

STL(01)

2023 STL 과제

제출일: 2023.05.18

전공: 게임공학과

학번: 2019184020

성명: 윤은지

목차

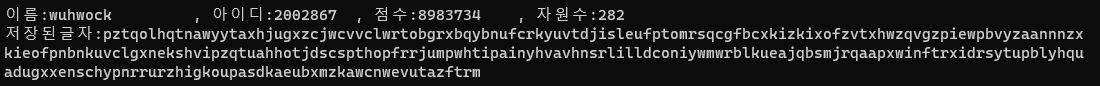
1. 과제 해결 방법
   1. 마지막 객체의 정보 출력하기
   2. 점수가 가장 큰 Player를 찾기, 평균 점수 계산
   3. “같은아이디.txt”에 기록, id가 같은 객체 개수 출력
   4. char 오름차순으로 정렬, ‘a’가 10글자 이상인 것 개수 출력
   5. id를 입력받아 한 번에 화면 출력
2. 제출한 과제 해결 방법이 효율적이라고 주장하는 내용

/ 개선할 부분

1. 과제를 하면서 느낀 점
2. 과제 해결 방법

**1.1 마지막 객체의 정보 출력하기**

출력 결과 :



“2023 STL 과제 파일”이라는 이름의 파일을 이진 모드로 열고, 해당 파일에서 데이터를 읽어와 Player객체의 벡터인 players에 저장하게끔 구현하였습니다.

먼저, pname이라는 문자열 변수를 선언하고 “2023 STL 과제 파일”이라는 값을 할당합니다. 이 값은 파일의 이름을 나타냅니다. 그리고 ifstream객체인 in을 선언하고 pname을 파일 이름으로 사용하여 이진 모드로 파일을 엽니다.

그리고 파일을 읽어서 컨테이너 객체에 저장하기 위해 어떤 걸 쓸지 고민을 했습니다. 컨테이너는 vector를 선택하였고, 이유는 벡터에 정보를 저장한 후 삭제 혹은 삽입할 경우가 없을 것이라고 판단하였기 때문입니다. 크기가 2000,000인 players벡터를 생성합니다. SIZE는 미리 정의된 상수로, 벡터의 크기를 나타냅니다.

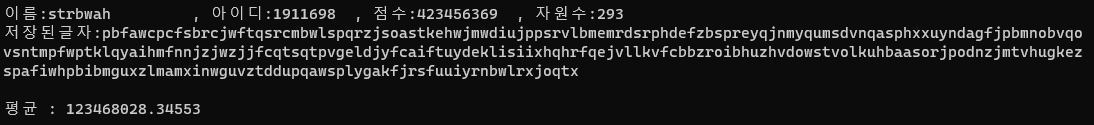
for루프를 사용하여 players벡터의 각 요소에 대해 파일에서 데이터를 읽어옵니다. players[i].read(in)은 Player클래스의 멤버 함수인 read()를 호출하여 파일에서 데이터를 읽어와 현재 인덱스의 Player객체에 저장합니다. read() 함수는 istream& 타입의 입력 스트림을 인자로 받고, this포인터를 사용하여 Player객체의 바이트 데이터를 읽어옵니다. 그리고 p멤버 변수에 대해서는 num개수의 문자를 동적으로 할당한 후 해당 문자열 데이터를 파일에서 읽어옵니다. 마지막으로, read()함수는 읽기 작업의 성공 여부를 나타내는 bool값을 반환합니다.

read()함수는 이진 형식으로 저장된 데이터를 읽어올 때 사용되며, 파일에서 데이터를 읽은 후 해당 데이터를 Player객체에 설정합니다. 함수가 true를 반환하면 읽기 작업이 성공했음을 나타내고, false를 반환하면 읽기 작업이 실패했음을 나타냅니다.

마지막으로, players[players.size()-1]은 players벡터에서 가장 마지막 요소를 나타냅니다. 이를 출력하여 마지막 플레이어의 정보를 화면에 출력합니다.

**1.2 점수가 가장 큰 Player를 찾기, 평균 점수 계산**

출력 결과 :



players 벡터를 점수에 따라 내림차순으로 정렬하고, 가장 큰 점수를 가진 플레이어를 출력하도록 구현했습니다.

sort()함수를 사용하여 players 벡터를 정렬합니다. 정렬 기준으로 compareByScore 함수나 함수 객체를 사용합니다. 이 함수나 함수 객체는 플레이어 객체의 점수를 비교하여 정렬 순서를 결정합니다. 점수가 큰 순서로 정렬되도록 설정되어 있습니다.

