STL 과제 보고서 - 2021184018

한국공학대학교 게임공학과 3학년 STL 과제 보고서

1. 파일 읽기 및 컨테이너 저장 방식

std::ifstream을 통해 바이너리 파일 "2025 STL 과제 파일 - 2021184018"을 열고, Player::read() 멤버 함수를 이용하여 std::array<Player, 2'500'000>에 모든 데이터를 한 번에 읽었다.

바이너리 파일 구조에 맞게 Player 객체의 기본 정보(name, score, id, num)를 먼저 읽고

그 이후에 unique\_ptr<char[]> p로 관리되는 메모리 블록을 num 바이트만큼 추가로 읽는다.

생성자에서 p를 release()하는 것과 reset()하는 것 중에서 고민했는데,

release는 코드가 실행될수록 메모리 누수가 생기는 문제가 있어서 p.reset()으로 메모리를 직접 해제하였다.

반복문을 통해 250만 개 객체에 대해 각각의 멤버함수read()를 호출하여 저장하였다.

마지막 Player 객체는 Show() 함수를 통해 다음과 같이 출력되었다.



2. 최고 점수와 평균 점수 계산

모든 Player의 점수를 비교하는 람다 함수를 사용하여 max\_element를 사용해 최댓 값을 구했다.

처음에는 scoreMap에 저장하는 방식을 사용하였으나

4번 문항에서 원본 배열을 한번 정렬할 때, 복사하지 않고 참조를 한 scoreMap의 내부 값이 꼬여서 다시 만들어야 하는 문제가 있었다.

이 경우 O(nlogn)만큼 걸리는 std::map의 생성을 한번 더 하는 건 비효율적이라고 생각했기 때문에, max\_element로 구하는 것이 낫다고 판단했다.

평균 점수는 std::accumulate를 사용하여 operator+= 연산자 오버로딩으로 모든 Player의 score를 더해 계산하였다.

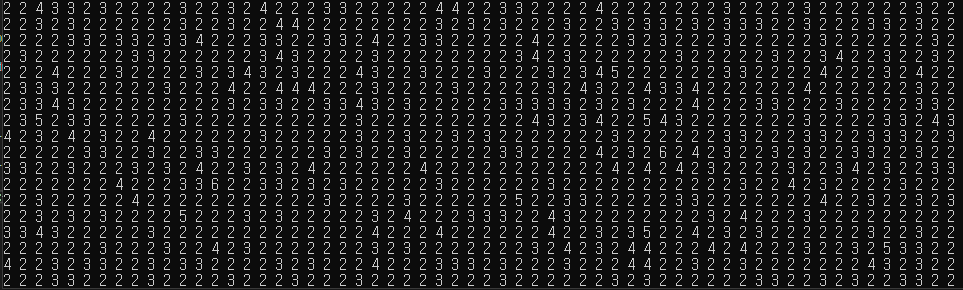


3. 같은 id를 가진 Player 탐색 및 저장

모든 Player를 id 기준으로 정렬한 후, 인접한 두 객체의 id가 같은 경우를 탐색하여 "같은아이디.txt"에 출력했다. 중복되는 id의 개수는 2개부터 전부 화면에 한 칸 간격으로 계속 출력되게 하였다.

중복되는 id의 개수들을 모두 합한다는 생각도 해보았으나

각 id 별로 얼마나 있는지를 출력하는 것이 문제와 더 부합한다고 생각하였다.



4. p가 가리키는 문자열 정렬 및 'a' 개수 기준 필터링

Player::getBuffer()를 이용해 unique\_ptr<char[]>가 가리키는 문자열을 std::string\_view로 받아 정렬에 사용하였다. 이후 std::count를 통해 'a'가 10개 이상 포함된 Player 객체의 수를 집계하였다.

초기에는 getBuffer의 반환 타입을 std::pair로 반환해서 문제를 해결하였는데,

string\_view로 반환하였을 때 수정은 불가능한 읽기 전용 이면서 pair에 비해 코드도 더 간결해졌고, 그래서 string\_view로 반환하였다.

출력 결과는 다음과 같다.



5. id 기반 검색 및 3가지 기준 탐색

이 문제에서는 2가지 std::map을 만들었다.

1. id를 저장하는 idmap
2. score를 저장하는 scoreMap

map을 사용하기로 결정한 이유는 다음과 같았다.

