PROGRAM 1

Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.

ALGORITHM

```
1. For Linear Search:
        LinearSearch(arr, target):
        for each element in arr with index i do:
            if arr[i] is equal to target then
                return i  //if Target found at index i
        end for
        return -1  //if Target not found in the array
2. Use a sorting algorithm to sort the array before applying Binary Search.
3. For Binary Search:
        BinarySearch(arr, target):
        left = 0
        right = length of arr - 1
        while left <= right do:
            mid = low + (high - low) / 2 //to avoid overflow
            if arr[mid] is equal to target then
                return mid //Target found at index mid
            else if arr[mid] < target then
                left = mid + 1 //Search the right half
            else
                right = mid - 1 //Search the left half
        return -1 Target not found in the array
4. Give choice to user and call the functions
```

PROGRAM CODE

```
#include<iostream>
#include<vector>
using namespace std;
template <typename T>
int linearSearch(vector<T>& arr, T& target){
    for(int i=0;i<arr.size();i++){
        if(arr[i]==target)
            return i;
    }
    return -1;
}
template <typename T>
int binarySearch(vector<T>& arr, T& target){
    int hole, value;
    for(int i=0;i<arr.size();i++){
        value = arr[i];
        hole = i - 1;
        while(hole>=0 && arr[hole]>value){
            arr[hole+1] = arr[hole];
            hole = hole - 1;
        }
```

```
            arr[hole+1]=value;
        }
        int low = 0;
        int high = arr.size()-1;
        while(low<=high){
            int mid = low + (high-low)/2;
            if(arr[mid]==target)
                return mid;
            else if(arr[mid]<target)
                return low = mid + 1;
            else
                return low = mid - 1;
        }
        return -1;
    }
    int main(){
        cout << "Enter the number of elements: ";
        int n;
        cin >> n;
        vector<int> List;
        cout << "Enter the elements of the lists: ";
        int element;
        for(int i=0;i<n;i++){
            cin >> element;
            List.push_back(element);
        }
        cout << "Enter the element to search for: ";
        int target;
        cin >> target;
        cout << "Choose Searching Algorithm:\n";
        cout << "1. Linear Search\n";
        cout << "2. Binary Search\n";
        int choice;
        cout << "Enter Choice: ";
        cin >> choice;
        int result;
        switch(choice){
            case 1:
                result = linearSearch(List,target);
                break;
            case 2:
                result = binarySearch(List,target);
                break;
            default:
                cout << "Invalid Choice. Exit Program.";
                return 1;
        }
        if(result != -1)
            cout << "Element " << target << " found at index " << result << endl;
        else
```

```
                cout << "Element " << target << " not found in the list." << endl;
        return 0;
}
```

OUTPUT

SET 1:

```
Enter the number of elements: 5
Enter the elements of the lists: 3 2 4 6 1
Enter the element to search for: 2
Choose Searching Algorithm:
1. Linear Search
2. Binary Search
Enter Choice: 1
Element 2 found at index 1
```

SET 2:

```
Enter the number of elements: 3
Enter the elements of the lists: 66 8 999
Enter the element to search for: 8
Choose Searching Algorithm:
1. Linear Search
2. Binary Search
Enter Choice: 2
Element 8 found at index 0
```

PROGRAM 2

WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

ALGORITHM

```
1.  For Insertion Sort:
        InsertionSort(arr):
        n = length of arr
        for i from 1 to n - 1 do:
            key = arr[i]
            j = i – 1 //Move elements of arr[0..i-1] that are greater than key to one
            position ahead of their current position
            while j >= 0 and arr[j] > key do:
                arr[j + 1] = arr[j]
                j = j - 1
            arr[j + 1] = key
2.  For Selection Sort:
        SelectionSort(arr):
        n = length of arr
        for i from 0 to n-2 do:
         //Assume the current index is the minimum
            minIndex = i
        //Find the index of the minimum element in the unsorted part of the array
        for j from i+1 to n-1 do:
                if arr[j] < arr[minIndex] then
                    minIndex = j   //Swap the found minimum element with the
                    first element in the unsorted part
        swap arr[i] and arr[minIndex]
3.  For Bubble Sort:
        BubbleSort(arr):
        n = length of arr
        for i from 0 to n-1 do:
        // Last i elements are already sorted, so we don't need to check them
            for j from 0 to n-i-1 do:
            // Swap if the element found is greater than the next element
                if arr[j] > arr[j+1] then
                    swap arr[j] and arr[j+1]
4.  Open main function and give user choice to perform sorting and call the functions
    according to choice.
```