정렬된 players 벡터가 비어 있지 않은 경우, 가장 큰 점수를 가진 플레이어를 출력합니다. players[0]은 정렬된 벡터에서 첫 번째 요소로 가장 큰 점수를 가진 플레이어입니다. < 연산자를 사용하여 플레이어의 정보를 출력합니다.

만약 players벡터가 비어 있는 경우, 즉 플레이어가 없는 경우에는 출력할 플레이어가 없으므로 출력을 생략합니다.

compareByScore함수는 두 개의 Player 객체를 인자로 받아서 그들의 점수를 비교하여 더 큰 점수를 가진 플레이어가 먼저 오도록 비교 결과를 반환합니다.

함수의 시그니처를 보면 compareByScore함수는 bool값을 반환하고, const Player& p1과 const Player& p2를 인자로 받습니다. 이는 두 개의 Player 객체를 상수 참조로 받는 것을 의미합니다. 이렇게 인자를 상수 참조로 받는 이유는 함수 내에서 인자를 수정하지 않고 읽기만 할 것이기 때문입니다.

함수의 본문을 보면, p1.getScore()와 p2.getScore()를 비교하여 비교 결과를 반환하고 있습니다. getScore() 함수는 Player 클래스에서 플레이어의 점수를 반환하는 멤버 함수입니다. 비교 결과는 >>연산자를 사용하여 크기를 비교하고 있으며, 반환되는 값은 bool 타입입니다.

결과적으로, 이 함수는 두 개의 Player 객체를 받아서 그들의 점수를 비교하여 더 큰 점수를 가진 플레이어가 먼저 오도록 하는 비교 결과를 반환합니다.

평균은 다음과 같은 방식으로 구했습니다. Sum 변수를 0으로 초기화했습니다. 이 변수는 점수의 총합을 저장하기 위한 용도로 사용되었습니다.

다음 줄에서 accumulate 함수를 호출하여 players 벡터의 요소들을 순회하면서 점수를 더했습니다. Accumulate 함수는 범위 내의 요소들을 합치는 데 사용되었습니다. accumulate 함수의 인자는 다음과 같습니다.

players.begin()과 players.end()는 players 벡터의 시작과 끝을 나타냅니다. 따라서 이 범위 내의 요소들이 순회됩니다. 0LL은 초기값으로 사용됩니다.

accumulate 함수는 각 요소를 이 값과 더합니다. 여기서 0LL은 0을 long long 형식으로 해석하도록 지정하는 접미사입니다.

람다 함수는 각 요소에 대해 실행되며, acc에 누적된 점수를 전달하고 p.getScore()를 더한 값을 반환합니다.

그리고 cout 객체를 사용하여 출력을 포맷하고 결과를 화면에 출력합니다. fixed를 사용하여 소수점 이하의 숫자를 고정된 소수점 표기법으로 출력하고, precision(5)를 사용하여 소수점 이하 5자리까지만 표시하도록 설정합니다.

마지막으로, players.size()로 플레이어의 수로 나누어 평균을 계산하고 출력합니다.

**1.3 “같은아이디.txt”에 기록, id가 같은 객체 개수 출력**

출력 결과 :





텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

먼저 주어진 players 벡터에서 동일한 ID를 가진 플레이어들을 찾아서 파일에 기록하는 함수인 recordSameIdPlayers를 구현한 부분입니다. 같은아이디.txt 라는 이름의 출력 파일을 생성하기 위해 std::ofstream 객체인 outFile을 생성합니다.

다음으로, sortedPlayers라는 새로운 벡터를 생성하여 players 벡터의 요소를 복사합니다. 이는 정렬을 위해 사용될 예정입니다. 그리고 std::sort 함수를 사용하여 sortedPlayers 벡터를 ID에 따라 정렬합니다.

sameIdPlayers 라는 빈 벡터와 정수형 변수 n을 선언합니다. sameIdPlayers는 동일한 ID를 가진 플레이어들을 저장할 용도로 사용되며, n은 동일한 ID를 가진 플레이어의 개수를 세기 위한 변수입니다.

다음은 sortedPlayers 벡터를 순회하면서 동일한 ID를 가진 플레이어들을 찾고 파일에 기록하는 과정입니다. i변수는 현재 플레이어의 인덱스를 나타냅니다.

