BÁO CÁO TRIỂN KHAI DANH SÁCH LIÊN KẾT ĐƠN

# 1. Mục đích

Danh sách liên kết đơn (Singly Linked List) là một cấu trúc dữ liệu động được sử dụng để lưu trữ và quản lý các phần tử theo cách không liên tục trong bộ nhớ. Mỗi phần tử (node) chứa dữ liệu và một con trỏ tới phần tử tiếp theo. Chương trình này được viết bằng C++ nhằm triển khai các tác vụ cơ bản của danh sách liên kết đơn, bao gồm quản lý danh sách, thêm/xóa phần tử, tìm kiếm, sắp xếp và duyệt dữ liệu.

struct Node

{

    int data;

    Node \*next;

    Node(int value) : data(value), next(nullptr) {}

};

class LinkedList {…}

# 2. Các tác vụ đã triển khai

## Khởi tạo danh sách rỗng (init)

Hàm: LinkedList()  
Mục đích: Tạo một danh sách liên kết đơn rỗng bằng cách khởi tạo con trỏ head trỏ tới nullptr.  
Kết quả: Danh sách ban đầu không chứa phần tử nào.

LinkedList() : head(nullptr) {}

## Kiểm tra danh sách rỗng (isEmpty)

Hàm: isEmpty()  
Mục đích: Kiểm tra xem danh sách có rỗng hay không bằng cách kiểm tra head == nullptr.  
Kết quả: Trả về true nếu danh sách rỗng, false nếu không.

bool isEmpty()

    {

        return head == nullptr;

    }

## Thêm một phần tử vào danh sách (insert)

Hàm: insert(int value)  
Mục đích: Thêm một phần tử mới với giá trị value vào cuối danh sách.  
Cách thực hiện: Tạo node mới, duyệt đến cuối danh sách (nếu không rỗng) và liên kết node mới vào.

void insert(int value)

    {

        Node \*newNode = new Node(value);

        if (isEmpty())

        {

            head = newNode;

            return;

        }

        Node \*current = head;

        while (current->next != nullptr)

        {

            current = current->next;

        }

        current->next = newNode;

    }

## Tìm kiếm một phần tử trong danh sách (search)

Hàm: search(int value)  
Mục đích: Tìm kiếm một phần tử có giá trị value trong danh sách.  
Cách thực hiện: Duyệt từng node từ head, so sánh giá trị và trả về true nếu tìm thấy, false nếu không.

bool search(int value)

    {

        Node \*current = head;

        while (current != nullptr)

        {

            if (current->data == value)

            {

                return true;

            }

            current = current->next;

        }

        return false;

    }

## Loại bỏ một phần tử của danh sách (remove)

Hàm: remove(int value)  
Mục đích: Xóa phần tử đầu tiên có giá trị value khỏi danh sách.  
Cách thực hiện:  
- Nếu node đầu chứa giá trị, xóa và cập nhật head.  
- Nếu không, duyệt danh sách để tìm và xóa node tương ứng, giải phóng bộ nhớ.

void remove(int value)

    {

        if (isEmpty())

        {

            std::cout << "List is empty, nothing to remove.\n";

            return;

        }

        if (head->data == value)

        {

            Node \*temp = head;

            head = head->next;

            delete temp;

            std::cout << "Removed " << value << " from the list.\n";

            return;

        }

        Node \*current = head;

        while (current->next != nullptr && current->next->data != value)

        {

            current = current->next;

        }

        if (current->next == nullptr)

        {

            std::cout << "Value " << value << " not found in the list.\n";

            return;

        }

        Node \*temp = current->next;

        current->next = temp->next;

        delete temp;

        std::cout << "Removed " << value << " from the list.\n";

    }

## Lấy nội dung của một phần tử (retrieve)

Hàm: retrieve(int value)  
Mục đích: Trả về giá trị của phần tử đầu tiên khớp với value.  
Cách thực hiện: Duyệt danh sách, nếu tìm thấy thì trả về giá trị, nếu không trả về -1 và thông báo lỗi.

int retrieve(int value)

    {

        Node \*current = head;

        while (current != nullptr)

        {

            if (current->data == value)

            {

                return current->data;

            }

            current = current->next;

        }

        std::cout << "Value " << value << " not found in the list.\n";

        return -1;

    }

## Duyệt danh sách (traverse)

Hàm: traverse()  
Mục đích: Hiển thị tất cả các phần tử trong danh sách theo thứ tự từ đầu đến cuối.  
Cách thực hiện: Duyệt từ head, in từng giá trị của node cho đến khi gặp nullptr.

void traverse()

    {

        if (isEmpty())

        {

            std::cout << "List is empty.\n";

            return;

        }

        Node \*current = head;

        std::cout << "List: ";

        while (current != nullptr)

