**Shri Vaishnav Vidyapeeth Vishwavidyala, Indore**

**Shri Vaishnav Institute of Information Technology**

**Department of Computer Science & Engineering**



**LAB FILE**

**Jan-Jun2023**

**Name of the Student:**

**Enrollment:**

**Program:**

**Section:**

**Year/Sem: I / II**

**Subject Code: BTCS201N**

**Subject Name: Data Structure and Algorithms**

**Name of Subject Teacher: Dr. Sandeep Kumar Jain**

**INDEX**

**Name:**

**Enrollment No.**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Aim/Objective** | **Date of Experiment** | **Sign / Remarks** |
| 1 | WAP to find maximum and minimum number in an array. |  |  |
| 2 | Perform searching and sorting in an array. |  |  |
| 3 | WAP to find lowercase, uppercase, wordcount, total characters and numericdigits in a string. |  |  |
| 4 | Perform insertion and deletion operation in 1D array. |  |  |
| 5 | WAP to perform various operation in the single linked list.   1. CreateFirstNode 2. InsertAtBeginning 3. InsertAtLast 4. DeleteFromBeginning 5. DeleteFromLast 6. DisplayList |  |  |
| 6 | WAP to perform insertion and deletion operation in circular linked list. |  |  |
| 7 | Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of static memory allocation. |  |  |
| 8 | Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of dynamic memory allocation. |  |  |
| 9 | WAP for Tower of Hanoi using recursion. |  |  |
| 10 | Illustrate queue implementation using array with following operation as enQueue, deQueue, isEmpty, displayQueue. |  |  |
| 11 | Illustrate queue implementation using linked list with following operation as enQueue, deQueue, isEmpty, displayQueue. |  |  |
| 12 | WAP to construct a binary search tree and perform deletion and inorder traversal on it. |  |  |
| 13 | WAP to develop an algorithm for binary search, implement and test it. |  |  |
| 14 | WAP for implementation of Bubble Sort. |  |  |
| 15 | WAP for implementation of Insertion Sort. |  |  |

