****

**: Data Structure Laboratory - 4.5CA251C01**

|  |
| --- |
| **Submitted by** |

|  |  |
| --- | --- |
| **Student name** | **Nidhi** |
| **Roll. No.** | **24/SCA/BCA/074** |
| **Programme** | **BCA** |
| **Semester** | **2nd Semester** |
| **Section** | **B** |
| **Session/Batch** | **2024-25** |

|  |
| --- |
| **Submitted to** |

|  |  |
| --- | --- |
| **Faculty Name** |  |

**Array**

**1. Program to demonstrate insertion and output in an array:**

**#include <stdio.h>**

**int main(){**

**int arr[5];**

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

**for(int i = 0; i<5; i++){**

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

**}**

**printf("\n The elements that you have enterd are: ");**

**for(int i = 0; i<5; i++){**

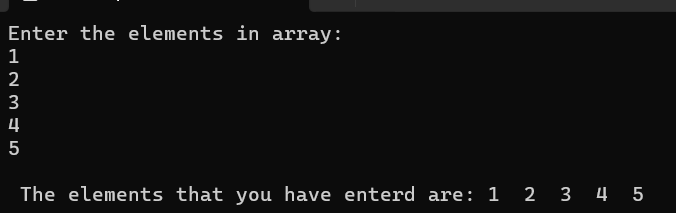
**printf("%d ", arr[i]);**

**}**

**return 0;**

**}**

**Output:**



**2. Program to demonstrate searching in an array:**

**#include <stdio.h>**

**int main() {**

**int arr[] = {10, 20, 30, 40, 50};**

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

**int key, found = 0;**

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

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

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

**if (arr[i] == key) {**

**printf("Element %d found at index %d.\n", key, i);**

**found = 1;**

**break;**

**}**

**}**

**if (found == 0) {**

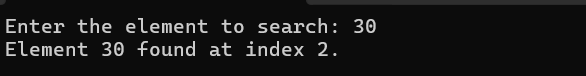
**printf("Element %d not found in the array.\n", key);**

**}**

**return 0;**

**}**

**Output:**



**3. Program to demonstrate sorting in an array:**

**#include <stdio.h>**

**int main(){**

**int arr[5] = {1,7,3,6,4};**

**int temp;**

**for(int i = 0; i<5; i++){**

**for(int j = 0; j<4; j++){**

**if(arr[j] > arr[j + 1]){**

**temp = arr[j];**

**arr[j] = arr[j + 1];**

**arr[j + 1] = temp;**

**}**

**}**

**}**

**printf("Ascending order: \n");**

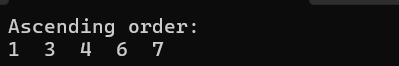
**for(int i = 0; i<5; i++){**

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

**}**

**}**

**Output:**



**Stacks**

**Stack is a linear data structure that follows the LIFO (Last –In-First-Out) method. It means that the last data element to be inserted in the stack will be the first to come out.**