PROGRAM CODE

```
#include<bits/stdc++.h>
#include<vector>
using namespace std;
template <typename T>
void insertionSort(vector<T>& arr){
    int hole, value;
    for(int i=1;i<arr.size();i++)
    {
```

```
                value = arr[i];
                hole = i - 1;
                while(hole>=0 && arr[hole]>value){
                        arr[hole+1] = arr[hole];
                        hole = hole-1;
                }
                arr[hole+1] = value;
        }
}
template <typename T>
void selectionSort(vector<T>& arr){
        int minDex, i;
        for(i=0;i<arr.size()-1;i++){
                minDex=i;
                for(int j=i+1;j<arr.size();j++){
                        if(arr[j]<arr[minDex])
                                minDex=j;
                }
                int temp = arr[minDex];
                arr[minDex]=arr[i];
                arr[i]=temp;
        }
}
template <typename T>
void bubbleSort(vector<T>& arr){
        for(int i=0;i<arr.size()-1;i++){
                for(int j=0;j<arr.size()-1;j++){
                        if(arr[j]>arr[j+1]){
                                int temp = arr[j];
                                arr[j]=arr[j+1];
                                arr[j+1]=temp;
                        }
                }
        }
}
template <typename T>
void printArray(vector<T>& arr){
        int i;
        for(i=0;i<arr.size();i++)
        {
                cout<<arr[i] << " ";
        }
        cout << endl;
}
int main(){
        cout << "Enter the number of elements: ";
        int n;
        cin >> n;
        vector<int> List;
        cout << "Enter the elements of the lists: ";
        int element;
        for(int i=0;i<n;i++){
                cin >> element;
                List.push_back(element);
        }
        cout << "Unsorted Array: ";
        printArray(List);
        cout << "Choose Sorting Algorithm:\n";
        cout << "1. Insertion Sort\n";
        cout << "2. Selection Sort\n";
        cout << "3. Bubble Sort\n";
        int choice;
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice) {
        case 1:
                insertionSort(List);
                break;
        case 2:
                selectionSort(List);
                break;
        case 3:
                bubbleSort(List);
                break;
        default:
                cout << "Invalid Choice. Exit Program.";
                return 1;
        }
        cout << "Sorted Array: ";
        printArray(List);
        return 0;
}
```

OUTPUT

SET 1:

```
Enter the number of elements: 5
Enter the elements of the lists: 2 6 1 88 9
Unsorted Array: 2 6 1 88 9
Choose Sorting Algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 1
Sorted Array: 1 2 6 9 88
```

SET 2:

```
Enter the number of elements: 4
Enter the elements of the lists: 33 6 999 1
Unsorted Array: 33 6 999 1
Choose Sorting Algorithm:
```

```
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 2
Sorted Array: 1 6 33 999
```

SET 3:

```
Enter the number of elements: 3
Enter the elements of the lists: 23 11 898
Unsorted Array: 23 11 898
Choose Sorting Algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 3
Sorted Array: 11 23 898
```

PROGRAM 3

Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

ALGORITHM

1. Node Class:
    - Define a class Node with data, prev, and next.
    - Implement constructor and destructor.
2. Display Function (display):
    - Initialize temp to head.
    - While temp not null, print data, move temp to next.
3. Get Length Function (getLength):
    - Initialize len to 0, temp to head.
    - While temp not null, increment len, move temp to next.
    - Return len.
4. Insert at Head Function (insertAtHead):
    - If head is null, create new node, set head and tail.
    - Else, create new node, set next and prev pointers.
5. Insert at Tail Function (insertAtTail):
    - If tail is null, create new node, set head and tail.
    - Else, create new node, set prev and next pointers.
6. Insert at Position Function (insertAtPosition):
    - Check if position valid; if not, print error and return.
    - If position is 1, call insertAtHead.
    - If position is length + 1, call insertAtTail.
    - Else, find node at position, create new node, adjust pointers.
7. Delete Node Function (deleteNode):
    - Check if position valid; if not, print error and return.
    - If position is 1, update pointers, delete head.
    - Else, find node at position, adjust pointers, delete node.
8. Search Node Function (searchNode):
    - Initialize temp to head, pos to 1.
    - While temp not null, if data matches target, return position; else, move temp to next, increment position.
    - If no match, return -1.
9. Reverse List Function (reverseList):
    - Initialize current to head, prevNode and nextNode to null.
    - While current not null, reverse pointers, update prevNode, nextNode, and current.
    - Update head and tail pointers.
10. Main Function:

- Initialize head and tail for integer and string lists.
- Enter loop to display menu, read choice, perform list operation.
- Continue loop until choice is 0.

