**STL 보고서**

폰트, 텍스트, 그래픽, 로고이(가) 표시된 사진

자동 생성된 설명

**2022180024 유영빈**

**[문제 1번]**

[마지막 Player 정보]

이름:jjjg, 아이디:1367890 , 점수:208602632, 자원수:310

저장된 글자: fcjjnbjhedwxaigexmvoldbtvkxpidscojvfcepqntjewoxmopdezaiecxsvhacydhniqzpznybxbkauowuwevzmiyyjyxacgmyflptzqynjmrziwgbbkhbdinogzoitlmrvhvrzdykhyguyidrfkegevlhxgftmxqfjmtcyhzdrcbjegnidnbzosabyctacdgqmezstjcnpftlcmyfxjsfbllofydibuswyimnbdklntkmpxhzvufhrnpmzyvlclmnsauwxqtrpehuwarqoudwhxrugigedkovbnrsbwzsreespdelxiy

1. 크기를 알고 있는 데이터를 저장하기 위해서 어떤 컨테이너를 사용해야 하는가?

2. Player 객체 내부에 문자열, 정수, unique\_ptr을 포함하는데 어떻게 읽어와야 하는가?

**[문제 1번] 해결 방법**

데이터 수가 250만 개로 고정되어 있어 std::array<Player, 2'500'000>를 사용했습니다.

write 함수에서 Player 객체 전체를 메모리에서 파일에 저장한 후, p가 가르키는 메모리도 write 함수에서 저장하였기 때문에, read() 함수를 작성하여 동일하게 구조체 크기만큼 읽은 후, 포인터 대상 데이터를 따로 읽어 메모리에 저장했습니다.

데이터를 메모리에 저장 후, back().print()를 사용하여 마지막 객체를 출력했습니다.

**[문제 2번]**

[최고 점수 Player]

이름:wnnfgpxzyrfynu, 아이디:650894 , 점수:423456430, 자원수:368

저장된 글자: kwselonivnjwkpggrxexbmtsofvvcptulswczeqkbsuihzcrcxsbsfufzdkolxpdbvzwtisumuvrcvphociqtwslylxeisvbvgxgpiymdtsjkujdvhnynyznwzjculkctcwoknictviaeowfyenlevgwcvkpbxqpzdkfmvrhnrkfvlqjjdzersqggdmynsdiparbuqrmxtedbvkcasgouzxkffalrynsrutftahqlrwmpbkwszxfurchltiwupchjnvyvcclkbjkrilesqzvnhlnsgfxgaqwwtvjmleryqvxavgahaucdsxnvkugcpmszaaeppfdrhgewhineqifxxudxjbzhkmwowrvrrbaouksprvn

전체 플레이어 평균 점수: 123512977.60512

1. 250만개의 데이터 중에서 점수가 가장 높은 Player 객체를 어떻게 효율적으로 찾을 수 있을까?

2. 평균 점수는 어떻게 계산하는 게 가장 효율적일까?

3. Player 객체끼리 비교할 때 람다, 외부 비교 함수, operator 중에서 어떤 비교 방식을 사용해야 하나?

**[문제 2번] 해결 방법**

단순 반복문을 사용해도 되지만 수업 시간에 배웠던 알고리즘 std::max\_element()을 사용하여 더 효율적으로 구현해 보았습니다.

STL의 std:: accumulate를 사용하여 전체 점수를 구한 후, 플레이어 객체의 크기를 나눠 평균을 구했습니다. 여기서 외부 비교 함수를 사용했고, 누적 합이 int 범위를 넘을 수 있으므로 long long 타입을 사용했습니다.

또한 std::max\_element()는 < 연산자를 사용하기 때문에 operator< 를 사용하여 score 기준으로 정의해 해결하였습니다.

**[문제 3번]**

중복된 아이디를 가진 Player 수: 1638464명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. 2,500,000명의 Player 중에서 같은 ID를 가진 사람이 몇 명이나 되는지, 그리고 누가 중복되는지 어떻게 찾아낼까?

**[문제 3번] 해결 방법**

이를 해결하기 위해서 unordered\_map<size\_t, size\_t>를 사용해서 각 ID마다 몇 번 나왔는지 집계하였습니다.

첫번째 루프에서 Player 객체의 ID를 id\_count 변수에 저장하며 횟수를 증가, 두번째 루프에서 횟수가 2 이상인 ID를 가진 Player를 같은아이디.txt에 출력했습니다.

**[문제 4번]**

'a' 글자가 10개 이상인 Player 수: 1443076명

1. p는 num의 크기만큼 동적 문자열 데이터를 가지고 있는데, 이를 효율적으로 ‘a’문자만 몇 개인지 효율적인 탐색 방법이 무엇이 있을까?

**[문제 4번] 해결 방법**

효율적인 탐색을 위해서 전체 데이터를 정렬을 해서 맨 앞 ‘a’ 문자만 탐색하기 위해서 sort를 사용해서 정렬했습니다.

선형 루프로 정렬된 문자열을 순회하여 ‘a’문자의 개수를 구했습니다.