        {

            std::cout << current->data << " -> ";

            current = current->next;

        }

        std::cout << "nullptr\n";

    }

## Hủy bỏ danh sách (clear)

Hàm: clear()  
Mục đích: Xóa toàn bộ danh sách và giải phóng bộ nhớ.  
Cách thực hiện: Duyệt từng node, xóa từng node và đặt lại head về nullptr.

void clear()

    {

        Node \*current = head;

        while (current != nullptr)

        {

            Node \*next = current->next;

            delete current;

            current = next;

        }

        head = nullptr;

        std::cout << "List has been cleared.\n";

    }

## Sắp xếp danh sách tăng dần (sort)

Hàm: sort()  
Mục đích: Sắp xếp các phần tử trong danh sách theo thứ tự tăng dần.  
Cách thực hiện: Sử dụng thuật toán Bubble Sort, hoán đổi giá trị của các node nếu chúng không theo thứ tự tăng dần.

void sort()

    {

        if (isEmpty() || head->next == nullptr)

        {

            return;

        }

        bool swapped;

        Node \*current;

        Node \*last = nullptr;

        do

        {

            swapped = false;

            current = head;

            while (current->next != last)

            {

                if (current->data > current->next->data)

                {

                    int temp = current->data;

                    current->data = current->next->data;

                    current->next->data = temp;

                    swapped = true;

                }

                current = current->next;

            }

            last = current;

        } while (swapped);

        std::cout << "List has been sorted in ascending order.\n";

    }

# 3. Cấu trúc chương trình

Node: Cấu trúc cơ bản chứa dữ liệu (data) và con trỏ tới node tiếp theo (next).  
LinkedList: Lớp quản lý danh sách với con trỏ head và các phương thức thực hiện các tác vụ trên.  
Main: Hàm chính để kiểm tra các chức năng đã triển khai với dữ liệu mẫu.

int main()

{

    LinkedList list;

    std::cout << "Is list empty? " << (list.isEmpty() ? "Yes" : "No") << "\n";

    list.insert(30);

    list.insert(10);

    list.insert(50);

    list.insert(20);

    std::cout << "After inserting 30, 10, 50, 20:\n";

    list.traverse();

    int searchValue = 50;

    std::cout << "Search for " << searchValue << ": "

              << (list.search(searchValue) ? "Found" : "Not found") << "\n";

    int retrievedValue = list.retrieve(10);

    std::cout << "Retrieved value for 10: " << retrievedValue << "\n";

    list.remove(50);

    std::cout << "After removing 50:\n";

    list.traverse();

    list.sort();

    std::cout << "After sorting:\n";

    list.traverse();

    list.clear();

    std::cout << "Is list empty after clear? " << (list.isEmpty() ? "Yes" : "No") << "\n";

    return 0;

}

# 4. Kết quả chạy chương trình

Chương trình thực hiện các bước:  
- Khởi tạo danh sách rỗng.  
- Thêm các giá trị (30, 10, 50, 20).  
- Kiểm tra tìm kiếm (tìm 50).  
- Lấy nội dung (lấy giá trị 10).  
- Xóa phần tử (xóa 50).  
- Sắp xếp danh sách (10, 20, 30).  
- Hủy danh sách và kiểm tra trạng thái rỗng.

Is list empty? Yes

After inserting 30, 10, 50, 20:

List: 30 -> 10 -> 50 -> 20 -> nullptr

Search for 50: Found

Retrieved value for 10: 10

Removed 50 from the list.

After removing 50:

List: 30 -> 10 -> 20 -> nullptr

List has been sorted in ascending order.

After sorting:

List: 10 -> 20 -> 30 -> nullptr

List has been cleared.

Is list empty after clear? Yes

List has been cleared.

# 5. Nhận xét

Ưu điểm:  
- Triển khai đầy đủ các tác vụ cơ bản của danh sách liên kết đơn.  
- Quản lý bộ nhớ hiệu quả với việc giải phóng node khi xóa hoặc hủy danh sách.  
- Dễ mở rộng để thêm các chức năng khác (ví dụ: thêm vào vị trí cụ thể).  
  
Hạn chế:  
- Thuật toán sắp xếp Bubble Sort có độ phức tạp O(n²), không hiệu quả với danh sách lớn.  
- Chỉ hỗ trợ dữ liệu kiểu int, có thể mở rộng sang các kiểu khác nếu cần.

# 6. Kết luận

Chương trình đã triển khai thành công danh sách liên kết đơn với các tác vụ cơ bản theo yêu cầu. Đây là một nền tảng tốt để học tập và phát triển thêm các cấu trúc dữ liệu phức tạp hơn như danh sách liên kết đôi hoặc danh sách vòng.