PROGRAM CODE

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
class Node {
        public:
          T data;
          Node* prev;
          Node* next;
          Node(T data) : data(data), next(nullptr), prev(nullptr) {}
          ~Node() {
                if (next != nullptr) {
                        delete next;
                        next = nullptr;
                }
                cout << "Memory freed with data " << data << endl;
        }
};
template <typename T>
void display(Node<T>*& head) {
        Node<T>* temp = head;
        while (temp != nullptr) {
                cout << temp->data << " ";
                temp = temp->next;
        }
        cout << endl;
}
template <typename T>
int getLength(Node<T>* head) {
        int len = 0;
        Node<T>* temp = head;
        while (temp != nullptr) {
                len++;
                temp = temp->next;
        }
        return len;
}
template <typename T>
void insertAtHead(Node<T>*& head, Node<T>*& tail, T data) {
        if (head == nullptr) {
                Node<T>* temp = new Node<T>(data);
                head = temp;
                tail = temp;
        } else {
                Node<T>* temp = new Node<T>(data);
        temp->next = head;
        head->prev = temp;
        head = temp;
    }
}
template <typename T>
void insertAtTail(Node<T>*& tail, Node<T>*& head, T data) {
    if (tail == nullptr) {
        Node<T>* temp = new Node<T>(data);
        tail = temp;
        head = temp;
    } else {
        Node<T>* temp = new Node<T>(data);
        tail->next = temp;
        temp->prev = tail;
        tail = temp;
    }
}
template <typename T>
void insertAtPosition(Node<T>*& head, Node<T>*& tail, int pos, T data) {
    if (pos < 1 || pos > getLength(head) + 1) {
        cout << "Invalid position. Insertion failed." << endl;
        return;
    }
    if (pos == 1) {
        insertAtHead(head, tail, data);
    } else if (pos == getLength(head) + 1) {
        insertAtTail(tail, head, data);
    } else {
        Node<T>* temp = head;
        for (int i = 1; i < pos - 1; ++i) {
            temp = temp->next;
        }
        Node<T>* nodeToInsert = new Node<T>(data);
        nodeToInsert->next = temp->next;
        temp->next->prev = nodeToInsert;
        temp->next = nodeToInsert;
        nodeToInsert->prev = temp;
    }
}
template <typename T>
void deleteNode(Node<T>*& head, int pos) {
    if (pos < 1 || pos > getLength(head)) {
        cout << "Invalid position. Deletion failed." << endl;
        return;
    }
    if (pos == 1) {
        Node<T>* temp = head;
        temp->next->prev = nullptr;
        head = temp->next;
        temp->next = nullptr;
        delete temp;
    } else {
        Node<T>* cur = head;
        for (int i = 1; i < pos; ++i) {
            cur = cur->next;
        }
        cur->prev->next = cur->next;
        if (cur->next != nullptr) {
            cur->next->prev = cur->prev;
        }
        cur->next = nullptr;
        delete cur;
    }
}
template <typename T>
int searchNode(Node<T>* head, T target) {
    Node<T>* temp = head;
    int pos = 1;
    while (temp != nullptr) {
        if (temp->data == target) {
            return pos;
        }
        temp = temp->next;
        pos++;
    }
    return -1;
}
template <typename T>
void reverseList(Node<T>*& head, Node<T>*& tail) {
    Node<T>* current = head;
    Node<T>* prevNode = nullptr;
    Node<T>* nextNode = nullptr;
    while (current != nullptr) {
        nextNode = current->next;
        current->next = prevNode;
        current->prev = nextNode;
        prevNode = current;
        current = nextNode;
    }
    tail = head;
    head = prevNode;
}
int main() {
    Node<int>* headInt = nullptr;
    Node<int>* tailInt = nullptr;
    Node<string>* headString = nullptr;
    Node<string>* tailString = nullptr;
    int choice;
    do {
```

