NAME	**:**	**AHMED JAMSHED**
ROLL NO	**:**	**SP22-BCS-001**
DATE	**:**	**10-10-2023**
ASSIGNMENT NO	**:**	**#2**
SUBJECT	**:**	**DSA lab**
SUBMITTED TO	**:**	**Ma’am Yasmeen Jana**

COMSATS UNIVERSITY ISLAMABAD VEHARI CAMPUS

Quention 1

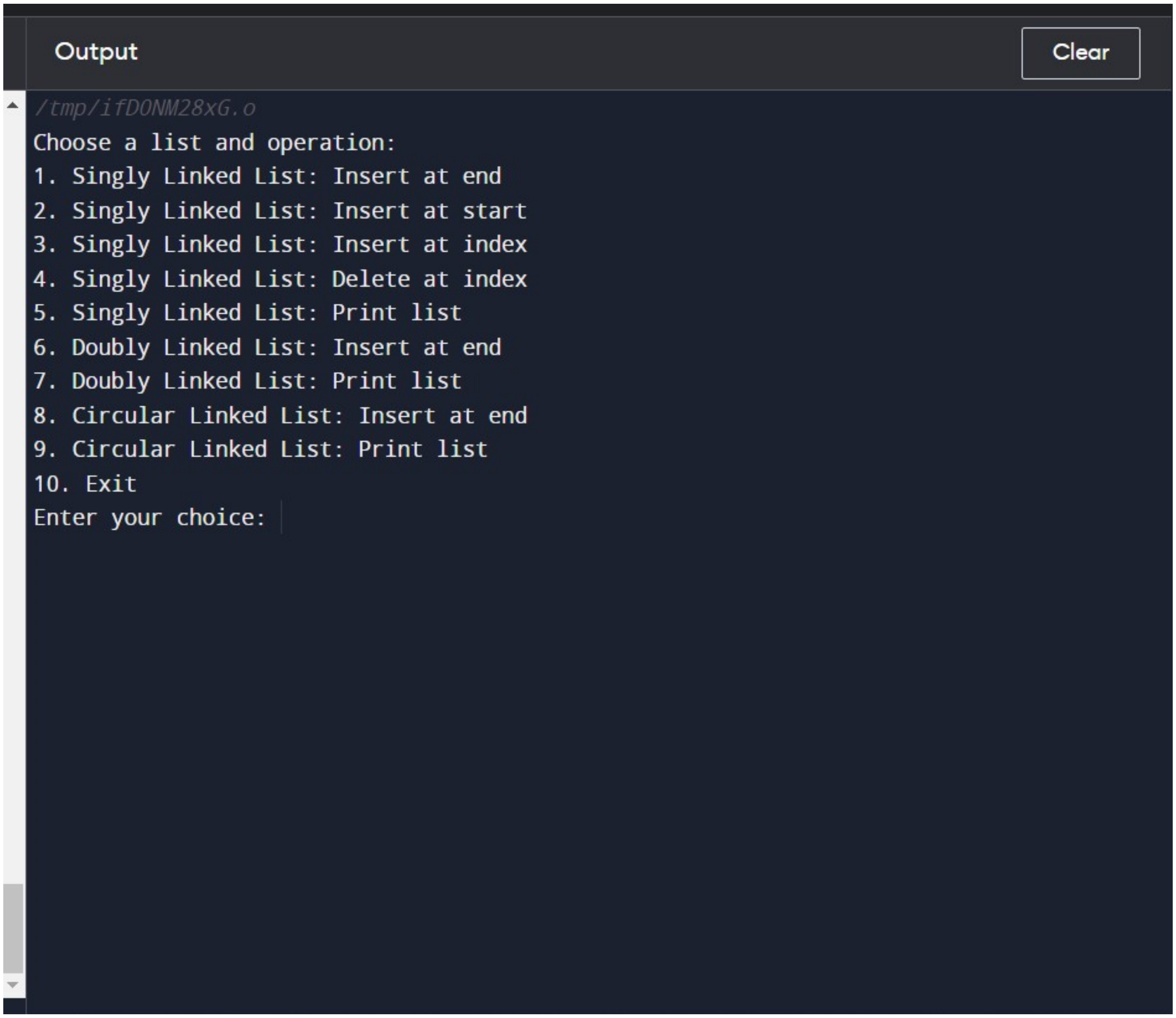
```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = NULL;
    }
};

class DoublyNode : public Node {
public:
    Node* prev;

    DoublyNode(int value) : Node(value) {
        prev = NULL;
    }
};

class CircularNode : public Node {
public:
    CircularNode(int value) : Node(value) {}
};

class LinkedList {
protected:
    Node* head;

public:
    LinkedList() {
        head = NULL;
    }

    // Function to add a node at the end of the list
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == NULL) {
```

```
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

// Function to add a node at the beginning of the list
void insertAtStart(int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

// Function to add a node at a specific index
void insertAtIndex(int value, int index) {
    if (index < 0) {
        cout << "Invalid index. Cannot insert at a negative index." << endl;
        return;
    }

    Node* newNode = new Node(value);
    if (index == 0) {
        newNode->next = head;
        head = newNode;
    } else {
        Node* current = head;
        int currentIndex = 0;
        while (current != NULL && currentIndex < index - 1) {
            current = current->next;
            currentIndex++;
        }
        if (current == NULL) {
            cout << "Invalid index. Cannot insert at the specified index." << endl;
            return;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
}
```

```
    // Function to delete a node at a specific index
    void deleteAtIndex(int index) {
        if (index < 0) {
            cout << "Invalid index. Cannot delete at a negative index." << endl;
            return;
        }

        if (head == NULL) {
            cout << "List is empty. Cannot delete from an empty list." << endl;
            return;
        }

        if (index == 0) {
            Node* temp = head;
            head = head->next;
            delete temp;
        } else {
            Node* current = head;
            int currentIndex = 0;
            while (current->next != NULL && currentIndex < index - 1) {
                current = current->next;
                currentIndex++;
            }
            if (current->next == NULL) {
                cout << "Invalid index. Cannot delete at the specified index." << endl;
                return;
            }
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
        }
    }

    // Function to print the entire linked list
    void printList() {
        Node* current = head;
        while (current != NULL) {
            cout << current->data << " -> ";
            current = current->next;
        }
        cout << "nullptr" << endl;
    }
};
```

```
class DoublyLinkedList : public LinkedList {
public:
    DoublyLinkedList() : LinkedList() {}

    // Function to add a node at the end of the doubly linked list
    void insertAtEnd(int value) {
        DoublyNode* newNode = new DoublyNode(value);