1. 생성의 결과가 동일한 값들의 본래 순서가 보장되는 stable한 정렬이 된다는 점
2. 탐색의 시간 복잡도가 O(logn)으로 array, vector에 비해 빠르다는 점
3. map의 생성 속도가 O(nlogn)으로, array, vector등의 정렬과 같은 시간 복잡도를 가지고 있다는 점
4. 중복되는 id들을 하나로 묶어서 관리할 수 있다는 점

초기에는 unordered\_map을 사용하고자 했지만,

1. 탐색 성능이O(1)로서 매우 빠른 성능을 보장하지만 정렬되지 않았기 때문에 “정렬된 기준”으로 자신의 앞, 뒤 값을 찾아내는 과정이 번거롭고, 그 방법이 알맞을 것이라는 확신이 없었다는 점
2. 탐색하는 코드가 복잡했다는 점

이 두가지 사유로 인해 사용하지 않기로 했고,

같은 id를 가진 Player들을 하나로 묶어서 관리하고자 했기에

multimap이 아닌 일반 map을 사용하였다

모든 map은 Player를 담을 수 있는 vector로 이루어진 value를 가지게 하였다

* 중복되는 id들을 하나로 묶어 관리하기 위해 vector를 사용하였다
* 내부 value들은 reference\_wrapper를 사용해서 레퍼런스처럼 사용하였다.

idmap의 find 함수를 사용하여 id가 존재하는지 검사하였다.

일반 std::find를 사용하려는 생각도 했지만

find를map에 사용하면 key가 아니라 pair로 된 쌍 전체를 비교한다

나는 key값만 비교할 것이기 때문에,

map이 가지고 있는 find 함수를 사용하였다.

초기에는 map의 contains라는 함수를 사용하였으나

find로 찾아낸 값을 받아서 사용하므로 find로 받은 뒤 end()가 아닌지를 검사하는 것이 낫다고 생각해 바꾸었다,

또한, id를 찾을 때는 최초로 찾아낸 id만을 기준으로 했다.

일치하는 id가 있을 때의 출력 방식은 다음과 같다

1. id 기준 앞 뒤 원소 출력
2. name 기준 앞 뒤 원소 출력
3. Score 기준 앞 뒤 출력

<id 기준 앞 뒤 원소 출력>

id기준 앞, 뒤 원소를 출력할 때,

id의 바로 앞은 바로 앞 id값의 맨 뒤 원소이고,

바로 뒤는 그 id 값의 vector바로 다음 원소

혹은 뒤 id 값의 맨 앞 원소이다

동일한 id는 전부 출력하기 때문에

앞, 뒤로 찾은 값이 속한 vector의 모든 원소를 출력한다

|  |  |  |  |
| --- | --- | --- | --- |
| prev |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| id |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| next |  |  |  |

<name 기준 앞 뒤 원소 출력>

2

key

value

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
|  |  | name |  |  |  |

reference wrapper

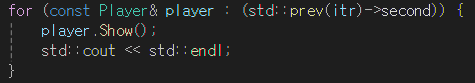
* name은 원본 Player array를 name기준 오름차순으로 정렬했다

name은 array를 재정렬해 사용한 이유는 다음과 같다

1. index를 구할 수 있으면 접근이 O(1)이 보장되어서 map보다 빠르다는 점
2. 앞 뒤 원소에 대한 접근 역시 O(1)이 보장된다는 점
3. name은 바로 앞 뒤 하나씩 만 출력하면 되기 때문에 추가적인 비용을 지불하며 map으로 묶을 필요가 없다는 점

* 기존 map의 value는 레퍼런스처럼 사용하고 있고, 이를 통해 본래 값을 불러올 수 있다
* 이를 통해 array의 index를 도출해서 random access로 접근, 자신과 앞 뒤 원소를 출력

