**2024 Spring OOP Assignment Report**

과제 번호 : 2

학번 : 20230024

이름 : 문요준

Povis ID : yojun313

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.

독창적인 아이디어와 추가 기능은 보너스 점수를 받을 수 있으므로, 보고서에 명확히 기재하십시오.

문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

**문제 3번에 대한 부분 명시하고 작성 (ex. 문제3> )**

각 문항별 설명은 편의를 위한 것으로, 삭제하고 제출한다.

1. **프로그램 개요**
   * 이 프로그램은 정렬된 집합을 구현하고 관리하는 기능을 제공한다. 정렬된 집합은 중복 없이 숫자를 오름차순으로 저장하는 데이터 구조로, 각 숫자는 노드(Node)로 표현되며, 노드들은 단일 연결 리스트(OrderedSet)를 통해 연결된다.
   * 프로그램은 함수 호출을 통해 작동한다. 사용자는 OrderedSet 구조체를 정의하고, 이를 관리하기 위한 여러 함수(예: add, remove, size, contains, getValue)를 호출하여 집합에 원소를 추가, 삭제하거나 집합의 크기를 조회하고 특정 원소의 존재 여부를 확인할 수 있다.
   * 헤더파일은 Node 구조체와 OrderedSet 구조체, 그리고 집합을 관리하기 위한 함수들을 정의한다. Node 구조체는 각 원소의 값을 저장하고 다음 노드를 가리키는 포인터를 포함한다. OrderedSet 구조체는 집합의 크기와 첫 번째 노드(헤드)를 가리키는 포인터를 포함한다.
   * add(OrderedSet\* ordered, int v): 이 함수는 주어진 정수 v를 집합에 추가한다. 집합에 이미 동일한 값이 없는 경우에만 새 노드를 생성하고 적절한 위치에 삽입하여 숫자들이 오름차순으로 정렬되도록 한다 / add(OrderedSet\* ordered, const int\* arr, int size): 이 함수는 정수 배열 arr의 각 원소를 집합에 추가한다. 배열의 각 원소에 대해 add(OrderedSet\* ordered, int v) 함수를 호출한다 / remove(OrderedSet\* ordered, int index): 지정된 인덱스의 노드를 집합에서 삭제한다. 인덱스는 0부터 시작하며, 해당 노드를 리스트에서 제거한다 / size(OrderedSet\* ordered): 집합의 크기(노드의 개수)를 반환한다 / contains(OrderedSet\* ordered, int v): 집합이 특정 값을 포함하는지 여부를 확인한다. 값이 존재하면 true를, 그렇지 않으면 false를 반환한다 / getValue(OrderedSet\* ordered, int idx): 지정된 인덱스에 위치한 노드의 값을 반환한다. 인덱스는 유효한 범위 내에 있어야 한다.
2. **프로그램의 구조 및 알고리즘**