        if (head == NULL) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newNode;
            newNode->prev = current;
        }
    }
};

class CircularLinkedList : public LinkedList {
public:
    CircularLinkedList() : LinkedList() {}

    // Function to add a node at the end of the circular linked list
    void insertAtEnd(int value) {
        Node* newNode = new CircularNode(value);
        if (head == NULL) {
            head = newNode;
            newNode->next = newNode; // Point to itself for circularity
        } else {
            Node* current = head;
            while (current->next != head) {
                current = current->next;
            }
            current->next = newNode;
            newNode->next = head; // Make it circular
        }
    }
};

int main() {
    LinkedList myList;
    DoublyLinkedList myDoublyList;
```

```
CircularLinkedList myCircularList;

while (true) {
    int choice;
    cout << "Choose a list and operation:" << endl;
    cout << "1. Singly Linked List: Insert at end" << endl;
    cout << "2. Singly Linked List: Insert at start" << endl;
    cout << "3. Singly Linked List: Insert at index" << endl;
    cout << "4. Singly Linked List: Delete at index" << endl;
    cout << "5. Singly Linked List: Print list" << endl;
    cout << "6. Doubly Linked List: Insert at end" << endl;
    cout << "7. Doubly Linked List: Print list" << endl;
    cout << "8. Circular Linked List: Insert at end" << endl;
    cout << "9. Circular Linked List: Print list" << endl;
    cout << "10. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    int value, index;

    switch (choice) {
        case 1:
            cout << "Enter value to insert at end: ";
            cin >> value;
            myList.insertAtEnd(value);
            break;
        case 2:
            cout << "Enter value to insert at start: ";
            cin >> value;
            myList.insertAtStart(value);
            break;
        case 3:
            cout << "Enter value to insert: ";
            cin >> value;
            cout << "Enter index to insert at: ";
            cin >> index;
            myList.insertAtIndex(value, index);
            break;
        case 4:
            cout << "Enter index to delete: ";
            cin >> index;
            myList.deleteAtIndex(index);
            break;
        case 5:
                cout << "Singly Linked List: ";
                myList.printList();
                break;
            case 6:
                cout << "Enter value to insert at end: ";
                cin >> value;
                myDoublyList.insertAtEnd(value);
                break;
            case 7:
                cout << "Doubly Linked List: ";
                myDoublyList.printList();
                break;
            case 8:
                cout << "Enter value to insert at end: ";
                cin >> value;
                myCircularList.insertAtEnd(value);
                break;
            case 9:
                cout << "Circular Linked List: ";
                myCircularList.printList();
                break;
            case 10:
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }

    return 0;
}
```