```
cout << "\nSelect an option:\n" << "1. Insert at Head\n"
<< "2. Insert at Tail\n"  << "3. Insert at Position\n" << "4. Delete a Node\
n"<< "5. Search for a Node\n" << "6. Reverse List\n" << "7. Display\n" <<
"0. Exit\n" << "Enter your choice: ";
cin >> choice;
switch (choice) {
	case 1: {
		int value;
		cout << "Enter the value to insert at the head: ";
		cin >> value;
		insertAtHead(headInt, tailInt, value);
		cout << "Inserted successfully." << endl;
		break;
	}
	case 2: {
		int value;
		cout << "Enter the value to insert at the tail: ";
		cin >> value;
		insertAtTail(tailInt, headInt, value);
		cout << "Inserted successfully." << endl;
		break;
	}
	case 3: {
		int pos;
		int value;
		cout << "Enter the position to insert at: ";
		cin >> pos;
		cout << "Enter the value to insert: ";
		cin >> value;
		insertAtPosition(headInt, tailInt, pos, value);
		cout << "Inserted successfully." << endl;
		break;
	}
	case 4: {
		int pos;
		cout << "Enter the position to delete from: ";
		cin >> pos;
		deleteNode(headInt, pos);
		cout << "Deleted successfully." << endl;
		break;
	}
	case 5: {
		int value;
		cout << "Enter the value to search for: ";
		cin >> value;
		int position = searchNode(headInt, value);
		if (position != -1) {
			cout << "Value " << value << " found at position " <<
			position << endl;
		} else {
```

```
                    cout << "Value " << value << " not found in the list."
                    << endl;
                    }
                break;
            }
            case 6: {
                cout << "Reversing the list." << endl;
                reverseList(headInt, tailInt);
                break;
            }
            case 7:
                cout << "Linked List Contents: ";
                display(headInt);
                break;
            case 0:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 0);
    return 0;
}
```

OUTPUT

```
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 1
Enter the value to insert at the head: 12
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 1
Enter the value to insert at the head: 31
Inserted successfully.
Select an option:
```

```
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 2
Enter the value to insert at the tail: 32
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 3
Enter the position to insert at: 2
Enter the value to insert: 55
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 7
Linked List Contents: 31 55 12 32
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 4
Enter the position to delete from: 2
Memory freed with data 55
Deleted successfully.
Select an option:
1. Insert at Head
```

```
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 7
Linked List Contents: 31 12 32
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 5
Enter the value to search for: 3
Value 3 not found in the list.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 6
Reversing the list.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 7
Linked List Contents: 32 12 31
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
```

```
7. Display
0. Exit
Enter your choice: 0
Exiting program.
```

PROGRAM 4

Perform Stack operations using Array implementation. Use Templates.

ALGORITHM

```
1. Prompt user for stack size
2. Read userSize
3. If userSize is 0
        Display error message and exit
        Instantiate stack with specified size and data type
4. Repeat until user chooses to exit:
        Display menu options
        Read user choice
        Switch (user choice):
            Case 1: Push value onto the stack
            Case 2: Pop value from the stack
            // ... (other cases for stack operations)
          End Switch
5. End
```

PROGRAM CODE

```
#include <iostream>
using namespace std;
template <typename T, size_t SIZE>
class Stack {
    private:
        int top;
        T arr[SIZE];
    public:
        Stack() : top(-1) {
            for (size_t i = 0; i < SIZE; i++) {
                arr[i] = T();
            }
        }
        void push(const T& val) {
            if (top == static_cast<int>(SIZE - 1)) {
                cout << "Stack Overflow" << endl;
                return;
            }
            top++;
            arr[top] = val;
        }
        T pop() {
            if (top == -1) {
                cout << "Stack Underflow" << endl;
                return T();
            }
            T popValue = arr[top];
            arr[top] = T();
            top--;
            return popValue;
            }
            void display() const {
                cout << "All values in the Stack are: ";
                for (int i = top; i >= 0; i--) {
                    cout << arr[i] << " ";
                }
                cout << endl;
            }
    };
    int main() {
        size_t size;
        cout << "Enter the size of the stack: ";
        cin >> size;
        if (size == 0) {
            cout << "Stack size must be greater than zero. Exiting program." << endl;
            return 1;
        }
        Stack<int, 5> s1;
        int opt, value;
        do {
            cout << "What operation do you want to perform? Select Option Number,
            Enter 0 to Exit." << endl;
            cout << "1. Push" << endl;
            cout << "2. Pop" << endl;
            cout << "3. Display" << endl;
            cin >> opt;
            switch (opt) {
                case 0:
                    break;
                case 1:
                    cout << "Enter an item to push in the stack: ";
                    cin >> value;
                    s1.push(value);
                    break;
                case 2:
                    cout << "Pop Function called - Popped Value: " << s1.pop() <<
                    endl;
                    break;
                case 3:
                    s1.display();
                    break;
                default:
                    cout << "Enter a proper option number" << endl;
            }
        } while (opt != 0);
        return 0;
    }
```

OUTPUT