<reference\_wrapper를 사용한 이유>



* range for문을 통한 각 player의 접근이 쉽고
* 메모리를 비교적 적게 사용할 수 있다

<Score 기준 앞 뒤 원소 출력>

score도 map의 형태로, id와 같이 앞, 뒤 원소를 특정한다

그러나 score의 경우에는 앞, 뒤 하나씩 만 출력하면 되니까

앞 vector.back()과, 뒤 vector.front() 혹은 현재 vector의 다음 원소만 출력

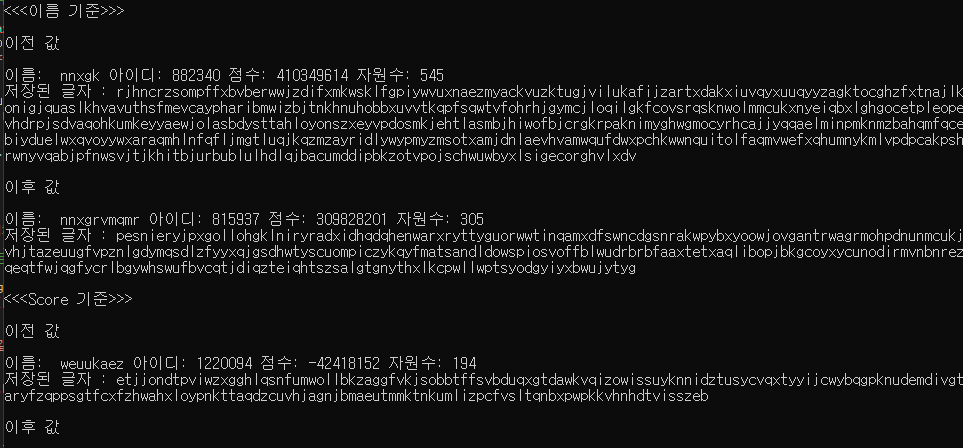
|  |  |  |  |
| --- | --- | --- | --- |
| prev |  |  |  |

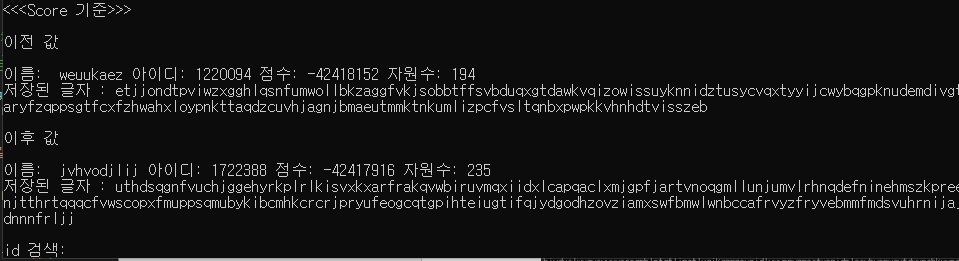
|  |  |  |  |
| --- | --- | --- | --- |
| score |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| next |  |  |  |

실행 결과의 일부는 다음과 같다









[효율적이라고 생각하는 내용]

1. 파일을 array 형태로 읽어온 것

* 250만개의 데이터만 읽고, 추가와 삭제가 없는 상황에서 array는 매우 효율적

1. 5번에서 2개의 map을 사용해 탐색한 것

* O(logn) 시간복잡도로 더 빠르게 탐색이 가능했다고 생각한다
* 동일한 id의 value를 묶어서 관리하므로 구분이 쉽다

1. map의 생성에서 원래 값을 복사하지 않은 것

* 데이터의 수가 250만개로 많기 때문에 복사보다는 참조나 포인터 등을 사용하는 게 더 효율적이었다고 생각했다

[개선점]

1. 5번 중간에서 index를 계산하는 것보다 안정적인 방법이 있을거라 생각한다



1. reference\_wrapper는 한 원소를 가져오는 코드가 복잡해서 다른 방법이 있을 거라고 생각했다
2. 코드가 조금 쓸 데 없이 비대한 느낌이다, 더 간소화할 수 있을 거라 생각한다.
3. 2번, 3번 문제에 map을 사용하지 않고 다른 방식을 사용했는데, 특히나 3번 문제의 코드가 난잡해서 복사해서 받는 map이나 다른 방식으로 더 간단한 구현 방법이 있을 것이라 생각했다.

[과제를 하면서 느낀 점]

단순히 문제를 푸는 것이 아니라, 더 좋은 방법을 생각해보며 스스로 알아보고, 적용해보면서 수업에서 배웠던 내용들을 떠올려볼 수 있어서 좋았다.

특히나 수업시간에 나오지 않았던 내용들을 많이 사용했는데,

내가 제대로 사용하고 있을지가 확실치 않아 계속 찾아보고 활용해보면서 교수님의 의견을 묻고 싶었지만 과제라 물어보지 못하는 점이 아쉬웠다.

특히나 std::map과 std::prev, std::nex, reference\_wrapper는 사용하면서도 사실 불안했다

array, vector, list, unique\_ptr처럼 내부적인 구조와 작동 방식을 배운 부분이 아니라,

순전히 내 힘으로 찾아서 알아보고, 그것이 맞는지도 스스로가 판단해야 했기 때문이었다.

난 예전부터 무엇인가를 스스로 판단하는 것에 확신이 없었고, 그래서 지금 보고서를 작성하는 이 순간까지도 불안감이 남아있다.

하지만 내가 판단해보고, 스스로 알아보면서 배웠던 내용들과 스스로 비교해보면서 작성한 방식이었기 때문에 단순히 배운 내용을 그대로 사용하던 것보다 즐거웠던 것 같다.