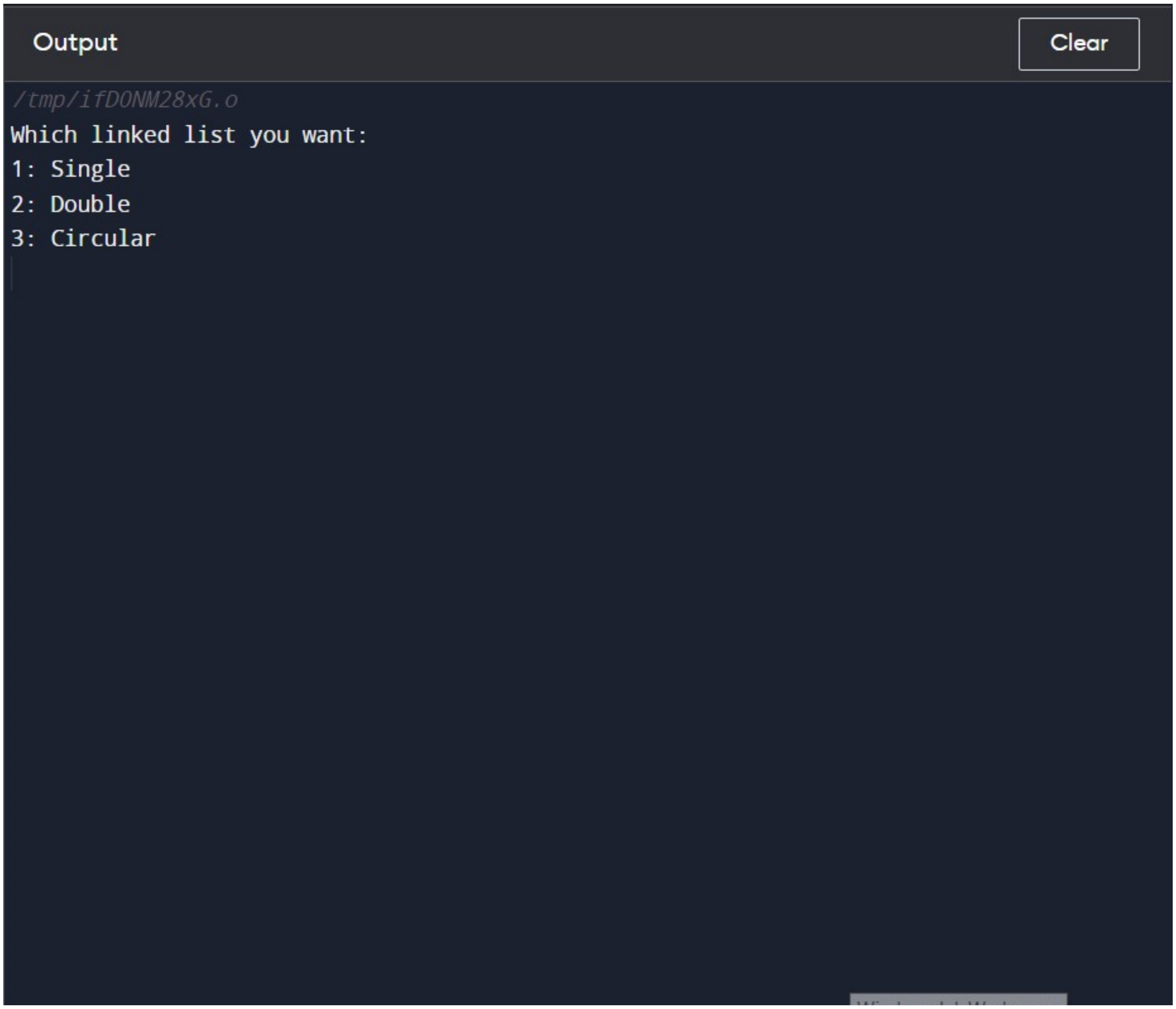


```
/tmp/ifDONM28xG.o
Choose a list and operation:
1. Singly Linked List: Insert at end
2. Singly Linked List: Insert at start
3. Singly Linked List: Insert at index
4. Singly Linked List: Delete at index
5. Singly Linked List: Print list
6. Doubly Linked List: Insert at end
7. Doubly Linked List: Print list
8. Circular Linked List: Insert at end
9. Circular Linked List: Print list
10. Exit
Enter your choice:
```