```
    Enter the size of the stack: 3
```

```
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
1
Enter an item to push in the stack: 12
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
1
Enter an item to push in the stack: 17
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
1
Enter an item to push in the stack: 56
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
2
Pop Function called - Popped Value: 56
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
3
All values in the Stack are: 17 12
What operation do you want to perform? Select Option Number, Enter 0 to Exit.
1. Push
2. Pop
3. Display
0
```

PROGRAM 5

Implement Circular Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

ALGORITHM

1. Node Class:
    - Define a templated Node class with a data field and a pointer to the next node.
    - Implement a constructor and a destructor for memory management.
2. Insert Node Function:
    - Implement a function insertNode to insert a new node after a specified element in a circular linked list.
    - If the list is empty, create a new node and make it the only node in the list.
    - Otherwise, traverse the list to find the specified element, and insert a new node after it.
3. Delete Node Function:
    - Implement a function deleteNode to delete a node with a specified value from a circular linked list.
    - If the list is empty, indicate that it's an empty list.
    - Otherwise, traverse the list to find the node with the specified value and delete it. Update the tail pointer if necessary.
4. Reverse List Function:
    - Implement a function reverseList to reverse the order of nodes in a circular linked list.
    - If the list is empty or has only one node, no change is needed.
    - Otherwise, use three pointers (prev, cur, nextNode) to reverse the next pointers of each node.
5. Search Node Function:
    - Implement a function searchNode to find the position of a node with a specified value in a circular linked list.
    - Traverse the list and return the position if the value is found; otherwise, return -1.
6. Print Function:
    - Implement a function print to print the elements of a circular linked list.
7. Main Function:
    - In the main function:
        - Create instances of circular linked lists for integers and strings (tailInt and tailString).
        - Use a menu-driven loop to allow the user to perform operations like insertion, deletion, reversal, searching, and printing on the linked list.
        - Handle user input and call the corresponding functions accordingly.

PROGRAM CODE

```
#include<iostream>
```

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
class Node {
    public:
        T data;
        Node *next;
        Node(T data) : data(data), next(NULL) {}
        ~Node() {
            T value = this->data;
            if (this->next != NULL) {
                delete next;
                this->next = NULL;
            }
        cout << "MEMORY FREE FOR NODE WITH DATA " << value << endl;
        }
};
template<typename T>
void insertNode(Node<T> *&tail, T element, T data) {
    if (tail == NULL) {
        Node<T> *newNode = new Node<T>(data);
        tail = newNode;
        newNode->next = newNode;
    } else {
        Node<T> *cur = tail;
        while (cur->next != tail) {
            cur = cur->next;
        }
        Node<T> *temp = new Node<T>(data);
        temp->next = tail;
        cur->next = temp;
    }
}
template<typename T>
void deleteNode(Node<T> *&tail, T value) {
    if (tail == NULL) {
        cout << "Empty List" << endl;
        return;
    } else {
        Node<T> *prev = tail;
        Node<T> *cur = tail->next;
        while (cur->data != value && cur != tail) {
            prev = cur;
            cur = cur->next;
        }
        if (cur->data == value) {
            prev->next = cur->next;
            if (cur == tail) {
                tail = (tail == tail->next) ? NULL : prev;
            }
            cur->next = NULL;
            delete cur;
        } else {
            cout << "Value " << value << " not found in the list." << endl;
        }
    }
}
template<typename T>
void reverseList(Node<T>*& tail) {
    if (tail == NULL || tail->next == tail) {
        return;
    }
    Node<T>* prev = NULL;
    Node<T>* cur = tail;
    Node<T>* nextNode;
    while (cur != NULL) {
        nextNode = cur->next;
        cur->next = prev;
        prev = cur;
        cur = nextNode;
        if (cur == tail) {
            break;
        }
    }
    tail = prev;
}
template<typename T>
int searchNode(Node<T> *tail, T target) {
    Node<T> *temp = tail;
    int pos = 1;
    do {
        if (temp->data == target) {
            return pos;
        }
        temp = temp->next;
        pos++;
    } while (temp != tail);
    return -1;
}
template<typename T>
void print(Node<T> *tail) {
    Node<T> *temp = tail;
    if (tail == NULL) {
        cout << "EMPTY LIST" << endl;
        return;
    }
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != tail);
```

```
        cout << endl;
}
int main() {
        Node<int> *tailInt = NULL;
        Node<string> *tailString = NULL;
        int choice;
        do {
                cout << "\nSelect an option:\n"
                 << "1. Insert Node\n"
                 << "2. Delete Node\n"
                 << "3. Reverse List\n"
                 << "4. Search Node\n"
                 << "5. Print List\n"
                 << "0. Exit\n"
                << "Enter your choice: ";
                cin >> choice;
                switch (choice) {
                        case 1: {
                                int element, data;
                                cout << "Enter the element after which to insert: ";
                                cin >> element;
                                cout << "Enter the data to insert: ";
                                cin >> data;
                                insertNode(tailInt, element, data);
                                break;
                        }
                        case 2: {
                                int value;
                                cout << "Enter the value to delete: ";
                                cin >> value;
                                deleteNode(tailInt, value);
                                break;
                        }
                        case 3:
                                reverseList(tailInt);
                                cout << "List reversed." << endl;
                                break;
                        case 4: {
                                int target;
                                cout << "Enter the value to search for: ";
                                cin >> target;
                                int position = searchNode(tailInt, target);
                                if (position != -1) {
                                        cout << "Value " << target << " found at position " <<
                                        position << endl;
                                } else {
                                        cout << "Value " << target << " not found in the list."
                                        << endl;
                                }
                                break;
```

```
            }
            case 5:
                cout << "Current List (int): ";
                print(tailInt);
                break;
            case 0:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 0);
    return 0;
}
```

