**2025 STL 과제 보고서**

2023180007 김도윤

**[문제 1]**

**고민한 점**

player 클래스 내부에서는 unique\_ptr<char[]>를 사용해 동적 메모리를 관리하기 때문에, 단순히 read() 함수만으로는 데이터를 모두 읽어올 수 없었다.

**해결 방법**

player::read(std::istream&) 함수를 정의하여, player 구조체 본체를 읽고, num 만큼 p가 가리키는 메모리를 새로 할당하고 그 메모리에 문자열 데이터를 추가로 읽었다.

컨테이너 선택 : std::array<Player,250’0000>를 사용하여 고정 크기 배열로 구성하였다. 문제에서 파일에는 2’500’000(250만) 개의 Player 객체가 기록되어 있다는 점을 고려해, 크기가 고정된 배열을 사용했다.

읽은 방식 : 범위 기반 for문을 사용해서 모든 player 객체에 접근하였고, 각 player 객체의 read()멤버 함수를 호출해 데이터를 읽었다.

**출력 결과**

제일 마지막 Player의 정보

이름: qor, 아이디: 363063, 점수: 198246315, 자원수: 527

저장된 글자: jcnlkecleukjqkbgfufhxzuedavwoobtilhfzejwbgsfprtvknfskqdlttriehvgzxrvdiqvcvaargidtmvejxzvcpebluvblouolatvxxodotzjnmhlugpebntwjzpmccqxjvtnjieadmhummsrutxijujarsgnxbfmynejgcutzmjzqsxafvvettskkgffnlphzcclkzwphrordsqazkxscvdmmiklgxmqepjcyzhulfebhgcwfbwlhsrqxtgodskuvioidoadqykrpwfgtfmeqhbyajduxjpydpuerhwvmyzdnxfoleergrfeeqynviminglktcstmmuezumbvaecgijaoqecqbudjoqpvfroxbshjoyccmgjtmrezibyriselbvdxtfhuftlysdelbiutlgenthbgslvekhceetyptravtrzlbiyqniptvqhprgodassipartxjsptejdmtyyqydfuikbxyosqxaodzplfjienhysfskkokvekoxqowkxplhbyethre

**[문제 2]**

**고민한 점**

전체를 한 번만 순회하면서 최고 점수와 평균 점수를 효율적으로 계산할 수 있는 STL 알고리즘을 고민했다.

**해결 방법**

std::max\_element()를 사용하여서 최고 점수 보유자를 찾아내었다.

std::accumulate()를 사용해 점수를 합산한 뒤, 전체 플레이어 수로 나눠 평균을 계산했다.

**출력결과**

점수가 가장 큰 Player를 찾아 화면에 출력

이름: saa, 아이디: 1272018, 점수: 423455987, 자원수: 377

저장된 글자: dqghbxgaxhxxfixitvadaesevhccbmgpztkdbewydhsufevdquiforylerpvprhkhwtcnbwcujjirgdwczbaoetqauruutibtappitrokpuxottdvnhwsmqicuoucvofnszkueiedrqmkcwrxtmhetlbyeoxlycfhgtoireyjqppkkqzxyndiegnmevkxeglqgsuapigkzfdhbocifxtjjxejbjbmvctyrylkfnyxgqhokojscgjbpljhdrbjyofkhvmqznopbatcgjqvfawqlomsivuzonhqrrfjuviqhqwcautgujfawsklkporaphjeqjicncxvrjcbxorhximtjvlsxqqhvjxdpiiehbjjblsswzcnzhsprlo

평균 점수: 123372392

**[문제 3]**

**고민한 점**

ID가 같은 player들을 그룹화하고, player들을 효율적으로 탐색하는 방법을 고민했다.

**해결 방법**

std::unordered\_map<size\_t, std::vector<const Player\*>>를 사용하여 id별로 Player들을 그룹화 했다. 이를 사용하면 id별로 그룹화를 해서 해당 아이디마다 player를 추가할 수 있기 때문에 해당 자료구조를 선택하였다.

const Player\*로 자료를 설정한 이유는 포인터가 가리키는 객체의 값을 변경하지 않겠다는 의미도 담겨있고, 또한 값을 복사하지 않고 주소만 넘기기 때문에 객체를 전체 복사하는 것보다 공간 절약이 된다.

모든 맵을 순회하면서 해당 id에 대해 벡터의 크기가 2 이상인 경우, 해당 player들의 id와 name을 "같은아이디.txt"에 기록했다.

이 과정에서 중복된 Player의 수를 세어 출력 화면에 출력했다.