**There are two operations on Stack. They are:**

**Push: It is used to insert an element into the stack.**

**Pop: It is used to remove an element from the stack.**

**4. Program to demonstrate PUSH() and operations in the stack:**

**#include<stdio.h>**

**#define Max 5**

**struct Stack{**

**int data[Max];**

**int top;**

**};**

**struct Stack s; Initializing the stack.**

**//Declaring Functions**

**void PUSH(int n);**

**void POP();**

**void PRINT();**

**Program execution starts from here.**

**int main(){**

**s.top = -1;**

**POP();**

**PUSH(10);**

**PUSH(20);**

**PUSH(30);**

**PUSH(40);**

**PUSH(50);**

**PUSH(60);**

**POP();**

**PRINT();**

**return 0;**

**}**

**4.1. Function to insert an element into the stack.**

**void PUSH(int n){**

**if(s.top == Max-1){**

**printf("Stack is full.\n\n");**

**return;**

**}**

**s.data[++s.top] = n;**

**printf("%d pushed into the stack\n\n", n);**

**}**

**// 4.2. Function to remove an element from the stack.**

**void POP(){**

**if(s.top == -1){**

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

**return;**

**}**

**int value = s.data[s.top--];**

**printf("%d removed from the stack\n\n", value);**

**}**

**4.3. Function to traverse and print all the elements in the stack.**

**void PRINT(){**

**if(s.top == -1){**

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

**return;**

**}**

**printf("Elements in the stack:\n");**

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

**printf("%d\t", s.data[i]);**

**if(i==s.top){**

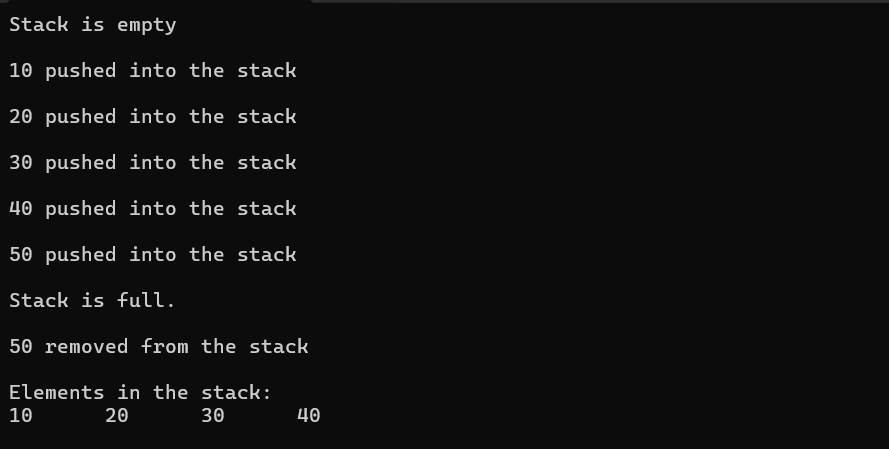
**printf("\n\n");**

**}**

**}**

**}**

**Output:**



**Queue**

**A queue is a linear data structure that stores data in FIFO (First-In-First-Out) order. It means that the first element to be inserted in the queue will also be the first element to come out.**

