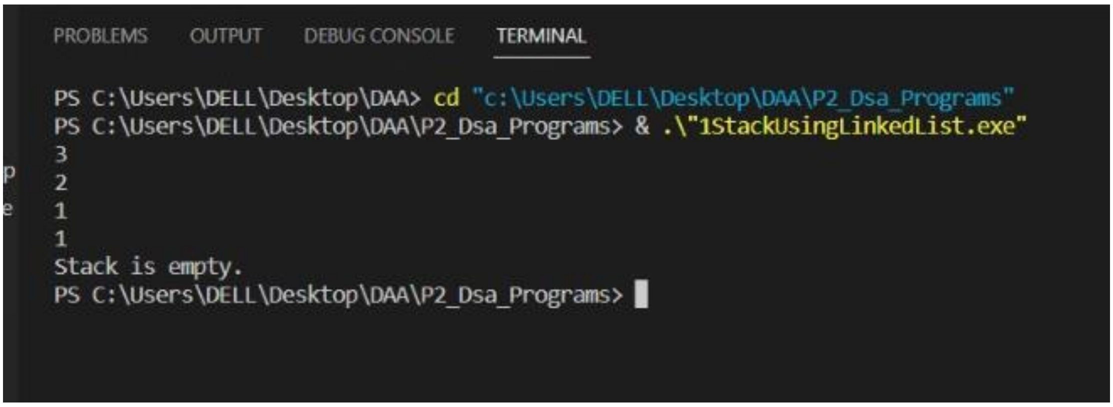
PRACTICAL 02

Write C/C++ code to implement concept of

1)Stack using linked list

```
#include <iostream>
struct Node {
  int data;
  Node *next;
};
class Stack {
private:
  Node *top;
public:
  Stack() : top(nullptr) {}
  void push(int value) {
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = top;
    top = newNode;
  }
  void pop() {
    if (top == nullptr) {
      std::cout << "Stack is empty." << std::endl;
      return;
    }
    Node *temp = top;
    top = top->next;
    delete temp;
  }
  int peek() {
    if (top == nullptr) {
      std::cout << "Stack is empty." << std::endl;
      return -1;
    }
    return top->data;
  }
  bool isEmpty() {
    return top == nullptr;
  }
};
int main() {
  Stack stack;
  stack.push(1);
  stack.push(2);
  stack.push(3);
  std::cout << stack.peek() << std::endl;
  stack.pop();
  std::cout << stack.peek() << std::endl;
  stack.pop();
  std::cout << stack.peek() << std::endl;
  stack.pop();
  std::cout << stack.isEmpty() << std::endl;
  stack.pop();
  return 0;  }
```

Output:

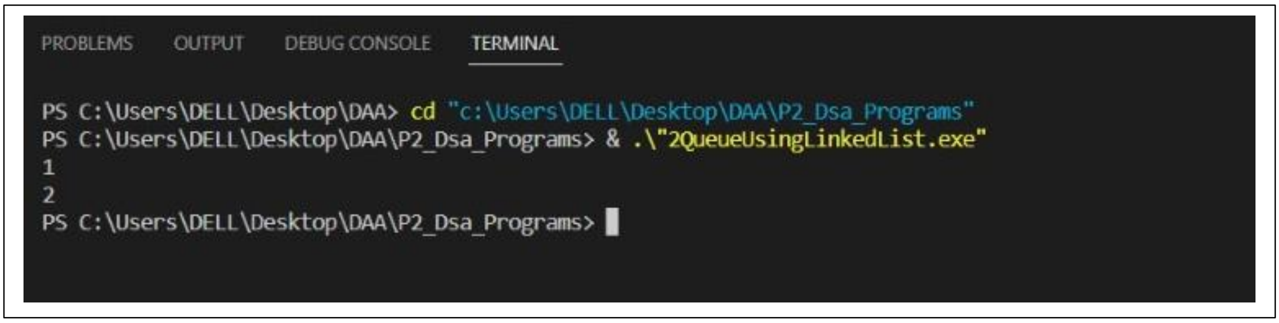


```
PS C:\Users\DELL\Desktop\DAA> cd "c:\Users\DELL\Desktop\DAA\P2_Dsa_Programs"
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs> & .\"1StackUsingLinkedList.exe"
3
2
1
1
Stack is empty.
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs>
```

