Bài tập về nhà buổi 2

- Họ và Tên: Nguyễn Trường Giang
- MSSV: 24520011

Báo cáo bài tập thực hành môn Cấu trúc dữ liệu & giải thuật

BT 04: So sánh các ưu khuyết điểm của các kiểu Linked List

a. Linked List có và không có con trỏ Tail luôn lưu địa chỉ Node cuối

- **Có con trỏ Tail**:
- **Ưu điểm**:
  - Thêm phần tử vào cuối danh sách ngay lập tức trong $O(1)$ mà không cần phải duyệt toàn bộ danh sách trong $O(n)$.
  - Giảm nhiều thời gian khi thao tác với nửa phần tử cuối danh sách.
- **Nhược điểm**:
  - Tốn thêm bộ nhớ cho con trỏ Tail.
  - Tăng sự phức tạp khi thao tác xóa/thêm phần tử ở vị trí cuối danh sách.
- **Không có con trỏ Tail**:
- **Ưu điểm**:
  - Tiết kiệm bộ nhớ vì không cần thêm con trỏ Tail.
  - Dễ dàng quản lý code khi không có con trò Tail.
- **Nhược điểm**:
  - Thao tác với phần tử ở nửa danh sách cuối nhanh hơn.

b. Linked List có và không có Header Node

- **Có Header Node**:
- **Ưu điểm**:
  - Đơn giản hóa thao tác với danh sách, vì luôn có một node đại diện cho danh sách mà không phải xử lý trường hợp danh sách rỗng riêng biệt.

    - Việc thêm phần tử vào đầu danh sách trở nên dễ dàng hơn.
- **Nhược điểm:**
    - Tốn thêm bộ nhớ cho Header Node.
- **Không có Header Node:**
- **Ưu điểm:**
    - Tiết kiệm bộ nhớ vì không cần lưu trữ Header Node.
    - Đơn giản hơn nếu không có nhiều thao tác thêm vào đầu danh sách.
- **Nhược điểm:**
    - Phải xử lý riêng biệt khi danh sách rỗng hoặc khi thao tác với phần tử đầu tiên.
    - Các thao tác có thể phức tạp hơn đôi chút.

Dưới đây là một số hàm C++ để giải quyết các bài tập 5 và 6 theo yêu cầu:

BT 05: Viết hàm chuyển danh sách sinh viên vào các kiểu Linked List khác nhau

a. Chuyển từ mảng động sang danh sách liên kết đơn không có con trỏ cuối và ngược lại

```
Node* arrayToLinkedList(const vector<Student> &arr) {
    Node* head = nullptr;
    for (int i = arr.size() - 1; i >= 0; i--) {
        Node *ptr =  new Node(arr[i], head);
        head = ptr;
    }
    return head;
}

vector<Student> linkedListToArray(Node* head) {
    vector<Student> arr;
    while (head != nullptr) {
        arr.push_back(head->data);
        head = head->next;
    }
    return arr;
}
```

b. Chuyển từ mảng động sang danh sách liên kết đơn có con trỏ cuối và ngược lại

```
LinkedList arrayToLinkedListWithTail(const vector<Student> &arr) {
    Node* head = nullptr;
    Node* tail = nullptr;

    for (Student student : arr) {
        Node* ptr = new Node(student);
        if (head == nullptr) {
            head = ptr;
            tail = ptr;
        }
        else {
            tail->next = ptr;
            tail = ptr;
        }
    }
    return LinkedList(head, tail);
}

vector<Student> linkedListToArrayWithTail(LinkedList &list) {
    vector<Student> arr;
    Node* ptr = list.head;
    while (ptr != nullptr) {
        arr.push_back(ptr->data);
        ptr = ptr->next;
    }
    return arr;
}
```

c. Chuyển từ mảng động sang danh sách liên kết kép và ngược lại

```
struct DNode {
    Student data;
    DNode* prev;
    DNode* next;
};

DNode* arrayToDoublyLinkedList(Student* arr, int n) {
    DNode* head = nullptr;
    DNode* tail = nullptr;

    for (int i = 0; i < n; i++) {
        DNode* newNode = new DNode{arr[i], tail, nullptr};
        if (!head)
            head = newNode;
        if (tail)
            tail->next = newNode;
        tail = newNode;
    }
    return head;
```

```
}

Student* doublyLinkedListToArray(DNode* head, int& n) {
    n = 0;
    DNode* temp = head;
    while (temp) {
        n++;
        temp = temp->next;
    }

    Student* arr = new Student[n];
    temp = head;
    for (int i = 0; i < n; i++) {
        arr[i] = temp->data;
        temp = temp->next;
    }
    return arr;
}
```