**The operations we can perform on a queue are:**

**Enqueue: To insert an element in the queue.**

**Dequeue: To remove an element from the queue.**

**5. Program to demonstrate ENQUEUE () and DEQUEUE () operations in a queue:**

**#include<stdio.h>**

**#define Max 5**

**//Structure of a queue.**

**struct Queue{**

**int data[Max];**

**int front;**

**int rear;**

**};**

**struct Queue q; //Initializing a queue.**

**//Declaring functions.**

**void ENQUEUE(int n);**

**void DEQUEUE();**

**void PRINT();**

**Program execution starts from here.**

**int main(){**

**q.front = -1;**

**q.rear = -1;**

**DEQUEUE();**

**ENQUEUE(10);**

**ENQUEUE(20);**

**ENQUEUE(30);**

**ENQUEUE(40);**

**ENQUEUE(50);**

**ENQUEUE(60);**

**DEQUEUE();**

**PRINT();**

**return 0;**

**}**

**5.1 Function to insert an element in the queue.**

**void ENQUEUE(int n){**

**if(q.rear == Max-1){**

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

**return;**

**}**

**q.data[++q.rear] = n;**

**printf("%d inserted in the queue\n\n", n);**

**}**

**5.2. Function to remove an element from a queue.**

**void DEQUEUE(){**

**if(q.rear == q.front){**

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

**q.front = -1;**

**q.rear = -1;**

**return;**

**}**

**int value = q.data[++q.front];**

**printf("%d removed from the queue\n\n", value);**

**}**

**// 5.3. Function to traverse and display all the elements in the queue.**

**void PRINT(){**

**if(q.front == q.rear){**

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

**return;**

**}**

**for(int i=q.front+1; i<=q.rear; i++){**

**printf("%d\t", q.data[i]);**

**if(i==q.rear){**

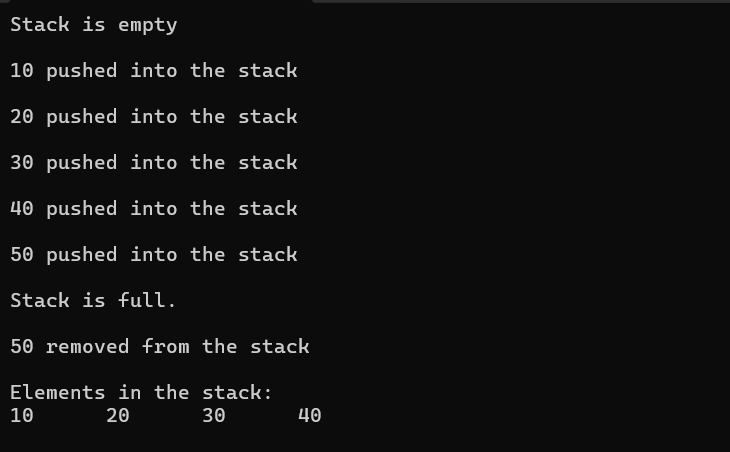
**printf("\n\n");**

**}**

**}**

**}**

**Output:**



**Circular Queue**

**A circular queue is a type of queue in which the last element of the queue is connected to the first element of the queue, making a circular chain.**

**6. Program to implement circular queue and operations on it.**

**#include <stdio.h>**

**#define MAX 5**

**struct Queue {**

**int data[MAX];**

**int front;**

**int rear;**

**};**

**struct Queue q = {.front = -1, .rear = -1};**

**6.1. Function to insert element into the circular queue**

**void ENQUEUE(int value) {**

**if ((q.rear + 1) % MAX == q.front) {**

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

**return;**

**}**

**if (q.front == -1) {**

**q.front = 0;**

**}**

**q.rear = (q.rear + 1) % MAX;**

**q.data[q.rear] = value;**

**printf("%d inserted into the queue\n", value);**

**}**

**6.2. Function to remove element from the circular queue**

**void DEQUEUE() {**

**if (q.front == -1) {**

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

**return;**

**}**

**int value = q.data[q.front];**

**if (q.front == q.rear) {**

**// Only one element in queue**

**q.front = -1;**

**q.rear = -1;**

**} else {**

**q.front = (q.front + 1) % MAX;**

**}**

**printf("%d removed from the queue\n", value);**

**}**

**6.3. Function to display the elements of the queue**

**void DISPLAY() {**

**if (q.front == -1) {**

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

**return;**

**}**

**printf("Queue elements: ");**

**int i = q.front;**

**while (1) {**

**printf("%d ", q.data[i]);**

**if (i == q.rear) break;**

**i = (i + 1) % MAX;**

**}**

**printf("\n");**

**}**

**Main function to test the queue**

**int main() {**

**ENQUEUE(10);**

**ENQUEUE(20);**

**ENQUEUE(30);**

**ENQUEUE(40);**

**ENQUEUE(50); // Will say "Queue is full" because one slot is always kept empty.**

**DISPLAY();**

**DEQUEUE();**

**DEQUEUE();**

**DISPLAY();**

**ENQUEUE(60);**

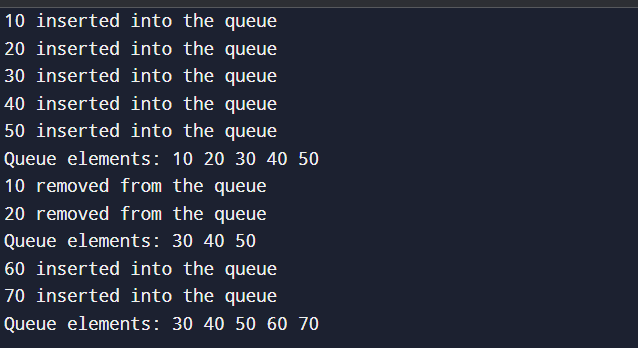
**ENQUEUE(70);**

**DISPLAY();**

**return 0;**

**}**

**Output:**



**Linked List**

**A linked list is a linear data structure that uses nodes to store the data, and each node is connected to another node via pointers.**