2) Queue using linked list

```
#include <iostream>
struct Node {
  int data;
  Node *next;
};
class Queue {
private:
  Node *front;
  Node *rear;
public:
  Queue() : front(nullptr), rear(nullptr) {}
  void enqueue(int value) {
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;
    if (rear == nullptr) {
      front = newNode;
      rear = newNode;
      return;
    }
    rear->next = newNode;
    rear = newNode;
  }
  void dequeue() {
    if (front == nullptr) {
      std::cout << "Queue is empty." << std::endl;
      return;
    }
    Node *temp = front;
    front = front->next;
    if (front == nullptr) {
      rear = nullptr;
    }
    delete temp;
  }
  int peek() {
    if (front == nullptr) {
      std::cout << "Queue is empty." << std::endl;
      return -1;
    }
    return front->data;
  }
  bool isEmpty() {
    return front == nullptr;
  }
};
int main() {
  Queue queue;
  queue.enqueue(1);
  queue.enqueue(2);
  queue.enqueue(3);
  std::cout << queue.peek() << std::endl;
  queue.dequeue();
  std::cout << queue.peek() << std::endl;
  queue.dequeue();
  return 0;
}
```

Output:



```
PS C:\Users\DELL\Desktop\DAA> cd "c:\Users\DELL\Desktop\DAA\P2_Dsa_Programs"
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs> & .\"2QueueUsingLinkedList.exe"
1
2
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs> 
```