**Program No:- 5**

**Aim/Title:** WAP to perform various operation in the single linked list.

1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List

**Date:**

**Tool:**

**Algorithm/Flow Chart:-**

**Algorithm:**

**1. Start the program.**

**2. Create a struct Node to represent the linked list node.**

**3. Define a function createNode(data) to create a new node with the given data.**

**- Allocate memory for a new node.**

**- Set the data of the new node.**

**- Set the next pointer of the new node to NULL.**

**- Return the new node.**

**4. Define a function insertAtBeginning(head, data) to insert a new node at the beginning of the list.**

**- Create a new node using createNode(data).**

**- Set the next pointer of the new node to the current head.**

**- Set the head pointer to the new node.**

**- Print "Node inserted at the beginning."**

**5. Define a function insertAtLast(head, data) to insert a new node at the end of the list.**

**- Create a new node using createNode(data).**

**- If the head is NULL, set the head to the new node and print "Node inserted at the last."**

**- Otherwise, iterate through the list until the last node.**

**- Set the next pointer of the last node to the new node.**

**- Print "Node inserted at the last."**

**6. Define a function deleteFromBeginning(head) to delete the first node from the list.**

**- If the head is NULL, print "List is empty. Cannot delete from beginning."**

**- Otherwise, store the head in a temporary variable.**

**- Set the head to the next node.**

**- Free the memory of the temporary variable.**

**- Print "Node deleted from the beginning."**

**7. Define a function deleteFromLast(head) to delete the last node from the list.**

**- If the head is NULL, print "List is empty. Cannot delete from last."**

**- If the head is the only node, free the head and set it to NULL.**

**- Otherwise, iterate through the list until the second-to-last node.**

**- Set the next pointer of the second-to-last node to NULL.**

**- Free the last node.**

**- Print "Node deleted from the last."**

**8. Define a function displayList(head) to display the elements of the list.**

**- If the head is NULL, print "List is empty."**

**- Otherwise, iterate through the list and print each node's data.**

**9. Start the main function.**

**10. Declare the head pointer and set it to NULL.**

**11. Declare variables choice and data.**

**12. Start a do-while loop.**

**- Print the menu options.**

**- Read the choice from the user.**

**- Use a switch statement based on the choice.**

**- Case 1: Call insertAtBeginning(&head, data).**

**- Case 2: Call insertAtLast(&head, data).**

**- Case 3: Call deleteFromBeginning(&head).**

**- Case 4: Call deleteFromLast(&head).**

**- Case 5: Call displayList(head).**

**- Case 0: Print "Exiting..." and exit the loop.**

**- Default: Print "Invalid choice!".**

**- Continue the loop until the choice is not 0.**

**13. End the main function.**

**14. End the program.**

**Flow char:-**



|  |
| --- |
| **Start the program** |
| **Line arrow: Straight with solid fill** |
| **Create first node** |
| **Line arrow: Straight with solid fill** |
| **Insert at beginning** |
| **Line arrow: Straight with solid fill** |
| **Insert at last** |
| **Line arrow: Straight with solid fill** |
| **Delete from beginning** |
| **Line arrow: Straight with solid fill** |
| **Delete from last** |
| **Line arrow: Straight with solid fill** |
| **Display the list** |
| **Line arrow: Straight with solid fill** |
| **Exit** |