d. Chuyển từ mảng động sang danh sách liên kết vòng và ngược lại

```
Node* arrayToCircularList(Student* arr, int n) {
    Node* head = nullptr;
    Node* tail = nullptr;

    for (int i = 0; i < n; i++) {
        Node* newNode = new Node{arr[i], nullptr};
        if (!head)
            head = tail = newNode;
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    if (tail) tail->next = head;
    return head;
}

Student* circularListToArray(Node* head, int& n) {
    if (!head) {
        n = 0;
        return nullptr;
    }
    n = 0;
    Node* temp = head;
    do {
        n++;
        temp = temp->next;
    } while (temp != head);

    Student* arr = new Student[n];
    temp = head;
```

```
    for (int i = 0; i < n; i++) {
        arr[i] = temp->data;
        temp = temp->next;
    }
    return arr;
}
```

e. Chuyển từ mảng động sang danh sách liên kết kép vòng và ngược lại

```
DNode* arrayToCircularDoublyList(Student* arr, int n) {
    DNode* head = nullptr;
    DNode* tail = nullptr;

    for (int i = 0; i < n; i++) {
        DNode* newNode = new DNode{arr[i], tail, nullptr};
        if (!head)
            head = newNode;
        if (tail)
            tail->next = newNode;
        tail = newNode;
    }

    if (head && tail) {
        head->prev = tail;
        tail->next = head;
    }

    return head;
}

Student* circularDoublyListToArray(DNode* head, int& n) {
    if (!head) {
        n = 0;
        return nullptr;
    }

    n = 0;
    DNode* temp = head;
    do {
        n++;
        temp = temp->next;
    } while (temp != head);

    Student* arr = new Student[n];
    temp = head;
    for (int i = 0; i < n; i++) {
        arr[i] = temp->data;
        temp = temp->next;
    }
    return arr;
}
```

BT 06: Viết hàm thêm phần tử vào

a. Thêm phần tử vào mảng động có thứ tự

```
void insertOrderedArray(Student*& arr, int& n, Student x) {
    // Tạo mảng mới với kích thước lớn hơn
    Student* newArr = new Student[n + 1];
    int i = 0, inserted = 0;

    for (; i < n; i++) {
        if (!inserted && x.id < arr[i].id) {
            newArr[i] = x;
            inserted = 1;
        }
        newArr[i + inserted] = arr[i];
    }

    if (!inserted) newArr[n] = x;

    delete[] arr;
    arr = newArr;
    n++;
```

b. Thêm phần tử vào xâu đơn có thứ tự (có và không có con trỏ cuối)

- Không có Tail

```
void insertOrderedList(Node*& head, Student x) {
    Node* newNode = new Node{x, nullptr};
    if (!head || x.id < head->data.id) {
        newNode->next = head;
        head = newNode;
        return;
    }

    Node* p = head;
    while (p->next && p->next->data.id < x.id)
        p = p->next;

    newNode->next = p->next;
    p->next = newNode;
}
```

- Có Tail

```
void insertOrderedListWithTail(Node*& head, Node*& tail, Student x) {
    Node* newNode = new Node{x, nullptr};
    if (!head || x.id < head->data.id) {
```

```
        newNode->next = head;
        head = newNode;
        if (!tail) tail = newNode;
        return;
    }

    Node* p = head;
    while (p->next && p->next->data.id < x.id)
        p = p->next;

    newNode->next = p->next;
    p->next = newNode;

    if (!newNode->next) tail = newNode;
}
```

c. Thêm phần tử vào xâu đơn có thứ tự + có Header Node + có và không có con trỏ cuối

- Không có Tail

```
void insertOrderedHeader(Node* header, Student x) {
    Node* newNode = new Node{x, nullptr};
    Node* p = header;

    while (p->next && p->next->data.id < x.id)
        p = p->next;

    newNode->next = p->next;
    p->next = newNode;
```

- Có tail

```
void insertOrderedHeaderWithTail(Node* header, Node*& tail, Student x) {
    Node* newNode = new Node{x, nullptr};
    Node* p = header;

    while (p->next && p->next->data.id < x.id)
        p = p->next;

    newNode->next = p->next;
    p->next = newNode;

    if (!newNode->next)
        tail = newNode;
}
```

d. Xâu đơn có thứ tự + có Header node và luôn có địa chỉ node cuối nằm trong các byte đầu của trường dữ liệu trong Header node.

```
struct HeaderNode {
    Node* tailPtr;
    Node* next;
};

void insertWithHeaderTailInData(HeaderNode* header, Student x) {
    Node* newNode = new Node{x, nullptr};
    Node* p = (Node*)header;

    while (p->next && p->next->data.id < x.id)
        p = p->next;

    newNode->next = p->next;
    p->next = newNode;

    // Cập nhật tail nếu chèn cuối
    if (!newNode->next)
        header->tailPtr = newNode;
}
```