OUTPUT

```
Select an option:
1. Insert Node
2. Delete Node
3. Reverse List
4. Search Node
5. Print List
0. Exit
Enter your choice: 1
Enter the element after which to insert: 1
Enter the data to insert: 2
Select an option:
1. Insert Node
2. Delete Node
3. Reverse List
4. Search Node
5. Print List
0. Exit
Enter your choice: 1
Enter the element after which to insert: 2
Enter the data to insert: 3
Select an option:
1. Insert Node
2. Delete Node
3. Reverse List
4. Search Node
5. Print List
0. Exit
Enter your choice: 5
Current List (int): 2 3
Select an option:
1. Insert Node
2. Delete Node
3. Reverse List
4. Search Node
```

```
5. Print List
0. Exit
Enter your choice: 0
Exiting program.
```

PROGRAM 6

Perform Stack operations using Linked List implementation.

ALGORITHM

```
1. Prompt user for stack size
2. Read userSize
3. If userSize is 0
        Display error message and exit
        Instantiate stack with specified size and data type
4. Repeat until user chooses to exit:
        Display menu options
        Read user choice
        Switch (user choice):
            Case 1: Push value onto the stack
            Case 2: Pop value from the stack
            // ... (other cases for stack operations)
        End Switch
5. End
```

PROGRAM CODE

```
#include<iostream>
using namespace std;
template<typename T>
class Node {
        public:
                T data;
                Node* next;
                Node(T data) : data(data), next(nullptr) {}
};
template<typename T>
class Stack {
        private:
                Node<T>* top;
                public:
                        Stack() : top(nullptr) {}
                        ~Stack() {
                                while (!isEmpty()) {
                                        pop();
                                }
                        }
                        bool isEmpty() {
                                return top == nullptr;
                        }
                        void push(T data) {
                                Node<T>* newNode = new Node<T>(data);
                                newNode->next = top;
                                top = newNode;
                                cout << "Pushed: " << data << endl;
                        }
```

```
            void pop() {
                if (isEmpty()) {
                    cout << "Stack underflow (empty)." << endl;
                    return;
                }
                Node<T>* temp = top;
                top = top->next;
                cout << "Popped: " << temp->data << endl;
                delete temp;
            }
            T peek() {
                if (isEmpty()) {
                    cerr << "Stack is empty." << endl;
                    exit(EXIT_FAILURE);
                }
                return top->data;
            }
            void print() {
                if (isEmpty()) {
                    cout << "Stack is empty." << endl;
                    return;
                }
                Node<T>* temp = top;
                while (temp != nullptr) {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
};
int main() {
    Stack<int> stack;
    int choice;
    do {
        cout << "\nSelect an option:\n"
         << "1. Push\n"
         << "2. Pop\n"
         << "3. Peek\n"
        << "4. Print Stack\n"
        << "0. Exit\n"
        << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1: {
                int data;
                cout << "Enter the element to push: ";
                cin >> data;
                stack.push(data);
                break;
            }
```

```
                        case 2:
                                stack.pop();
                                break;
                        case 3:
                                if (!stack.isEmpty()) {
                                        cout << "Top element: " << stack.peek() << endl;
                                }
                                break;
                        case 4:
                                cout << "Stack elements: ";
                                stack.print();
                                break;
                        case 0:
                                cout << "Exiting program.\n";
                                break;
                        default:
                                cout << "Invalid choice. Please try again.\n";
                }
        } while (choice != 0);
        return 0;
}
```