**Source Code:-**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

**struct Node\* createNode(int data) {**

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

**if (newNode == NULL) {**

**printf("Memory allocation failed!");**

**exit(1);**

**}**

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

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

**struct Node\* newNode = createNode(data);**

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

**\*head = newNode;**

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

**}**

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

**struct Node\* newNode = createNode(data);**

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

**\*head = newNode;**

**printf("Node inserted at the last.\n");**

**return;**

**}**

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

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

**temp = temp->next;**

**}**

**temp->next = newNode;**

**printf("Node inserted at the last.\n");**

**}**

**void deleteFromBeginning(struct Node\*\* head) {**

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

**printf("List is empty. Cannot delete from beginning.\n");**

**return;**

**}**

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

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

**free(temp);**

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

**}**

**void deleteFromLast(struct Node\*\* head) {**

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

**printf("List is empty. Cannot delete from last.\n");**

**return;**

**}**

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

**free(\*head);**

**\*head = NULL;**

**printf("Node deleted from the last.\n");**

**return;**

**}**

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

**struct Node\* prev = NULL;**

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

**prev = temp;**

**temp = temp->next;**

**}**

**prev->next = NULL;**

**free(temp);**

**printf("Node deleted from the last.\n");**

**}**

**void displayList(struct Node\* head) {**

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

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

**return;**

**}**

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

**printf("List: ");**

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

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

**temp = temp->next;**

**}**

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

**}**

**int main() {**

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

**int choice, data;**

**do {**

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

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

**printf("2. Insert at last\n");**

**printf("3. Delete from beginning\n");**

**printf("4. Delete from last\n");**

**printf("5. Display list\n");**

**printf("0. Exit\n");**

**printf("Enter your choice: ");**

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

**switch (choice) {**

**case 1:**

**printf("Enter data to insert: ");**

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

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

**break;**

**case 2:**

**printf("Enter data to insert: ");**

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

**insertAtLast(&head, data);**

**break;**

**case 3:**

**deleteFromBeginning(&head);**

**break;**

**case 4:**

**deleteFromLast(&head);**

**break;**

**case 5:**

**displayList(head);**

**break;**

**case 0:**

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

**break;**

**default:**

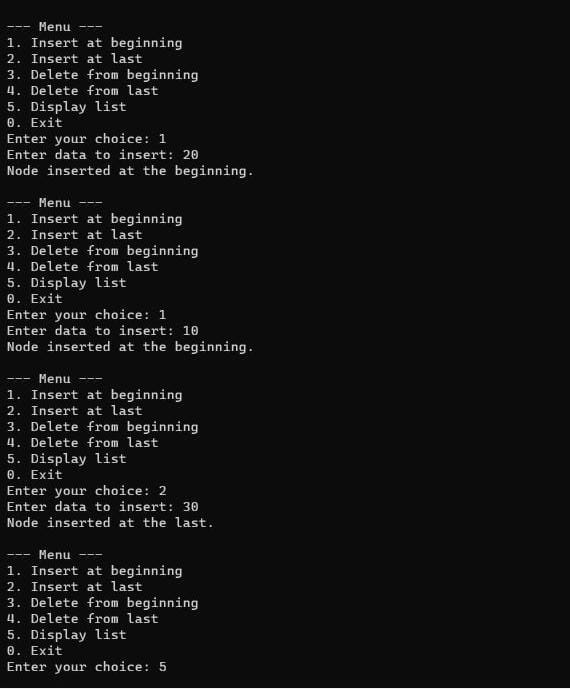
**printf("Invalid choice! Please enter a valid option.\n");**