안쪽의 if문은 현재 플레이어의 이전 플레이어와 ID가 동일한지 확인합니다. 이미 해당 ID를 가진 객체를 파일에 기록했을 경우, 중복 기록을 방지하기 위해 continue를 사용하여 다음 플레이어로 건너뜁니다.

sameIdPlayers 벡터를 비워줍니다.

그다음, 이진 탐색을 사용하여 sortedPlayers 벡터에서 현재 ID와 동일한 플레이어를 찾습니다. std::lower\_bound 함수를 사용하며, 람다 함수를 통해 ID를 비교하여 동일한 ID를 가진 플레이어를 찾습니다. 이때 it는 해당 플레이어의 반복자를 가리킵니다.

while 루프를 사용하여 it가 벡터의 끝에 도달하거나 ID가 다른 플레이어를 가리킬 때까지 sameIdPlayers 벡터에 플레이어를 추가합니다.

마지막으로, smaeIdPlayers 벡터의 요소들을 파일에 기록합니다. for 루프를 사용하여 sameIdPlayers 벡터를 순회하며, 플레이어의 이름과 ID를 파일에 출력합니다. 단, 플레이어가 하나인 경우에는 기록하지 않습니다. 기록할 때마다 n을 증가시킵니다.

마지막으로, n값을 출력하여 ID가 같은 객체의 개수를 보여줍니다. 그리고 outFile을 닫아 파일을 종료합니다.

**1.4 char 오름차순으로 정렬, ‘a’가 10글자 이상인 것 개수 출력**

출력 결과 :



주어진 players 배열의 각 요소들을 처리하여 특정 조건을 만족하는 플레이어의 개수를 세고 결과를 출력하는 부분입니다.

첫 번째 루프에서는 players 배열의 각 요소에 대해 sortP() 함수를 호출하여 p 멤버를 오름차순으로 정렬합니다. 이 부분은 players 배열의 모든 요소에 대해 순차적으로 실행됩니다.

두 번째 루프에서는 players 배열의 각 요소를 순회하면서 다음 작업을 수행합니다. const auto& player는 현재 요소를 나타내는 참조로 선언됩니다.

먼저, player.getP()를 사용하여 현재 플레이어의 p멤버를 string으로 변환합니다. 그리고 변환된 문자열에서 문자 a의 개수를 세는 작업을 수행합니다. std::count 함수를 사용하여 문자열에서 특정 문자의 개수를 세는데, 여기서는 a의 개수를 세고 있습니다. 만약 a의 개수가 10 이상이면, count 변수를 증가시킵니다.

마지막으로, count 값을 출력하여 a가 10글자 이상인 Player의 개수를 보여줍니다. 이후에 빈 줄을 출력하여 결과 출력을 마칩니다.

**1.5 id를 입력받아 한 번에 화면 출력**

출력 결과 :

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이 코드는 무한 루프로 동작하며, 선택지를 제공하고 선택한 작업을 수행하는 기능을 가지고 있습니다.

먼저, 1, 2, 3 중 하나의 선택지를 입력하도록 안내합니다. 선택한 값을 choose 변수에 저장합니다.

다음으로, 찾을 ID를 입력하도록 안내합니다. 입력한 값을 targetId 변수에 저장합니다.

그 후, 잘못된 입력을 한 경우를 처리하기 위해 std::cin.fail()을 사용하여 입력이 실패한 경우를 체크합니다. 입력이 실패한 경우, 에러 메시지를 출력하고 std::cin.clear()를 호출하여 입력 스트림의 에러 상태를 초기화합니다. 또한, 잘못된 입력을 무시하기 위해 std::cin.ignore()를 사용하여 입력 버퍼를 비웁니다. 그리고 루프의 처음으로 돌아가 다음 반복을 시작합니다.

switch문을 사용하여 choose변수의 값에 따라 다른 동작을 수행합니다.

1. choose값이 1인 경우, players 벡터를 ID를 기준으로 오름차순으로 정렬합니다.
2. choose값이 2인 경우, players 벡터를 이름을 기준으로 오름차순으로 정렬합니다. 정렬 기준으로 람다 함수를 사용하여 sort() 함수를 호출합니다.
3. choose값이 3인 경우, players 벡터를 점수를 기준으로 오름차순으로 정렬합니다. 정렬 기준으로 람다 함수를 사용하여 sort() 함수를 호출합니다.

