue(queue);

break;

case 'm':

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

break;

default:

printf("Invalid choice, please enter a valid option.\n");

}

} while (choice != 'm');

return 0;

}

* **Output:**

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: i

Enter the element to enqueue: 22

Element enqueued successfully.

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: i

Enter the element to enqueue: 44

Element enqueued successfully.

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: i

Enter the element to enqueue: 66

Element enqueued successfully.

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: j

Dequeued element: 22

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: k

Front element of the queue: 44

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: l

Circular Queue elements: 44 66

Circular Queue Operations using Array:

i. Insert element into the queue (Enqueue)

j. Delete element from the queue (Dequeue)

k. Return value of the FRONT element (Peek)

l. Display elements of the queue

m. Exit

Enter your choice: m

Exiting the program.

Data Structure lab assignment 6

* **Problem No: 1**
* **Problem Statement:** Write a Menu driven C program to accomplish the following functionalities in Stack using an Array:

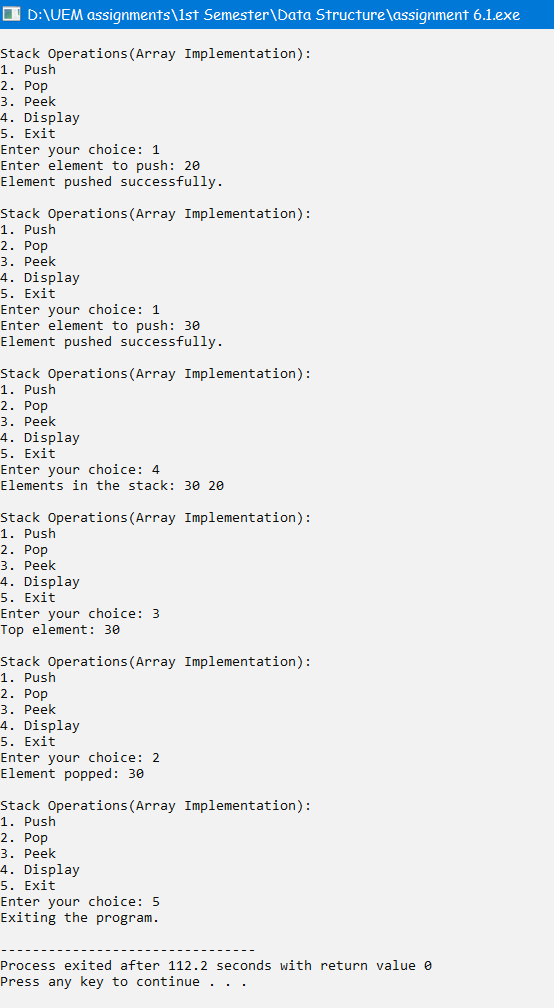
a. Insert an element into the stack using an array (Push Operation).

b. Delete an element from the stack using an array (Pop Operation).

c. Return the value of the topmost element of the stack (without deleting it from the stack) using an array.

d. Display the elements of a stack using an array.

* **Source Code:**
* **Output:**

****

*    **Problem No:  2**

* **Problem Statement:** Write a Menu driven C program to accomplish the following functionalities in Stack using Linked List:

a. Insert an element into the stack using a Linked List (Push Operation).

b. Delete an element from the stack using a Linked List (Pop Operation).