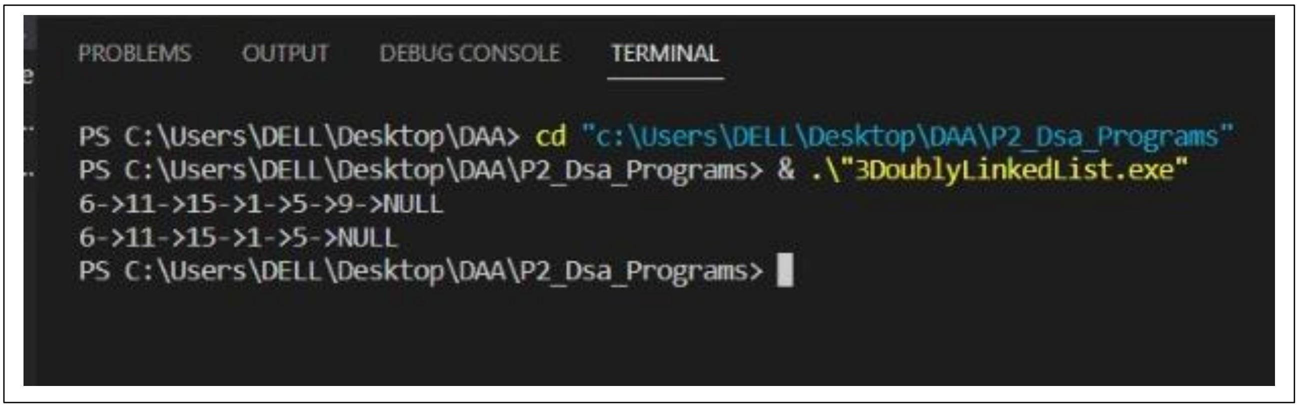
3) Doubly Linked List

```
#include <iostream>
using namespace std;

// node creation
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

// insert node at the front
void insertFront(struct Node** head, int data) {
  // allocate memory for newNode
  struct Node* newNode = new Node;

  // assign data to newNode
  newNode->data = data;

  // make newNode as a head
  newNode->next = (*head);

  // assign null to prev
  newNode->prev = NULL;

  // previous of head (now head is the second node) is newNode
  if ((*head) != NULL)
    (*head)->prev = newNode;

  // head points to newNode
  (*head) = newNode;
}
```

```
// insert a node after a specific node
void insertAfter(struct Node* prev_node, int data) {
  // check if previous node is null
  if (prev_node == NULL) {
    cout << "previous node cannot be null";
    return;
  }

  // allocate memory for newNode
  struct Node* newNode = new Node;

  // assign data to newNode
  newNode->data = data;

  // set next of newNode to next of prev node
  newNode->next = prev_node->next;

  // set next of prev node to newNode
  prev_node->next = newNode;

  // set prev of newNode to the previous node
  newNode->prev = prev_node;

  // set prev of newNode's next to newNode
  if (newNode->next != NULL)
    newNode->next->prev = newNode;
}

// insert a newNode at the end of the list
void insertEnd(struct Node** head, int data) {
  // allocate memory for node
  struct Node* newNode = new Node;
  // assign data to newNode
  newNode->data = data;

  // assign null to next of newNode
  newNode->next = NULL;

  // store the head node temporarily (for later use)
  struct Node* temp = *head;

  // if the linked list is empty, make the newNode as head node
  if (*head == NULL) {
    newNode->prev = NULL;
    *head = newNode;
    return;
  }

  // if the linked list is not empty, traverse to the end of the linked list
  while (temp->next != NULL)
    temp = temp->next;

  // now, the last node of the linked list is temp

  // assign next of the last node (temp) to newNode
  temp->next = newNode;

  // assign prev of newNode to temp
  newNode->prev = temp;
}

// delete a node from the doubly linked list
void deleteNode(struct Node** head, struct Node* del_node) {
  // if head or del is null, deletion is not possible
  if (*head == NULL || del_node == NULL)
```

```
    return;

  // if del_node is the head node, point the head pointer to the next of del_node
  if (*head == del_node)
    *head = del_node->next;

  // if del_node is not at the last node, point the prev of node next to del_node to the previous of del_node
  if (del_node->next != NULL)
    del_node->next->prev = del_node->prev;

  // if del_node is not the first node, point the next of the previous node to the next node of del_node
  if (del_node->prev != NULL)
    del_node->prev->next = del_node->next;

  // free the memory of del_node
  free(del_node);
}

// print the doubly linked list
void displayList(struct Node* node) {
  struct Node* last;

  while (node != NULL) {
    cout << node->data << "->";
    last = node;
    node = node->next;
  }
  if (node == NULL)
    cout << "NULL\n";
}

int main() {
  // initialize an empty node
  struct Node* head = NULL;

  insertEnd(&head, 5);
  insertFront(&head, 1);
  insertFront(&head, 6);
  insertEnd(&head, 9);

  // insert 11 after head
  insertAfter(head, 11);

  // insert 15 after the seond node
  insertAfter(head->next, 15);

  displayList(head);

  // delete the last node
  deleteNode(&head, head->next->next->next->next->next);

  displayList(head);
}
```