**출력결과**

id가 중복된 Player는 총 1637626명입니다.

**[문제 4]**

**고민한 점**

포인터가 가리키는 메모리 (char 배열)를 정렬하고 탐색하는 데 안전성과 정확성을 보장할 방법을 고민했다.

**해결 방법:**

모든 Player객체를 순회하면서, 각 객체의 p.get()부터 p.get() + num까지 영역을 std::sort()로 정렬했다.  
문자 char들이 내부적으로 아스키 값으로 비교되어, 별도의 비교 함수 없이 알파벳 순으로 정렬된다.

정렬된 후 std::count()로 'a'문자의 개수를 세고, 10개 이상인 경우만 카운트하였다.

**출력결과**

'a'가 10개 이상인 player는 1443553명입니다.

**[문제 5]**

**출력결과**

- **Player를 id 기준 오름차순으로 정렬하였을 때**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

- **Player를 name 기준 오름차순으로 정렬하였을 때**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

- **Player를 score 기준 오름차순으로 정렬하였을 때**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**고민한 점**

루프마다 매번 정렬을 수행하면 비효율적이므로, 한 번만 정렬하고 재사용 가능한 구조가 필요했다.

모든 Player 객체를 통째로 복사하여 세 개의 배열로 각각 정렬해 저장하는 방식은 메모리 사용량이 과도해질 수 있다.

다양한 기준 (id, name, score)으로 정렬된 데이터를 저장하는 배열을 만든다면 어떻게 특정 항목과 그 앞뒤 데이터를 빠르게 탐색할지 고민했다.

**해결 방법**

각 기준별로 포인터 배열(sortedById, sortedByName, sortedByScore)을 구성한 뒤, 각각 정렬했다.

id, name, score 기준으로 정렬된 컨테이너에서, 동일한 값을 가지는 Player 객체들이 여러 개 존재할 수 있기 때문에, Player들의 구간을 정확히 찾기 위해 std::lower\_bound()와 std::upper\_bound()를 사용하였고, 이로써 값이 일치하는 모든 Player들과, 해당 구간의 앞뒤에 위치한 Player도 함께 출력할 수 있었다.

id, name, score 기준으로 정렬된 컨테이너에서 검색된 Player의 앞 또는 뒤 Player를 출력할 때, 해당 Player가 컨테이너의 처음이나 마지막 원소인 경우, 존재하지 않는 앞/뒤 Player를 출력하지 않도록 안전하게 처리되어 있다.

사용자 입력을 무한 루프 형태로 받아, 반복적으로 처리할 수 있도록 구현하였다.

**[ 제출한 과제해결 방법이 효율적이라고 주장하는 내용 / 개선할 부분 ]**

배열을 사용할 때 모든 요소를 복사하지 않고, 포인터 배열을 활용하여 기존 데이터를 참조하는 방식으로 구성함으로써 불필요한 메모리 사용을 줄였다. 이렇게 함으로써 정렬 기준별로 데이터를 별도로 보관할 수 있으면서도 공간 효율성을 확보할 수 있었다.

또한 STL의 알고리즘 등을 적절히 활용해 정렬과 탐색을 효율적으로 처리하였다.

다만, 현재 구조에서는 파일속에 잘못된 데이터에 대한 예외 처리가 생략되어 있어, 향후에는 예외 처리나 검증 코드도 함께 구현한다면 더욱 완성도 높은 코드가 될 수 있을 것이다.

**[ 과제를 하면서 느낀 점 ]**

이번 과제를 통해서 C++의 STL이 제공하는 다양한 컨테이너와 알고리즘의 장점을 실감할 수 있었다. 초반에는 unique\_ptr가 포함된 클래스 데이터를 읽는 것에 어려움을 느꼈지만, 스마트 포인터를 적절히 활용하고, read() 멤버 함수를 직접 정의함으로써 문제를 해결할 수 있었다.

또한 여러가지 STL알고리즘을 직접 적용해보며, 어떤 목적을 위해 자료구조와 알고리즘을 선택해야 하는지에 대한 고민의 중요성을 다시 한 번 느꼈다. 특히 무한 반복 처리와 같이 성능이 중요한 상황에서는, 공간을 더 사용할 것인지 아니면 공간을 절약하고 시간을 감수할지에 대한 고민을 통해 보다 효율적인 방법을 찾아낼 수 있었다.

이번 경험을 바탕으로 앞으로의 프로젝트에서도 STL을 적극 활용해 효율적이고 안정적인 프로그램을 만들 수 있을 것이라는 자신감을 얻었다.