**We can perform various operations on a linked list such as:**

**Insertion at beginning.**

**Insertion at end.**

**Insertion at specific position.**

**Deletion from beginning.**

**Deletion form end.**

**Deletion from a specific position.**

**7. Program to show the implementation of a linked list and its operations.**

**#include<stdio.h>**

**#include<stdlib.h>**

**//Structure of a Node.**

**struct Node{**

**int data;**

**struct Node\* next;**

**};**

**//Declaring functions.**

**void insertAtBeginning(Node\*\* pointerToHead, int x);**

**void insertAtEnd(Node\*\* pointerToHead, int x);**

**void insertAt(Node\*\* pointerToHead, int pos, int x);**

**void deleteFromFirst(Node\*\* pointerToHead);**

**void deleteFromEnd(Node\*\* pointerToHead);**

**void deleteFrom(Node\*\* pointerToHead, int pos);**

**void Print(Node\* head);**

**Execution of program starts from here.**

**int main(){**

**struct Node\* head = NULL;**

**insertAtEnd(&head, 20);**

**insertAtBeginning(&head, 10);**

**insertAtBeginning(&head, 30);**

**insertAt(&head, 1, 70);**

**insertAt(&head, 4, 80);**

**insertAt(&head, 5, 90);**

**insertAtEnd(&head, 100);**

**deleteFrom(&head, 1);**

**deleteFrom(&head, 4);**

**deleteFrom(&head, 5);**

**deleteFromFirst(&head);**

**deleteFromEnd(&head);**

**Reverse(&head);**

**Print(head);**

**}**

**7.1. Insert a node at the beginning.**

**void insertAtBeginning(Node\*\* pointerToHead, int x){**

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

**newNode -> data = x;**

**newNode -> next = \*pointerToHead;**

**\*pointerToHead = newNode;**

**}**

**7.2. Insert a node at the end.**

**void insertAtEnd(Node\*\* pointerToHead, int x){**

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

**newNode -> data = x;**

**newNode -> next = NULL;**

**if(\*pointerToHead == NULL){**

**\*pointerToHead = newNode;**

**}**

**else{**

**struct Node\* endNode = \*pointerToHead;**

**while(endNode -> next != NULL){**

**endNode = endNode -> next;**

**}**

**endNode -> next = newNode;**

**}**

**}**

**7.3. Insert a node at nth position.**

**void insertAt(Node\*\* pointerToHead, int pos, int x){**

**if(pos<1){**

**printf("Invalid Position\n");**

**return;**

**}**

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

**newNode -> data = x;**

**if(pos == 1){**

**newNode -> next = \*pointerToHead;**

**\*pointerToHead = newNode;**

**return;**

**}**

**struct Node\* current = \*pointerToHead;**

**for(int i=1; i<pos-1 && current!=NULL; i++){**

**current = current -> next;**

**}**

**if(current == NULL){**

**printf("Invlalid position\n");**

**free(newNode);**

**return;**

**}**

**newNode -> next = current -> next;**

**current -> next = newNode;**

**}**

**7.4. Delete a node from nth position.**

**void deleteFrom(Node\*\* pointerToHead, int pos){**

**if(\*pointerToHead == NULL || pos<1){**

**printf("Invalid position or empty list\n");**

**return;**

**}**

**if(pos == 1){**

**Node\* temp = \*pointerToHead;**

**\*pointerToHead = (\*pointerToHead)-> next;**

**free(temp);**

**return;**

**}**

**Node\* current = \*pointerToHead;**

**for(int i=1; i<pos-1 && current != NULL; i++){**

**current = current -> next;**

**}**

**if(current == NULL || current -> next == NULL){**

**printf("Invalid position\n");**

**return;**

**}**

**Node\* temp = current -> next;**

**current -> next = current -> next -> next;**

**free(temp);**

**}**

**7.5. Delete a node from first position.**

**void deleteFromFirst(Node\*\* pointerToHead){**

**if(\*pointerToHead == NULL){**

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

**return;**

**}**

**Node\* temp = \*pointerToHead;**

**\*pointerToHead = (\*pointerToHead) -> next;**

**free(temp);**

**}**

**7.6. Delete a node from the end.**

**void deleteFromEnd(Node\*\* pointerToHead){**

**if(\*pointerToHead == NULL){**

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

**return;**

**}**

**if((\*pointerToHead)-> next == NULL){**

**\*pointerToHead = NULL;**

**return;**

**}**

**Node\* endNode = \*pointerToHead;**

**while(endNode -> next -> next != NULL){**

**endNode = endNode -> next;**

**}**

**Node\* temp = endNode -> next;**

**endNode -> next = NULL;**

**free(temp);**

**}**

**7.7. Print the list.**

**void Print(Node\* head){**

**while(head != NULL){**

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

**head = head -> next;**

**}**

**printf("\n");**

**}**

**Output:**

