LAB 8

CSE225L

Sorted List (Linked List—Based)

In this lab, we will:

- Design and implement the List ADT, where the items are sorted.
- Implement the List ADT using a linked list—based structure.
- Create the **SortedType** class with methods for insertion, searching, deletion, and resetting the list.
- Test the functionality of the **SortedType** class by performing various operations.

SORTED LIST (LINKED LIST-BASED)

sortedtype.h

```h
#ifndef SORTEDTYPE_H
#define SORTEDTYPE_H

template <class T>
class SortedType
{
private:
    struct Node
    {
        T data;
        Node* next;
    };
    Node* head;
    Node* pointTo;
    int size;
public:
    SortedType();
    ~SortedType();
    int Length();
    void Insert(T value);
    void Search(T value, bool &found);
    void Delete(T value);
    void MakeEmpty();
    void GetNext(T &value);
    void Reset();
};
#endif // SORTEDTYPE_H
```

sortedtype.cpp

```cpp
#include "sortedtype.h"
#include <iostream>
using namespace std;

template <class T>
SortedType<T>::SortedType()
{
    head = NULL;
    pointTo = NULL;
    size = 0;
}

template <class T>
int SortedType<T>::Length()
{
    return size;
}
```

```
template <class T>
void SortedType<T>::Insert(T value)
{
    Node* temp = new Node;      // Create a new node
    temp->data = value;         // Set the data of the new node
    temp->next = NULL;          // Initialize the next pointer

    // Case 1: Empty list
    if (head == NULL)
    {
        head = temp;            // Insert the new node as the head
    }
    else
    {
        // Case 2: Insert at the beginning
        if (value < head->data)
        {
            temp->next = head;  // New node points to the old head
            head = temp;        // New node becomes the new head
        }
        else
        {
            // Case 3: Traverse the list to find the correct position
            Node* i = head;
            Node* prev = NULL;

            while (i != NULL && value > i->data)
            {
                prev = i;       // Move prev to 'i'
                i = i->next;    // Move to the next node
            }
            // Insert between prev and 'i'
            temp->next = i;     // New node points to 'i'
            prev->next = temp;  // Previous node points to new node
        }
    }
    size++;  // Increment the size of the list
}
```

```
template <class T>
void SortedType<T>::Search(T value, bool &found)
{
    found = false;

    Node* i = head;

    while(i != NULL)
    {
        if (value == i->data)
        {
            found  = true;
            break;
        }
        else
        {
            i = i->next;
        }
    }
}

template <class T>
void SortedType<T>::Delete(T value)
{
    Node* i = head;
    Node* prev = NULL;
    bool found = false;

    while(i != NULL)
    {
        if (value == i->data)
        {
            found = true;
            break;
        }
        else
        {
            prev = i;
            i = i->next;
        }
    }

    if (found)
    {
        if (prev == NULL) // first node / no previous nodes
            head = i->next;
        else
            prev->next = i->next;
        delete i;
        size--;
    }
    else
    {
        cout << "Error: Item could not be found in the list" << endl;
    }
}
```

```
template <class T>
void SortedType<T>::MakeEmpty()
{
    Node* i = head;
    Node* nextNode;

    while (i != NULL)
    {
        nextNode = i->next; // Store the next node
        delete i;           // Delete the current node
        i = nextNode;       // Move to the next node
    }

    head = NULL;
    size = 0;
}

template <class T>
SortedType<T>::~SortedType()
{
    MakeEmpty();
}

template <class T>
void SortedType<T>::GetNext(T &value)
{
    if (pointTo == NULL)
    {
        pointTo = head;
        value = pointTo->data;
    }
    else
    {
        value = pointTo->data;
    }
    pointTo = pointTo->next;
}

template <class T>
void SortedType<T>::Reset()
{
    pointTo = NULL;
}
```

SORTED LIST (LINKED LIST-BASED)

TASKS:

Instructions:

- Create the driver file (main.cpp) and perform the following tasks.
- You cannot make any changes to the header (**.h**) or source (**.cpp**) files of the **SortedType** class.

Operation	Input Values	Expected Output
Create a list of integers		
Insert four items	5 4 2 1	
Print the list		1 2 4 5
Insert one item	7	
Print the list		1 2 4 5 7
Search 6 and print whether found or not		Item is not found
Search 5 and print whether found or not		Item is found
Delete 1		
Print the list		2 4 5 7
Delete 4		
Print the list		2 5 7
Delete 16		Error: Item could not be found in the list

Tasks 1	Input Values
Given two sorted linked lists, find the union (∪) of the lists (all elements from both lists without duplicates) using the **SortedType** class.	T = {1,2,5,6,10,14} (First List) F = {3,2,4,6,9,16,19} (Second List)
	Expected Output
	T∪F = {1,2,3,4,5,6,9,10,14,16,19}

Tasks 2	Input Values
Given two sorted linked lists, find the intersection (∩) of the lists (all elements that appear in both lists) using the **SortedType** class.	T = {1,2,5,6,10,14} (First List) F = {3,2,4,6,9,16,19} (Second List)
	Expected Output
	T∩F = {2,6}