**Ý tưởng;**

* **Init:** Tạo một danh sách rỗng ban đầu
* **InsertFirst:** Thêm một phần tử mới vào đầu danh sách
* **InsertAfter:** Chèn phần tử mới sau một node đã có trong danh sách
* **InsertOrder:** Thêm phần tử sao cho danh sách luôn có thứ tự tăng dần.
* **ShowList:** Xuất toàn bộ danh sách ra màn hình.
* **ListSize:** Đếm số phần tử trong danh sách.
* **isEmpty:** Xác định xem danh sách có phần tử nào không.
* **isFull:** Xác định xem danh sách có đầy không.
* **Find:** Xác định xem phần tử có tồn tại không.
* **Retrieve:** Trả về node tại vị trí cụ thể.
* **DeleteFirst:** Loại bỏ phần tử đầu tiên.
* **DeleteAfter:** Xóa node đứng ngay sau một node đã có.
* **RemoveOrder:** Loại bỏ phần tử mà vẫn giữ thứ tự danh sách.
* **SelectionSort:** Sắp xếp danh sách theo thứ tự tăng dần.
* **ClearList:** Giải phóng toàn bộ bộ nhớ của danh sách.

**Input:**

1. Khởi tạo danh sách liên kết rỗng.
2. Thêm các phần tử vào danh sách theo thứ tự ( chèn các phần tử vào đầu danh sách ):
   * 10
   * 5
   * 15
   * 7
   * 12
   * 20 ( Thay thế một phần tử tại vị trí cụ thể )
3. Hiển thị danh sách sau khi thêm các phần tử.
4. Lấy kích thước danh sách.
5. Kiểm tra danh sách có rỗng không.
6. Kiểm tra danh sách có đầy không.
7. Tìm kiếm phần tử có giá trị: 10.
8. Lấy phần tử tại vị trí 2 trong danh sách.
9. Xóa phần tử có giá trị: 5 (phần tử không có trong danh sách nên không bị ảnh hưởng).
10. Hiển thị danh sách sau khi xóa phần tử.
11. Sắp xếp danh sách theo thứ tự tăng dần ( SelectionSort ).
12. Hiển thị danh sách sau khi sắp xếp.
13. Xóa toàn bộ danh sách.
14. Hiển thị danh sách sau khi xóa toàn bộ.

**Output:**

Danh sách sau khi chèn: 7 -> 15 -> 20 -> 10 -> 12 -> NULL

Kích thước danh sách: 5

Danh sách có rỗng không? Không

Danh sách có đầy không? Không

Tìm thấy phần tử: 10

Phần tử tại vị trí 2: 20

Danh sách sau khi xóa 5: 7 -> 15 -> 20 -> 10 -> 12 -> NULL

Danh sách sau khi sắp xếp: 7 -> 10 -> 12 -> 15 -> 20 -> NULL

Danh sách sau khi xóa toàn bộ: NULL

#include <iostream>

using namespace std;

// Định nghĩa cấu trúc Node

struct Node {

    int info;

    Node\* next;

};

// Khởi tạo danh sách rỗng

void Init(Node\*& pHead) {

    pHead = NULL;

}

// Kiểm tra danh sách có rỗng không

bool isEmpty(Node\* pHead) {

    return (pHead == NULL);

}

// Kiểm tra danh sách có đầy không (đối với danh sách liên kết, luôn có thể mở rộng nên trả về false)

bool isFull(Node\* pHead) {

    return false;

}

// Đếm kích thước danh sách

int ListSize(Node\* pHead) {

    int count = 0;

    Node\* p = pHead;

    while (p != NULL) {

        count++;

        p = p->next;

    }

    return count;

}

// Truy xuất phần tử tại vị trí pos

Node\* Retrieve(Node\* pHead, int pos) {

    if (pos < 0) return NULL;

    Node\* p = pHead;

    int index = 0;

    while (p != NULL && index < pos) {

        p = p->next;

        index++;

    }

    return p;

}

// Tạo một Node mới

Node\* CreateNode(int x) {

    Node\* p = new Node;

    p->info = x;

    p->next = NULL;

    return p;