* **1. 단일 값 추가 (void add(OrderedList\* ordered, int v))**
* **새 노드 생성:** 입력된 값을 가지는 새로운 노드(newNode)를 생성한다.
* **리스트가 비었는지 확인:** 리스트의 크기(m\_size)가 0이면, 새 노드를 리스트의 헤드로 설정하고, 리스트의 크기를 1 증가시킨다.
* **리스트에 노드가 있는 경우:**
* **v** 값이 이미 존재하는지 확인한다. 이미 존재한다면 함수를 종료한다.
* **헤드보다 새 노드가 작거나 같은 경우:** 새 노드를 리스트의 맨 앞에 삽입하고, m\_size를 1 증가시킨다.
* **새 노드를 적절한 위치에 삽입:** 새 노드가 리스트 중간 또는 끝에 삽입되어야 하는 경우, 적절한 위치를 찾아 새 노드를 삽입한다. 이 과정에서 두 가지 경우를 고려한다:
* **새 노드가 현재 노드보다 크고 다음 노드보다 작거나 같은 경우:** 새 노드를 현재 노드와 다음 노드 사이에 삽입한다.
* **리스트의 끝까지 도달했을 때:** 새 노드가 모든 노드보다 크면, 새 노드를 리스트의 마지막에 삽입한다.
* **2. 배열을 통한 값 추가 (void add(OrderedList\* ordered, const int\* arr, int size))**
* **반복 처리:** 주어진 배열 arr의 각 원소에 대해 add(OrderedList\* ordered, int v) 함수를 반복적으로 호출하여, 배열의 모든 값을 차례대로 리스트에 추가한다.
* **3. 노드 제거 (void remove(OrderedList\* ordered, int index))**
* **탐색:** 삭제하려는 노드의 위치(index)까지 현재 노드(current)를 이동시킨다.
* **노드 삭제:** 현재 노드의 next 포인터를 삭제하려는 노드의 다음 노드로 변경하여, 리스트에서 해당 노드를 건너뛰도록 한다. 그 후, m\_size를 1 감소시킨다.
* **4. 리스트 크기 반환 (int size(OrderedList\* ordered))**
* **리스트 크기 반환:** 리스트의 현재 크기(m\_size)를 반환한다.
* **5. 값의 존재 여부 확인 (bool contains(OrderedList\* ordered, int v))**
* **순차 탐색:** 리스트의 시작부터 끝까지 모든 노드를 순차적으로 탐색한다.
* **값 비교:** 각 노드의 값과 입력된 값 v를 비교한다.
* **결과 반환:** 입력된 값 v를 가진 노드를 찾으면 true를, 찾지 못하면 false를 반환한다.
* **6. 인덱스에 위치한 값 조회 (int getValue(OrderedList\* ordered, int idx))**
* **인덱스 유효성 검사:** 주어진 인덱스 idx가 유효한 범위 내에 있는지 확인한다. 유효하지 않으면, 정의된 에러 값을 반환한다.
* **탐색:** 유효한 인덱스인 경우, 해당 인덱스 위치까지 노드(current)를 이동시킨다.
* **값 반환:** 인덱스에 해당하는 노드의 값을 반환한다.
* **[변수 설명]**
* (Node 구조체: 각 노드의 데이터(value)와 다음 노드를 가리키는 포인터(next)를 포함한다), (OrderedList 구조체: 연결 리스트의 헤드 노드(head)와 리스트의 전체 크기(m\_size)를 포함한다) (ordered: OrderedList 타입의 포인터로, 함수들이 리스트를 조작할 때 사용하는 주요 매개변수다), (v: 추가하거나 탐색할 값의 변수다), (arr: 한 번에 여러 값을 리스트에 추가하기 위한 배열이다), (size: 배열의 크기나, 리스트의 크기를 나타내는 변수다), (index: 제거하거나 조회할 노드의 위치를 나타내는 인덱스다), (newNode: add 함수 내에서 선언된 이 변수는 새로운 Node 타입의 인스턴스를 참조한다. 사용자가 추가하려는 새로운 데이터 값을 저장하기 위해 생성되며, 이후 적절한 위치에 연결 리스트에 삽입된다), (current: 여러 함수(add, remove, contains, getValue) 내에서 사용되는 이 포인터 변수는 연결 리스트를 순회하며 현재 노드를 가리키는 데 사용된다. 주로 새 노드를 삽입하거나, 특정 조건을 만족하는 노드를 찾거나, 특정 인덱스의 노드에 접근하는 등의 작업을 수행할 때 현재 노드의 위치를 추적한다)

1. **토론 및 개선**
   * + 이 코드를 작성하면서, 중복을 허용하지 않는 정렬된 집합을 관리하기 위한 다양한 함수를 구현하는 방법을 배웠다. 그리고 정렬된 집합을 유지하면서 데이터를 추가하고 삭제하는 방법에 대해 배웠다. 특히, 적절한 위치 탐색과 노드의 연결 상태를 관리하는 방법이 중요하다는 것을 깨달았다. 또한 포인터와 동적 메모리 할당을 사용하는 방법에 대해 더 잘 이해하게 되었다.
2. **참고 문헌**