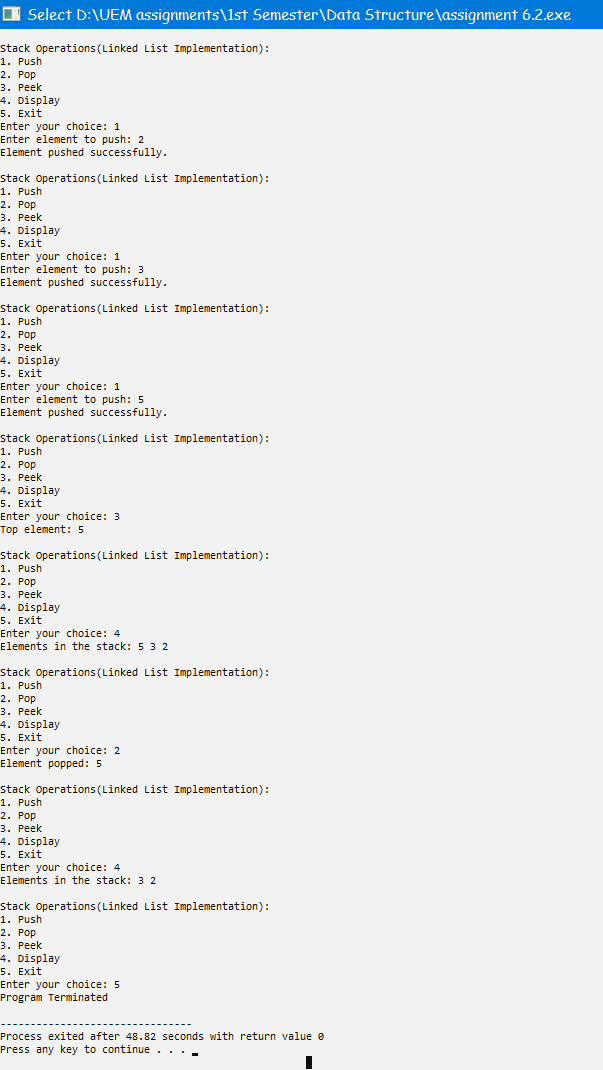
c. Return the value of the topmost element of the stack (without deleting it from the stack) using a Linked List.

d. Display the elements of the stack using a Linked List.

* **Source Code:**



*    **Output:**

****

*    **Problem No:  3**

* **Problem Statement:**

Write a program to convert an infix expression into its equivalent postfix notation.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

char stack[MAX\_SIZE];

int top = -1;

int precedence(char ch) {

if (ch == '+' || ch == '-')

return 1;

else if (ch == '\*' || ch == '/')

return 2;

else

return 0;

}

void push(char ch) {

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

printf("Stack Overflow\n");

exit(1);

}

else {

top++;

stack[top] = ch;

}

}

char pop() {

if (top == -1) {

printf("Stack Underflow\n");

exit(1);

} else

return stack[top--];

}

void infixToPostfix(char\* infix) {

char postfix[MAX\_SIZE];

int i = 0, j = 0;

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

char token = infix[i];

if (token >= 'a' && token <= 'z' || token >= 'A' && token <= 'Z')

postfix[j++] = token;

else if (token == '(')

push(token);

else if (token == ')') {

while (stack[top] != '(')

postfix[j++] = pop();

top--;

} else {

while (top != -1 && precedence(stack[top]) >= precedence(token))

postfix[j++] = pop();

push(token);

}

i++;

}

while (top != -1)

postfix[j++] = pop();

postfix[j] = '\0';

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

}

int main() {

char infix[MAX\_SIZE];

printf("Enter an Infix Expression: ");

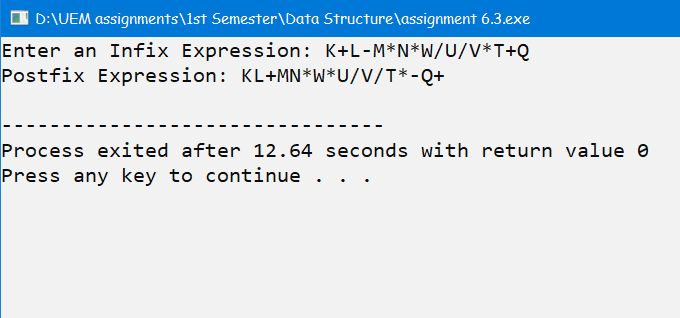
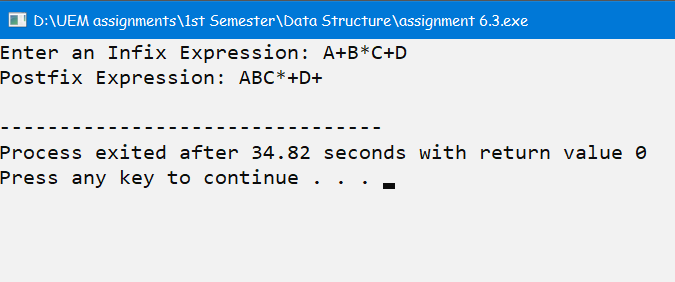
scanf("%s", infix);