그 후, targetId를 찾기 위해 이진 탐색을 수행합니다.

Vector<Player> 타입인 tp벡터를 선언하여 일치하는 플레이어를 저장합니다. 이진 탐색을 수행하면서 tp벡터에 일치하는 플레이어를 추가합니다.

choose값이 1인 경우와 그 외의 경우로 나누어 처리합니다.

1. choose값이 1인 경우, players 벡터에서 앞 플레이어와 뒤 플레이어를 찾습니다. pre변수와 nex변수에 앞 플레이어의 ID와 뒤 플레이어의 ID를 저장합니다. 그리고 앞 플레이어와 뒤 플레이어를 preV 벡터와 nexV 벡터에 추가합니다.
2. choose값이 1이 아닌 경우, players 벡터에서 해당 ID와 일치하는 플레이어의 앞 인덱스와 뒤 인덱스를 preV 벡터와 nexV 벡터에 추가합니다.

마지막으로, 결과를 출력합니다. choose값이 1인 경우와 그 외의 경우로 나누어 출력합니다. choose값이 1인 경우, 앞 플레이어, 일치하는 플레이어, 뒤 플레이어의 정보를 출력합니다.

만약 tp벡터가 비어 있는 경우, 일치하는 ID를 가진 플레이어를 찾지 못한 것으로 간주하고 해당 메시지를 출력하고, 다음 반복을 시작합니다.

1. 제출한 과제 해결 방법이 효율적이라고 주장하는 내용

/ 개선할 부분

**2.1 마지막 객체의 정보 출력하기**

효율적이라고 주장하는 부분은 다음과 같습니다.

1. 이진 모드 파일 입출력: ifstream 객체를 이진 모드(ios::binary)로 열어 파일을 읽고 쓰고 있습니다. 이진 모드로 파일을 처리하면 텍스트 모드보다 더 효율적으로 데이터를 읽고 쓸 수 있습니다. 특히, 이진 데이터를 다루는 경우에는 효율적입니다.
2. read()함수 사용: read()함수를 사용하여 istream 객체로부터 바이트 단위로 데이터를 읽어오고, 이를 Player객체에 설정하는 방법을 사용합니다. read()함수는 파일에서 바이트 단위로 읽는 것이기 때문에 효율적입니다.
3. 루프에서 조기 종료: for루프에서 players[i].read(in)을 호출하고, 만약 읽기 작업이 실패하면 루프를 종료합니다. 이렇게 함으로써 불필요한 읽기 작업을 줄일 수 있고, 파일의 끝에 도달하거나 오류가 발생한 경우 빠르게 처리할 수 있습니다.

개선할 부분은 다음과 같습니다.

1. 동적 메모리 할당과 관리: Player클래스의 멤버 변수 p는 동적으로 할당된 메모리를 가리키는 포인터입니다. 하지만 코드에서는 new를 사용하여 메모리를 할당하고, 할당된 메모리를 해제하는 부분이 없습니다. 이는 메모리 누수로 이어질 수 있으며, 효율적인 메모리 관리를 위해 동적 할당을 사용하기 전에 적절한 메모리 해제가 필요합니다. 예를 들어, Player클래스 내부에 소멸자를 추가하여 메모리를 해제하는 방식도 괜찮을 것 같습니다.
2. 예외 처리: 현재 코드에서는 파일 입출력 중에 발생하는 오류나 예외를 적절히 처리하지 않고 있습니다. 파일을 열지 못한 경우나 읽기 작업에서 오류가 발생한 경우에 대한 처리가 필요합니다.
3. 반복문 최적화: for루프에서 반복자를 사용하여 컨테이너를 순회하는 것보다 범위 기반 for루프를 사용하는 것이 더 간결하고 가독성이 좋다고 생각합니다. 또한, players 벡터의 크기를 반복문 안에서 평가하는 것이 아니라 미리 변수에 저장하여 사용하는 것이 성능상 더 효율적이라고 생각합니다.

**2.2 점수가 가장 큰 Player를 찾기, 평균 점수 계산**

효율적이라고 주장하는 부분은 다음과 같습니다.