Output:-



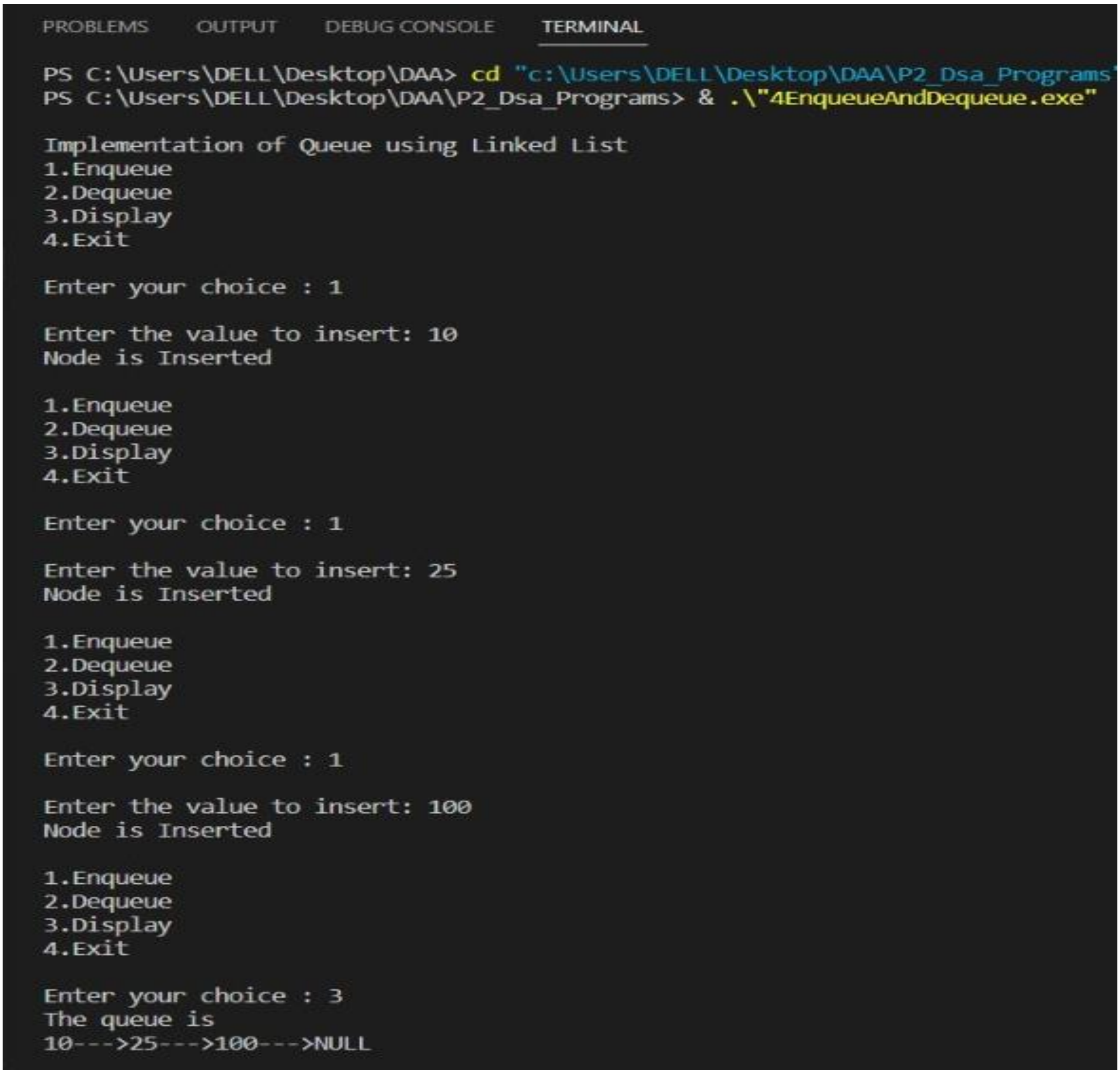
```
PS C:\Users\DELL\Desktop\DAA> cd "c:\Users\DELL\Desktop\DAA\P2_Dsa_Programs"
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs> & .\"3DoublyLinkedList.exe"
6->11->15->1->5->9->NULL
6->11->15->1->5->NULL
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs>
```