}

// Thêm vào đầu danh sách

void InsertFirst(Node\*& pHead, int x) {

    Node\* p = CreateNode(x);

    p->next = pHead;

    pHead = p;

}

// Thêm phần tử sau một nút cụ thể

void InsertAfter(Node\* p, int x) {

    if (p != NULL) {

        Node\* q = CreateNode(x);

        q->next = p->next;

        p->next = q;

    }

}

// Thêm phần tử có thứ tự

void InsertOrder(Node\*& pHead, int x) {

    if (pHead == NULL || pHead->info >= x) {

        InsertFirst(pHead, x);

        return;

    }

    Node\* p = pHead;

    while (p->next != NULL && p->next->info < x) {

        p = p->next;

    }

    InsertAfter(p, x);

}

// Thay thế phần tử tại vị trí pos

void Replace(Node\* pHead, int pos, int x) {

    Node\* p = Retrieve(pHead, pos);

    if (p != NULL) {

        p->info = x;

    }

}

// Duyệt danh sách và hiển thị các phần tử

void ShowList(Node\* pHead) {

    Node\* p = pHead;

    while (p != NULL) {

        cout << p->info << " -> ";

        p = p->next;

    }

    cout << "NULL" << endl;

}

// Tìm kiếm một phần tử trong danh sách

Node\* Find(Node\* pHead, int x) {

    Node\* p = pHead;

    while (p != NULL) {

        if (p->info == x) return p;

        p = p->next;

    }

    return NULL;

}

// Xóa phần tử đầu danh sách

void DeleteFirst(Node\*& pHead) {

    if (isEmpty(pHead)) return;

    Node\* p = pHead;

    pHead = pHead->next;

    delete p;

}

// Xóa phần tử sau một nút cụ thể

void DeleteAfter(Node\* p) {

    if (p == NULL || p->next == NULL) return;

    Node\* q = p->next;

    p->next = q->next;

    delete q;

}

// Xóa phần tử theo giá trị có thứ tự

void RemoveOrder(Node\*& pHead, int x) {

    if (isEmpty(pHead)) return;

    if (pHead->info == x) {

        DeleteFirst(pHead);

        return;

    }

    Node\* p = pHead;

    while (p->next != NULL && p->next->info < x) {

        p = p->next;

    }

    if (p->next != NULL && p->next->info == x) {

        DeleteAfter(p);

    }

}

// Xóa toàn bộ danh sách

void ClearList(Node\*& pHead) {

    while (!isEmpty(pHead)) {

        DeleteFirst(pHead);

    }

}

// Sắp xếp danh sách bằng Selection Sort

void SelectionSort(Node\*& pHead) {

    for (Node\* p = pHead; p != NULL && p->next != NULL; p = p->next) {

        Node\* min = p;

        for (Node\* q = p->next; q != NULL; q = q->next) {

            if (q->info < min->info) {

                min = q;

            }

        }

        swap(p->info, min->info);

    }

}

int main() {

    Node\* myList;

    Init(myList);

    InsertFirst(myList, 10);

    InsertFirst(myList, 5);

    InsertFirst(myList, 15);

    InsertOrder(myList, 7);

    InsertAfter(Find(myList, 10), 12);

    Replace(myList, 2, 20);

    cout << "Danh sách sau khi chèn: ";

    ShowList(myList);

    cout << "Kích thước danh sách: " << ListSize(myList) << endl;

    cout << "Danh sách có rỗng không? " << (isEmpty(myList) ? "Có" : "Không") << endl;

    cout << "Danh sách có đầy không? " << (isFull(myList) ? "Có" : "Không") << endl;

    Node\* found = Find(myList, 10);

    if (found) cout << "Tìm thấy phần tử: " << found->info << endl;

    else cout << "Không tìm thấy phần tử!" << endl;

    cout << "Phần tử tại vị trí 2: " << Retrieve(myList, 2)->info << endl;

    RemoveOrder(myList, 5);

    cout << "Danh sách sau khi xóa 5: ";

    ShowList(myList);

    SelectionSort(myList);

    cout << "Danh sách sau khi sắp xếp: ";

    ShowList(myList);

    ClearList(myList);

    cout << "Danh sách sau khi xóa toàn bộ: ";

    ShowList(myList);

    return 0;

}