1. 정렬 알고리즘의 선택: std::sort함수를 사용하여 플레이어를 점수에 따라 정렬하는 부분은 효율적입니다. std::sort함수는 일반적으로 효율적인 정렬 알고리즘, 예를 들어 퀵소트 또는 힙소트를 사용하며, 평균적으로 O(N log N)의 시간 복잡도를 가지므로 효율적이라고 생각합니다.
2. compareByScore함수의 활용: compareByScore함수는 플레이어의 점수를 비교하여 정렬 순서를 결정하는 데 사용됩니다. 이 비교 함수를 사용하여 정렬하는 것은 표준 라이브러리의 기능을 활용하는 것이며, 코드의 가독성과 재사용성을 높일 수 있다고 생각합니다.
3. 불필요한 출력 방지: if(!players.empty())를 사용하여 플레이어가 존재할 때만 가장 큰 점수를 가진 플레이어를 출력하는 부분은 불필요한 출력을 방지하여 효율적입니다. 이렇게 하면 플레이어가 비어 있는 경우에는 출력을 수행하지 않으므로 불필요한 작업을 줄일 수 있습니다.

개선할 부분은 다음과 같습니다.

1. 입출력 오류 처리: 현재 코드에서는 is.read((char\*)this,sizeof(Player));와 같이 읽기 작업을 수행한 후 반환값을 bool로 변환하여 오류 여부를 확인하고 있는데 is.read작업 자체가 실패할 경우에도 해당 오류를 처리하는 로직이 필요하다고 생각합니다.
2. player.size()를 미리 변수에 저장하여 반복문에서 매번 호출하는 대신 변수를 사용하는 것이 일부 성능 향상을 가져올 것 같습니다.
3. players벡터가 비어 있을 경우에 대한 예외 처리를 추가하는 게 나을 것 같습니다.

**2.3 “같은아이디.txt”에 기록, id가 같은 객체 개수 출력**

개선할 부분은 다음과 같습니다.

1. 벡터 복사 최소화: std::vector<Player> sortedPlayers = players;에서 복사 생성자를 사용하여 새로운 벡터를 생성하는 부분은 메모리 복사와 할당 작업을 수행하므로 성능에 영향을 줄 수 있습니다. 대신 std::vector<player>sortedPlayers(players.begin(), players.end());와 같이 범위 생성자를 사용하여 벡터를 복사하는 것이 더 효율적일 것 같습니다.
2. sameIdPlayers벡터 재할당 최소화: smaeIdPlayers.clear();를 통해 smaeIdPlayers벡터를 비우는 작업은 반복문마다 재할당을 유발하여 성능 저하를 일으킬 수 있습니다. 대신 smaeIdPlayers벡터를 반목분 밖에서 한 번 초기화하고 반복문 내에서 원소를 추가하도록 변경하면 재할당을 최소화할 수 있습니다.
3. 불필요한 조건문 확인 제거: if(1==samePlayers.size()) continue; 조건문은 smaePlayers벡터의 크기가 1일 때 파일에 기록하지 않는 것을 의도하였지만 sameIdPlayers벡터를 반복문 밖에서 초기화하면 해당 조건문을 제거하고 반복문을 단순화할 수도 있을 것 같습니다.

**2.4 char 오름차순으로 정렬, ‘a’가 10글자 이상인 것 개수 출력**

개선할 부분은 다음과 같습니다.

1. 문자열을 생성할 때 string pString(player.getP()) 형태로 생성하는 대신, const string& pString=player.getP()와 같이 참조 형태로 사용하면 복사 비용을 줄일 수 있다고 생각합니다.

**2.5 id를 입력받아 한 번에 화면 출력**

개선할 부분은 다음과 같습니다.

1. 이진 탐색 대신 std::equal\_range 함수를 사용하여 targetId를 찾을 수 있습니다. std::equal\_range함수는 정렬된 범위에서 동일한 값을 갖는 원소들의 구간을 반환해 주므로, 이를 활용하여 동일한 targetId를 가진 플레이어들을 더 효율적으로 찾을 수 있습니다.
2. preV와 NexV 벡터를 사용하여 앞선 플레이어와 뒤따르는 플레이어를 저장하고 출력하는 부분에서 preV와 nexV벡터를 사용하지 않고 바로 출력하는 방식으로 개선할 수 있습니다. 이미 정렬된 벡터에서 특정 플레이어의 앞과 뒤를 찾기 위해 추가적인 탐색을 수행하는 것보다, 이를 바로 출력하는 것이 더 효율적일 것 같습니다.
3. std::cout을 반복적으로 사용하여 플레이어 정보를 출력하는 것은 성능에 불리한 영향을 줄 수 있다고 생각합니다. 대신, 출력할 내용을 문자열로 미리 구성한 다음 한 번에 출력하는 방식으로 개선할 수 있습니다. 이를 위해 임시 문자열 버퍼를 사용하거나 std::stringstream을 활용할 수 있습니다.
4. 입력이 잘못된 경우에 대한 처리가 부족합니다. 현재는 잘못된 입력이 있으면 오류 메시지를 출력하고 다음 반복으로 넘어가는 방식인데 잘못된 입력을 반복적으로 입력하는 경우 무한 루프에 빠질 수도 있을 것 같습니다. 입력 오류가 발생한 경우, 적절한 처리를 통해 올바른 입력을 유도할 수 있도록 개선할 수 있다고 생각합니다.
5. 과제를 하면서 느낀 점