4) Enqueue And Dequeue

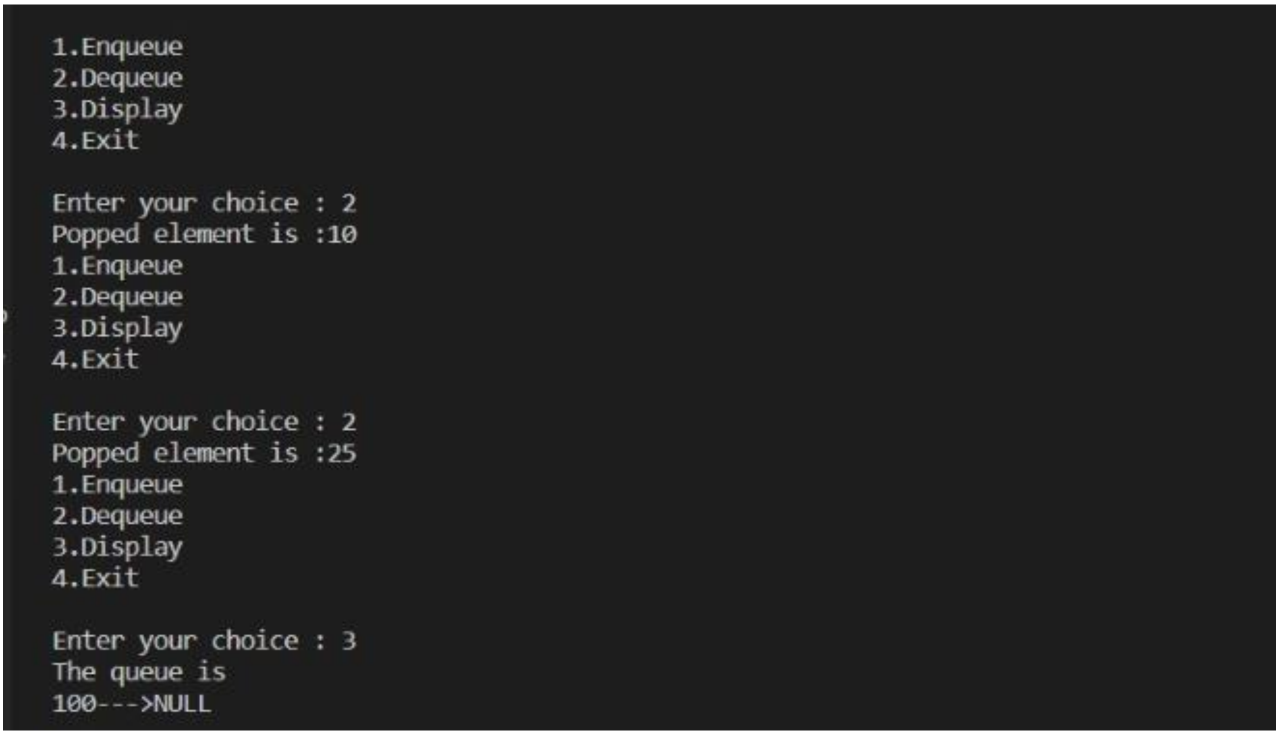
```
#include < stdio.h >
#include < stdlib.h >
// Structure to create a node with data and the next pointer
struct node {
    int data;
    struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;
// Enqueue() operation on a queue
void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr - > data = value;
    ptr - > next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear - > next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}

// Dequeue() operation on a queue
int dequeue() {
    if (front == NULL) {
        printf("\nUnderflow\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front - > data;
        front = front - > next;
        free(temp);
        return temp_data;
    }
}

// Display all elements of the queue
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp - > data);
            temp = temp - > next;
        }
        printf("NULL\n\n");
    }
}

int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
```

```
   while (choice != 4) {
      printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
      printf("\nEnter your choice : ");
      scanf("%d", & choice);

switch (choice) {
         case 1:
            printf("\nEnter the value to insert: ");
            scanf("%d", & value);
            enqueue(value);
            break;
         case 2:
            printf("Popped element is :%d\n", dequeue());
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
            break;
         default:
            printf("\nWrong Choice\n");
      }
   }
   return 0;
}
```

Output:-



```
PS C:\Users\DELL\Desktop\DAA> cd "c:\Users\DELL\Desktop\DAA\P2_Dsa_Programs"
PS C:\Users\DELL\Desktop\DAA\P2_Dsa_Programs> & .\"4EnqueueAndDequeue.exe"

Implementation of Queue using Linked List
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 1

Enter the value to insert: 10
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 1

Enter the value to insert: 25
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 1

Enter the value to insert: 100
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 3
The queue is
10--->25--->100--->NULL
```



```
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 2
Popped element is :10
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 2
Popped element is :25
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter your choice : 3
The queue is
100--->NULL
```