OUTPUT

```
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 1
Enter the element to push: 12
Pushed: 12
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 1
Enter the element to push: 15
Pushed: 15
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 1
Enter the element to push: 17
Pushed: 17
```

```
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 2
Popped: 17
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 3
Top element: 15
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 4
Stack elements: 15 12
Select an option:
1. Push
2. Pop
3. Peek
4. Print Stack
0. Exit
Enter your choice: 0
Exiting program.
Popped: 15
Popped: 12
```

PROGRAM 7

WAP to display Fibonacci series (i)using recursion, (ii) using iteration

ALGORITHM

(i) Using Recursion

```
Algorithm RecursiveFibonacci(n):
if n is 0 or 1
    return n
return RecursiveFibonacci(n-1) + RecursiveFibonacci(n-2)
```

(ii) Using Iteration

```
Algorithm IterativeFibonacci(n):
a <- 0
b <- 1
for i from 0 to n-1 do
    print a
    nextTerm <- a + b
    a <- b
b <- nextTerm
```

PROGRAM CODE

(i) Using Recursion

```
#include<iostream>
using namespace std;
int fibo(int n){
    if(n<=1) return n;
    else return (fibo(n-1)+fibo(n-2));
}
int main(){
    int i=0,n;
    cout<<"Enter the number of terms for Fibonacci series: ";
    cin>>n;
    cout<<"The Fibonacci Series: ";
    while(i<n){
        cout<<" "<<fibo(i);
        i++;
    }
    return 0;
}
```

(ii) Using Iteration

```
#include<iostream>
using namespace std;
long long fibo(int n) {
    if (n <= 1) {
        return n;
    }
    long long a = 0, b = 1;
    for (int i = 2; i <= n; ++i) {
        long long nextTerm = a + b;
        a = b;
```

```
                b = nextTerm;
            }
            return b;
        }
        int main() {
            int n;
            cout << "Enter the number of terms for Fibonacci series: ";
            cin >> n;
            cout << "The Fibonacci series: ";
            for (int i = 0; i < n; ++i) {
                cout << fibo(i) << " ";
            }
            return 0;
        }
```

OUTPUT

```
SET 1
Enter the number of terms for Fibonacci series: 5
The Fibonacci series:  0 1 1 2 3

SET 2
Enter the number of terms for Fibonacci series:12
The Fibonacci series:  0 1 1 2 3 5 8 13 21 34 55 89
```

PROGRAM 8

WAP to scan a polynomial using linked list and add two polynomial.

ALGORITHM

```
Algorithm PolynomialAddition():
    head1 <- create()  // Input coefficients and exponents for the first
                //polynomial
    head2 <- create()  // Input coefficients and exponents for the second
                //polynomial
    head3 <- NULL      // Initialize the result polynomial linked list
    ptr1 <- head1      // Pointer for the first polynomial
    ptr2 <- head2      // Pointer for the second polynomial
    while ptr1 is not NULL and ptr2 is not NULL do
        if ptr1.expo = ptr2.expo
            head3 <- insert(head3, ptr1.coeff + ptr2.coeff, ptr1.expo)
            ptr1 <- ptr1.link
            ptr2 <- ptr2.link
        else if ptr1.expo > ptr2.expo
            head3 <- insert(head3, ptr1.coeff, ptr1.expo)
            ptr1 <- ptr1.link
        else if ptr1.expo < ptr2.expo
            head3 <- insert(head3, ptr2.coeff, ptr2.expo)
            ptr2 <- ptr2.link
    while ptr1 is not NULL do
        head3 <- insert(head3, ptr1.coeff, ptr1.expo)
        ptr1 <- ptr1.link
    while ptr2 is not NULL do
        head3 <- insert(head3, ptr2.coeff, ptr2.expo)
        ptr2 <- ptr2.link
    print(head3)  // Print the result polynomial
```