**3.1 마지막 객체의 정보 출력하기**

데이터를 파일로부터 읽어오는 과정을 구현하면서, 데이터 구조와 파일 입출력의 상호작용을 이해하는 데 도움이 되었습니다. 또한, 동적 메모리 할당과 해제를 적절히 처리하는 방법을 익힐 수 있었습니다. 이를 통해 C++에서의 객체 지향 프로그래밍 개념을 활용하는 경험을 할 수 있었습니다.

그리고 요소의 삽입과 삭제가 일어나지 않기에 벡터를 선택하였는데 5번까지 과제를 해결하고 나서 Array가 더 나은 선택일 수 있겠다는 생각이 들어서 아쉬웠습니다. Array는 첫째로 고정된 크기의 정적 배열을 관리하고, 둘째로 크기가 고정되어 있어 추가적인 요소의 삽입과 삭제가 불가능 하고, 셋째로 요소의 개수가 변경되지 않고 임의 접근이 주로 필요한 경우에 유리합니다. 마지막으로 Vector보다 메모리 사용량이 작고, 임의 접근에서도 성능이 더 우수할 수 있다는 점에서 파일을 읽어서 저장할 컨테이너 객체로 Array를 선택하는 게 맞는다는 생각이 뒤늦게 들었습니다.

**3.2 점수가 가장 큰 Player를 찾기, 평균 점수 계산**

플레이어를 정렬하고 점수를 계산하는 과정을 구현하면서 C++의 다양한 알고리즘과 람다 함수를 활용하는 방법을 익힐 수 있었습니다. 특히, sort함수와 accumulate함수의 활용법을 익히고, 사용자 정의 비교 함수를 정의하여 원하는 기준으로 정렬할 수 있는 경험을 할 수 있었습니다. 이를 통해 C++의 유용한 기능들을 익히고 활용하는 방법을 습득하는 데 도움이 되었습니다.

**3.3 “같은아이디.txt”에 기록, id가 같은 객체 개수 출력**

플레이어 객체들을 정렬하고 이진 탐색을 활용하여 같은 ID를 가진 플레이어들을 찾고, 이를 파일에 기록하는 과정을 구현하면서 C++의 다양한 알고리즘과 컨테이너의 활용법을 익힐 수 있었습니다. 특히, 벡터의 정렬, 이진 탐색, 파일 출력 등의 기능을 적절하게 조합하여 원하는 결과를 얻을 수 있는 경험을 할 수 있었습니다.

**3.4 char 오름차순으로 정렬, ‘a’가 10글자 이상인 것 개수 출력**

이 코드를 구현하면서 문자열 처리와 알고리즘 함수를 활용하는 경험을 할 수 있었습니다. 문자열을 다루는 함수들과 알고리즘 함수들을 적절하게 활용하여 원하는 결과를 얻을 수 있었습니다. 또한, 반복문과 조건문을 활용하여 플레이어 객체들을 처리하고 결과를 계산하는 과정을 구현하면서, 생산적인 프로그래밍 능력을 향상시킬 수 있었습니다.

**3.5 id를 입력받아 한 번에 화면 출력**

이 코드를 구현하면서 사용자 입력 처리, 정렬, 이진 탐색, 결과 출력 등 다양한 기능을 활용하여 프로그램을 작성하였습니다. 또한, 벡터를 다루는 데에도 익숙해지고, 이진 탐색을 활용하여 원하는 값을 찾는 경험을 하였습니다. 코드의 구조와 로직을 이해하고, 생산적인 방식으로 문제를 해결하는 데에 집중하면서 프로그래밍 능력을 향상시킬 수 있었습니다.