Question 2

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

// Define the structure for a node
struct Node {
    int data;
    Node* next;
};

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Linked list class
class LinkedList {
public:
    Node* head;

    LinkedList() : head(nullptr) {}

    // Function to insert a node at the beginning
    void insertAtBeginning(int data) {
        Node* newNode = createNode(data);
        newNode->next = head;
        head = newNode;
        cout << "Insertion at the beginning successful." << endl;
    }

    // Function to insert a node at the end
    void insertAtEnd(int data) {
        Node* newNode = createNode(data);
        if (!head) {
            head = newNode;
            cout << "Insertion at the end successful." << endl;
            return;
        }
        Node* current = head;
        while (current->next) {
```

```
        current = current->next;
    }
    current->next = newNode;
    cout << "Insertion at the end successful." << endl;
}

// Function to delete a node with a specific value
void deleteNode(int data) {
    if (!head) {
        cout << "List is empty. Deletion not possible." << endl;
        return;
    }
    if (head->data == data) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Deletion successful." << endl;
        return;
    }
    Node* current = head;
    while (current->next) {
        if (current->next->data == data) {
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
            cout << "Deletion successful." << endl;
            return;
        }
        current = current->next;
    }
    cout << "Element not found. Deletion not possible." << endl;
}

// Function to reverse the linked list
void reverse() {
    Node* prev = nullptr;
    Node* current = head;
    Node* nextNode = nullptr;

    while (current) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
        }

        head = prev;
        cout << "Reversal successful." << endl;
    }

    // Function to display the linked list
    void display() {
        Node* current = head;
        while (current) {
            cout << current->data << " -> ";
            current = current->next;
        }
        cout << "nullptr" << endl;
    }
};

int main() {
    LinkedList singleLinkedList;

    while (true) {
        cout << "Which linked list you want:" << endl;
        cout << "1: Single" << endl;
        cout << "2: Double" << endl;
        cout << "3: Circular" << endl;

        int listChoice;
        cin >> listChoice;

        if (listChoice == 1) {
            int choice;
            while (true) {
                cout << "\nSingle Linked List Operations:" << endl;
                cout << "1: Insertion" << endl;
                cout << "2: Deletion" << endl;
                cout << "3: Display" << endl;
                cout << "4: Reverse" << endl;
                cout << "5: Seek" << endl;
                cout << "6: Exit" << endl;

                cin >> choice;

                if (choice == 1) {
                    int insertionChoice;
                    cout << "Insertion Options:" << endl;
                    cout << "1: Insertion at Beginning" << endl;
                    cout << "2: Insertion at End" << endl;
                    cin >> insertionChoice;

                    if (insertionChoice == 1) {
                        int data;
                        cout << "Enter data: ";
                        cin >> data;
                        singleLinkedList.insertAtBeginning(data);
                    } else if (insertionChoice == 2) {
                        int data;
                        cout << "Enter data: ";
                        cin >> data;
                        singleLinkedList.insertAtEnd(data);
                    }
                } else if (choice == 2) {
                    int data;
                    cout << "Enter data to delete: ";
                    cin >> data;
                    singleLinkedList.deleteNode(data);
                } else if (choice == 3) {
                    singleLinkedList.display();
                } else if (choice == 4) {
                    singleLinkedList.reverse();
                } else if (choice == 5) {
                    // Handle seek option
                } else if (choice == 6) {
                    // Exit the program
                    return 0;
                } else {
                    cout << "Invalid choice. Please enter a valid option." << endl;
                }
            }
        }
    }

    return 0;
}
```



```
/tmp/ifDONM28xG.o
Which linked list you want:
1: Single
2: Double
3: Circular
```