infixToPostfix(infix);

return 0;

}

*    **Output:**

****

*    **Problem No:  4**

*    **Problem Statement:** Write a program to convert an infix expression into its equivalent prefix notation.

* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

char stack[MAX\_SIZE];

int top = -1;

int precedence(char ch) {

if (ch == '+' || ch == '-')

return 1;

else if (ch == '\*' || ch == '/')

return 2;

else

return 0;

}

void push(char ch) {

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

printf("Stack Overflow\n");

exit(1);

} else {

top++;

stack[top] = ch;

}

}

char pop() {

if (top == -1) {

printf("Stack Underflow\n");

exit(1);

} else

return stack[top--];

}

void infixToPrefix(char\* infix) {

int length = strlen(infix), i;

char prefix[MAX\_SIZE];

int j = 0;

for (i = length - 1; i >= 0; i--) {

char token = infix[i];

if (token >= 'a' && token <= 'z' || token >= 'A' && token <= 'Z')

prefix[j++] = token;

else if (token == ')')

push(token);

else if (token == '(') {

while (stack[top] != ')')

prefix[j++] = pop();

top--;

} else {

while (top != -1 && precedence(stack[top]) > precedence(token))

prefix[j++] = pop();

push(token);

}

}

while (top != -1)

prefix[j++] = pop();

for (i = 0; i < j / 2; i++) {

char temp = prefix[i];

prefix[i] = prefix[j - i - 1];

prefix[j - i - 1] = temp;

}

prefix[j] = '\0';

printf("Prefix Expression: %s\n", prefix);

}

int main() {

char infix[MAX\_SIZE];

printf("Enter an Infix Expression: ");

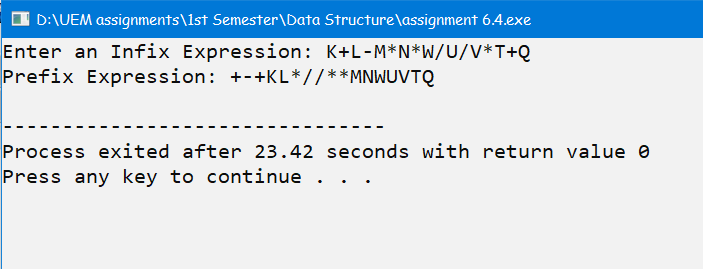
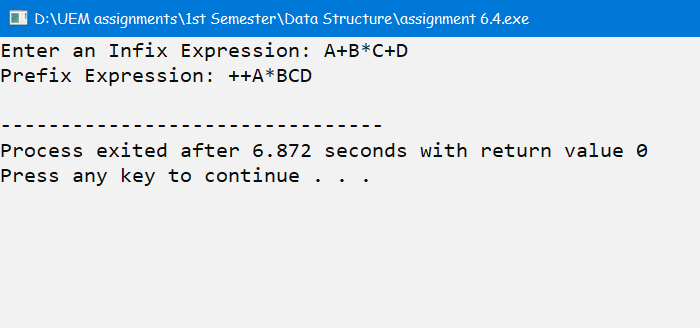
scanf("%s", infix);

infixToPrefix(infix);

return 0;

}

*    **Output:**

****

* **Problem No: 5**
* **Problem Statement:** Write a program to evaluate a postfix expression.
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

void push(int value) {

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

printf("Stack Overflow\n");

exit(1);