**[문제 5번]**

검색 ID : 53466

**[Player를 id 기준 오름차순으로 정렬하였을 때]**

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**[Player를 name 기준 오름차순으로 정렬하였을 때]**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**[Player를 score 기준 오름차순으로 정렬하였을 때]**

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. 250만개의 Player중 ID를 찾을 때, Name을 찾을 때, Score를 찾을 때 마다 정렬을 새롭게 해야 하는가?

2. ID는 중복될 수 있으므로, 연속해서 같은 ID를 가진 Player를 어떻게 찾는가?

**[문제 5번] 해결 방법**

각 기준에 맞춰 250만 개 배열을 반복 정렬하면 시간이 오래 걸린다.

그래서 미리 3개의 정렬된 배열을 복사하고, 이를 사용하여 정렬 시간을 줄였습니다.

이때 3개의 배열의 컨테이너는 이미 데이터의 개수를 알 고 있기 때문에 std::array를 사용했습니다.

std::ranges::find\_if()로 첫 대상 찾은 뒤, 전후 인덱스를 이용해 앞뒤 플레이어를 출력했습니다.

Name과 Score 같은 경우 처음(맨 앞) Player 객체 출력 후, 같은 Name, Score의 객체를 출력 후, 마지막 (맨 뒤) Player 객체를 출력했습니다.

*5번이 시작될 때 3개의 컨테이너를 미리 정렬해서 시간이 조금 걸릴 수 있습니다!*

**[효율적인 부분 / 개선할 부분]**

**[효율적인 부분]**

1. 데이터 수가 고정되어 std::array 사용
2. 복사 생성자 / 대입 연산자 구현
3. std::copy를 사용해 동적 메모리를 안전하게 복사함.
4. 정렬과 탐색을 위한 복사 배열 구성 ( playersId, playersName, playersScore )
5. 평균 누적 계산 시, long long 타입을 사용하여 오버플로우 문제 방지

**[개선할 부분]**

1. std::ranges::find\_if()를 사용해서 250만개의 배열을 매번 탐색 O(n)
2. 정렬을 시킨 후, std::equal\_range() 사용하여 효율성 증가

**[과제를 하면서 느낀 점]**

이번 STL 과제를 진행하면서 그동안 배웠던 STL의 주요 개념들을 실습을 통해 종합적으로 복습할 수 있는 좋은 기회가 되었습니다. STL 수업을 들으며 무심코 작성해오던 코드에 대해 스스로 다시 되돌아보게 되었고, 특히 데이터의 저장, 관리, 처리 과정에서 얼마나 비효율적인 코드를 작성해왔는지 다시금 느낄 수 있는 계기가 되었습니다.

작년 이맘때쯤 윈도우 프로그래밍 수업에서 항상 고정 크기의 정적 배열만을 사용했던 저는 vector를 처음 접하고 그 편리함에 감탄하며 모든 배열과 리스트를 무조건 vector로 처리하는 습관을 가지게 되었습니다. (vector는 만능이야! 라는 발언을 했던 기억이...) 그 당시에는 데이터를 추가하고 삭제하는 것에 대한 깊은 고민 없이 push\_back과 같은 함수를 남발했으며, 이러한 습관은 작은 프로젝트에서는 문제를 일으키지 않았지만, 대용량 데이터를 다루는 상황에서는 명확히 비효율적임을 STL 수업과 과제를 통해 알 수 있었습니다.

이번 STL 과제에서는 250만 개의 Player 객체를 한 번에 메모리에 올리고 효율적으로 관리하는 과정에서, 컨테이너 선택의 중요성, 정렬과 검색의 성능 차이, 메모리 관리의 효율성을 몸소 경험하게 되었고, 특히 std::array와 std::vector의 사용 목적과 상황에 따른 선택 기준에 대해 깊이 고민하게 되었습니다.

또한 복사 생성자, 대입 연산자, 이동 생성자 같은 특별한 멤버 함수의 필요성과 그 중요성에 대해서도 실습을 통해 확실히 이해할 수 있었습니다.

무엇보다 이번 과제를 통해 코드를 왜 이렇게 작성해야 하는가? 라는 질문을 항상 염두에 두고, 수업에서 배운 원리를 직접 내 코드에 적용해보는 과정에서 이전보다 더 신중하고 구조적으로 코드를 작성하려는 습관을 가지게 되었습니다. 많은 데이터를 메모리에 어떻게 안전하게 관리하고, 검색, 정렬 등의 과정에서 어떤 방식이 더 효율적인지를 직접 테스트하고 고민한 시간은 매우 의미 있었습니다.

앞으로도 개인 프로젝트나 졸업작품, 더 나아가 실무에서 STL을 어떻게 올바르게 활용해야 하는지를 고민하는 습관을 더욱 발전시켜야겠다는 다짐을 하게 되었습니다. STL수업을 통해 C++의 메모리 관리, 컨테이너 사용에 대한 깊이 있는 이해를 쌓을 수 있어서 매우 유익한 시간이었습니다. 감사합니다!