PROGRAM CODE

```
#include <iostream>
using namespace std;
struct Node {
    float coeff;
    int expo;
    Node* link;
};
Node* insert(Node* head, float co, int ex) {
    Node* temp;
    Node* newP = new Node;
    newP->coeff = co;
    newP->expo = ex;
    newP->link = nullptr;
    if (head == nullptr || ex > head->expo) {
        newP->link = head;
        head = newP;
    } else {
        temp = head;
        while (temp->link != nullptr && temp->link->expo >= ex)
            temp = temp->link;
        newP->link = temp->link;
        temp->link = newP;
    }
    return head;
}
Node* create(Node* head) {
    int n, i;
    float coeff;
    int expo;
    cout << "Enter the number of terms: ";
    cin >> n;
    for (i = 0; i < n; i++) {
        cout << "Enter the coefficient for term " << i + 1 << ": ";
        cin >> coeff;
        cout << "Enter the exponent for term " << i + 1 << ": ";
        cin >> expo;
        head = insert(head, coeff, expo);
    }
    return head;
}
void print(Node* head) {
    if (head == nullptr)
        cout << "No Polynomial." << endl;
    else {
        Node* temp = head;
        while (temp != nullptr) {
            cout << "(" << temp->coeff << "x^" << temp->expo << ")";
            temp = temp->link;
            if (temp != nullptr)
                cout << " + ";
            else cout << endl;
        }
    }
}
void polyAdd(Node* head1, Node* head2) {
    Node* ptr1 = head1;
    Node* ptr2 = head2;
    Node* head3 = nullptr;
    while (ptr1 != nullptr && ptr2 != nullptr) {
        if (ptr1->expo == ptr2->expo) {
            head3 = insert(head3, ptr1->coeff + ptr2->coeff, ptr1->expo);
            ptr1 = ptr1->link;
            ptr2 = ptr2->link;
        } else if (ptr1->expo > ptr2->expo) {
            head3 = insert(head3, ptr1->coeff, ptr1->expo);
            ptr1 = ptr1->link;
        } else if (ptr1->expo < ptr2->expo) {
```

```
                        head3 = insert(head3, ptr2->coeff, ptr2->expo);
                        ptr2 = ptr2->link;
                }
        }
        while (ptr1 != nullptr) {
                head3 = insert(head3, ptr1->coeff, ptr1->expo);
                ptr1 = ptr1->link;
        }
        while (ptr2 != nullptr) {
                head3 = insert(head3, ptr2->coeff, ptr2->expo);
                ptr2 = ptr2->link;
        }
        cout << "Added polynomial is: ";
        print(head3);
}
int main() {
        Node* head1 = nullptr;
        Node* head2 = nullptr;
        cout << "Enter the First polynomial" << endl;
        head1 = create(head1);
        cout << "Enter the second polynomial" << endl;
        head2 = create(head2);
        polyAdd(head1, head2);
        return 0;
}
```

OUTPUT

```
SET 1:
Enter the First polynomial
Enter the number of terms: 3
Enter the coefficient for term 1: 2
Enter the exponent for term 1: 3
Enter the coefficient for term 2: 3
Enter the exponent for term 2: 2
Enter the coefficient for term 3: 4
Enter the exponent for term 3: 1
Enter the second polynomial
Enter the number of terms: 2
Enter the coefficient for term 1: 3
Enter the exponent for term 1: 4
Enter the coefficient for term 2: 2
Enter the exponent for term 2: 1
Added polynomial is: (3x^4) + (2x^3) + (3x^2) + (6x^1)

SET 2:
Enter the First polynomial
Enter the number of terms: 2
Enter the coefficient for term 1: 12
Enter the exponent for term 1: 2
Enter the coefficient for term 2: 3
Enter the exponent for term 2: 1
Enter the second polynomial
Enter the number of terms: 1
Enter the coefficient for term 1: 15
Enter the exponent for term 1: 3
Added polynomial is: (15x^3) + (12x^2) + (3x^1)
```