} else {

top++;

stack[top] = value;

}

}

int pop() {

if (top == -1) {

printf("Stack Underflow\n");

exit(1);

} else

return stack[top--];

}

int evaluatePostfix(char\* postfix) {

int i, a, b;

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

char token = postfix[i];

if (token >= '0' && token <= '9')

push(token - '0');

else {

a = pop();

b = pop();

switch (token) {

case '+':

push(b + a);

break;

case '-':

push(b - a);

break;

case '\*':

push(b \* a);

break;

case '/':

push(b / a);

break;

}

}

}

return stack[top];

}

int main() {

char postfix[MAX\_SIZE];

printf("Enter a Postfix Expression: ");

scanf("%s", postfix);

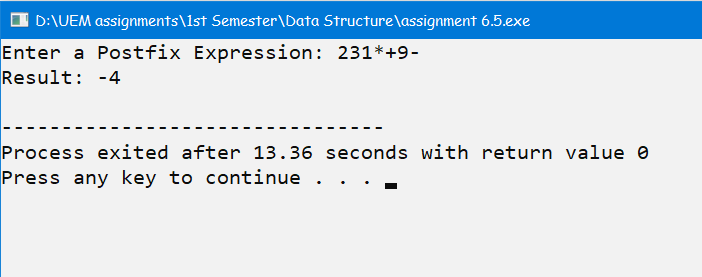
int result = evaluatePostfix(postfix);

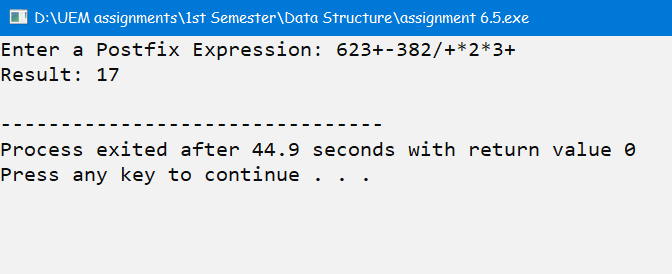
printf("Result: %d\n", result);

return 0;

}

* **Output:**

****

****

* **Problem No: 6**
* **Problem Statement:** Write a program to evaluate a prefix expression.
* **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

void push(int value) {

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

printf("Stack Overflow\n");

exit(1);

} else {

top++;

stack[top] = value;

}

}

int pop() {

if (top == -1) {

printf("Stack Underflow\n");

exit(1);

} else

return stack[top--];

}

int evaluatePrefix(char\* prefix) {

int length = strlen(prefix), i;

for (i = length - 1; i >= 0; i--) {

char token = prefix[i];

if (token >= '0' && token <= '9')

push(token - '0');

else {

int a = pop();

int b = pop();

switch (token) {

case '+':

push(a + b);

break;

case '-':

push(a - b);

break;

case '\*':

push(a \* b);

break;

case '/':

push(a / b);

break;

}

}

}

return stack[top];

}

int main() {

char prefix[MAX\_SIZE];

printf("Enter a Prefix Expression: ");

scanf("%s", prefix);

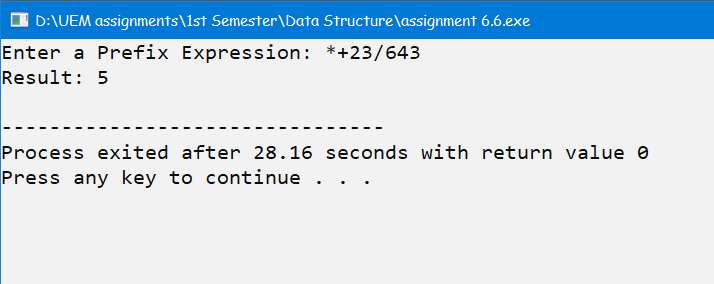
int result = evaluatePrefix(prefix);