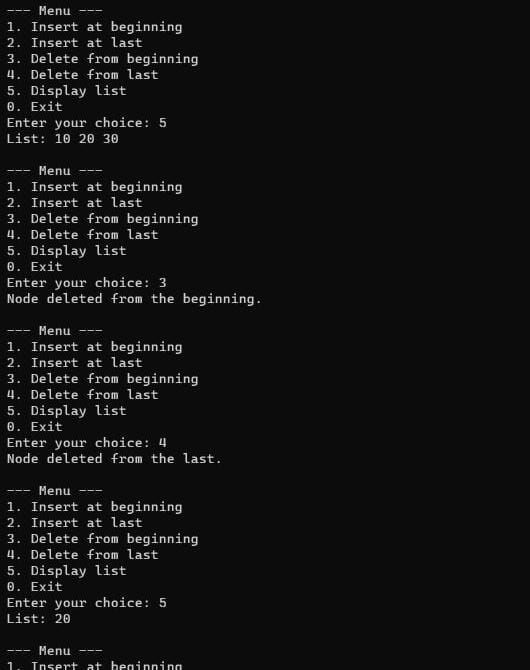
**}**

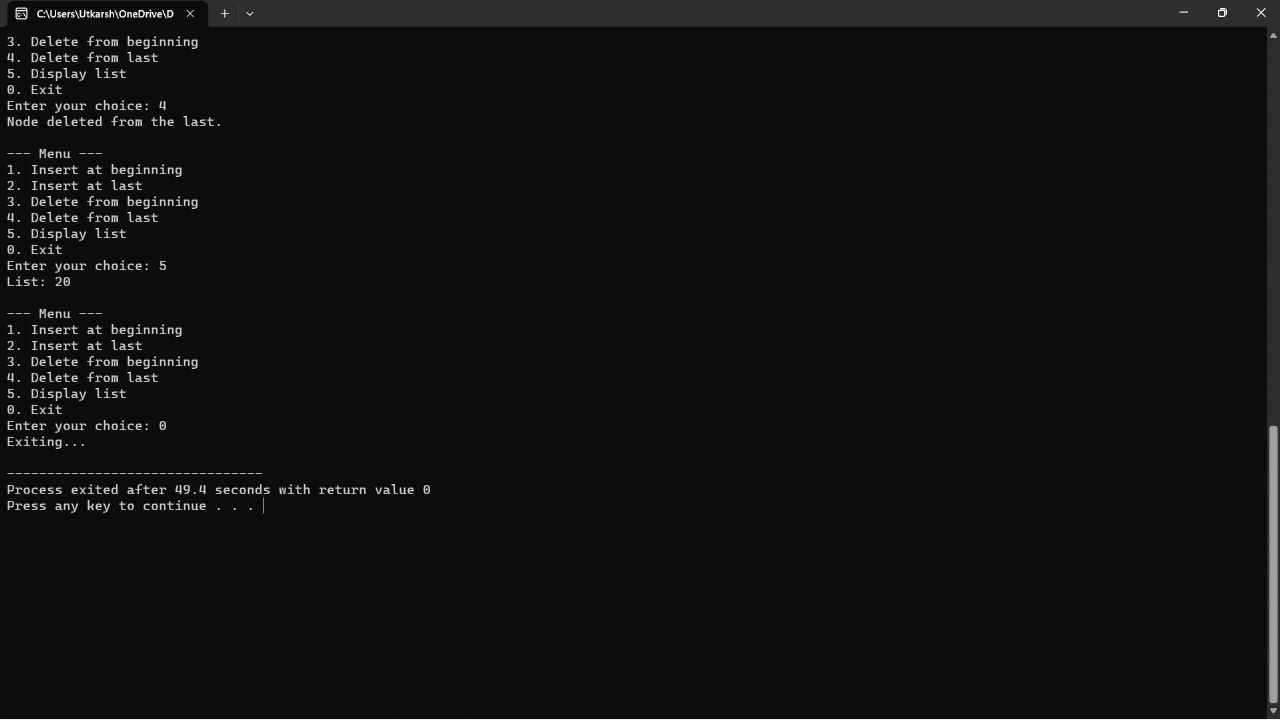
**} while (choice != 0);**

**return 0;**

**Outcomes** **:-**







**Program No:- 09**

**Aim/Title:**  WAP for Tower of Hanoi using recursion.

**Date:**

**Tool:**

**Algorithm/Flow Chart:-**

**Algorithm:-**

1. Define a function towerOfHanoi that takes four parameters:

* **n**: the number of disks to be moved
* **source**: the tower from which disks are initially placed
* **auxiliary**: the auxiliary tower used for intermediate moves
* **destination**: the tower where disks are to be moved

1. Inside the towerOfHanoi function:

* If n is equal to 1 (base case), then:

1. Print the move of the disk from the source tower to the destination tower
2. Return from the function.

* Otherwise (recursive case), do the following:

1. Recursively call the towerOfHanoi function with n-1 disks, moving them from the source tower to the auxiliary tower.
2. Print the move of the nth disk from the source tower to the destination tower.
3. Recursively call the towerOfHanoi function with n-1 disks, moving them from the auxiliary tower to the destination tower.
4. In the main function:

 Prompt the user to enter the number of disks.

 Read the input value and store it in a variable **numDisks**.

 Print the message indicating the start of the Tower of Hanoi solution.

 Call the **towerOfHanoi** function with **numDisks** disks, using towers A, B, and C as the source, auxiliary, and destination towers, respectively.

**Source Code:-**

**#include <stdio.h>**

**void towerOfHanoi(int n, char source, char auxiliary, char destination) {**

**if (n == 1) {**

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

**return;**

**}**

**towerOfHanoi(n - 1, source, destination, auxiliary);**

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

**towerOfHanoi(n - 1, auxiliary, source, destination);**

**}**

**int main() {**

**int numDisks;**

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

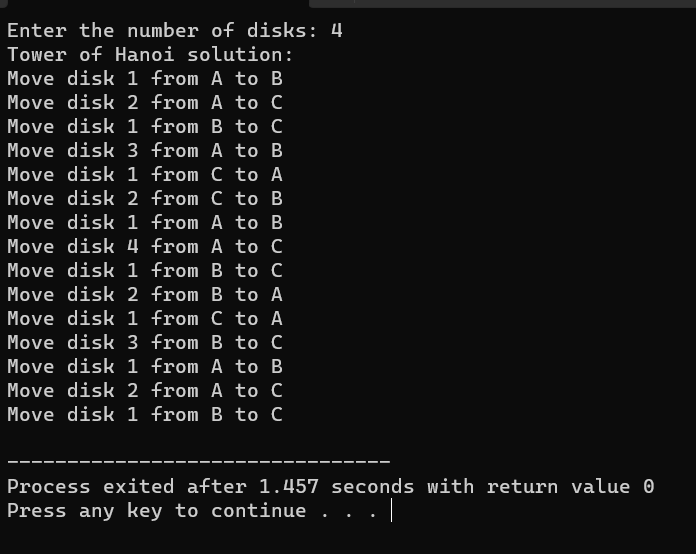
**scanf("%d", &numDisks);**

**printf("Tower of Hanoi solution:\n");**

**towerOfHanoi(numDisks, 'A', 'B', 'C');**

**return 0;**

**}**

**Outcomes :-** ****

**Program No:- 10**

**Aim/Title:** iilustrate queue implementation using array in c with following operation as enQueue, deQueue, isEmpty, displayQueue.