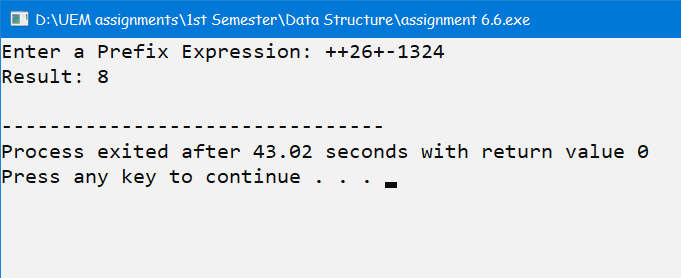
printf("Result: %d\n", result);

return 0;

}

* **Output:**

****

****

*    **Problem No: 7**

*    **Problem Statement:** Write a program to print the Fibonacci series using recursion.

**Source Code:**



#include <stdio.h>

int fibonacci(int n) {

if (n <= 1)

return n;

else

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

int n, i;

printf("Enter the number of terms in the Fibonacci series: ");

scanf("%d", &n);

printf("Fibonacci Series: ");

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

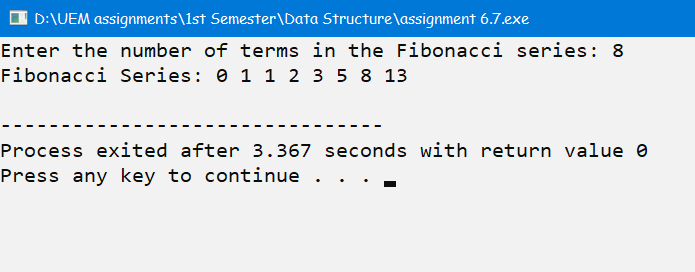
printf("%d ", fibonacci(i));

printf("\n");

return 0;

}

*    **Output:**

****

*    **Problem No: 8**

*    **Problem Statement:** Write a program to solve the tower of Hanoi problem using recursion.

*    **Source Code:**

 #include <stdio.h>

void towerOfHanoi(int n, char source, char aux, char dest) {

if (n == 1){

printf("Move disk 1 from disk %c to disk %c\n", source, dest);

return;

}

towerOfHanoi(n - 1, source, dest, aux);

printf("Move disk %d from disk %c to disk %c\n", n, source, dest);

towerOfHanoi(n - 1, aux, source, dest);

}

int main() {

int num\_disks;

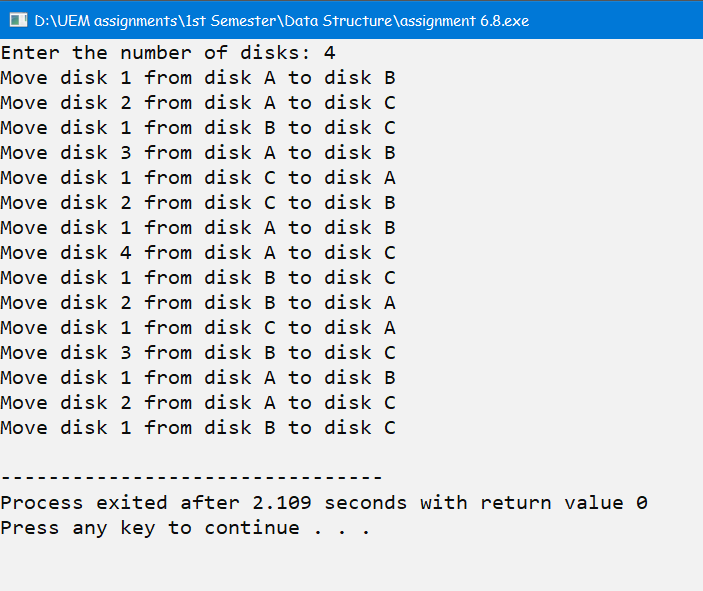
printf("Enter the number of disks: ");

scanf("%d", &num\_disks);

towerOfHanoi(num\_disks, 'A', 'B', 'C');

return 0;

}

*****    **Output:**