**Date:**

**Tool:**

**Algorithm/Flow Chart:-**

**Algorithm:-**

1. Create a structure Queue with an array items, front, and rear variables.
2. Initialize the queue by setting front and rear to -1.
3. Implement the isEmpty function to check if the queue is empty by comparing front with -1.
4. Implement the isFull function to check if the queue is full by comparing rear with the maximum size minus 1.
5. Implement the enQueue function to add an element to the queue:
   * Check if the queue is full. If it is, display an error message and return.
   * If the queue is empty, set **front** to 0.
   * Increment **rear** by 1 and assign the given value to **items[rear]**.
6. Implement the deQueue function to remove an element from the queue:
   * Check if the queue is empty. If it is, display an error message and return -1.
   * Retrieve the value at items[front].
   * Increment front by 1.
   * If front is greater than rear, reset the queue by setting front and rear to -1.
   * Return the dequeued value.
7. Implement the displayQueue function to display the elements in the queue:
   * Check if the queue is empty. If it is, display an appropriate message.
   * Iterate from **front** to **rear** and print each element.

**Source Code:-**

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**// Structure to represent a queue**

**typedef struct {**

**int items[MAX\_SIZE];**

**int front;**

**int rear;**

**} Queue;**

**// Function to initialize a queue**

**void initializeQueue(Queue\* queue) {**

**queue->front = -1;**

**queue->rear = -1;**

**}**

**// Function to check if the queue is empty**

**int isEmpty(Queue\* queue) {**

**return queue->front == -1;**

**}**

**// Function to check if the queue is full**

**int isFull(Queue\* queue) {**

**return queue->rear == MAX\_SIZE - 1;**

**}**

**// Function to add an element to the queue (enQueue)**

**void enQueue(Queue\* queue, int value) {**

**if (isFull(queue)) {**

**printf("Queue is full. Cannot enqueue %d\n", value);**

**return;**

**}**

**if (isEmpty(queue)) {**

**queue->front = 0;**

**}**

**queue->rear++;**

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

**printf("%d enqueued to the queue.\n", value);**

**}**

**// Function to remove an element from the queue (deQueue)**

**int deQueue(Queue\* queue) {**

**if (isEmpty(queue)) {**

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

**return -1;**

**}**

**int value = queue->items[queue->front];**

**queue->front++;**

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

**// Reset the queue**

**queue->front = -1;**

**queue->rear = -1;**

**}**

**return value;**

**}**

**// Function to display the elements in the queue**

**void displayQueue(Queue\* queue) {**

**if (isEmpty(queue)) {**

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

**return;**

**}**

**printf("Elements in the queue: ");**

**for (int i = queue->front; i <= queue->rear; i++) {**

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

**}**

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

**}**

**int main() {**

**Queue queue;**

**initializeQueue(&queue);**

**enQueue(&queue, 10);**

**enQueue(&queue, 20);**

**enQueue(&queue, 30);**

**displayQueue(&queue);**

**int dequeuedElement = deQueue(&queue);**

**if (dequeuedElement != -1) {**

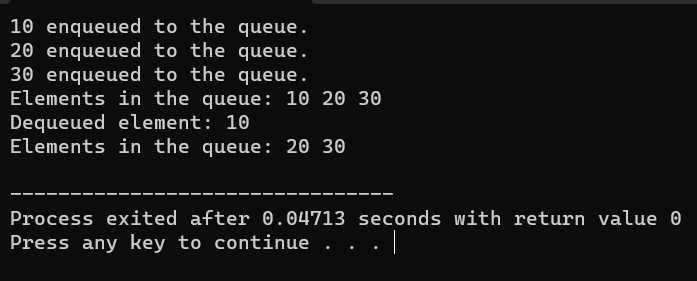
**printf("Dequeued element: %d\n", dequeuedElement);**

**}**

**displayQueue(&queue);**

**return 0;**

**}**

**Outcomes :-**