STL 과제 보고서

2021182009 김태순

# 문제 1

Player는 string name, int score, size\_t id, size\_t num, unique\_ptr<char[]> p를 저장하는 클래스로 문제에서 저장함수 write를 제시하였다. 이 경우 기본적으로 저장방식을 반대로 하면 읽어내는 것에 문제가 없지만 player의 경우에는 unique\_ptr이 멤버변수로 존재하기 때문에 저장할 때의 포인터가 가리키는 주소의 값을 읽어낼 때 사용하면 그곳에 어떤 데이터가 저장되어 있는지 모르기 때문에 p의 경우에는 읽을 때 다시 make\_unique를 통해 새로 주소를 할당 받아 그곳에 기존 데이터를 읽어오는 방식으로 구현하였다.

그 이외에는 전체 객체의 수를 알려주었기에 array를 이용해 Player를 저장할 배열을 준비하였고 그 수만큼 파일에서 읽어오게 하였다. 또한 istream의 read를 이용해 Player의 read를 구현하였고 read에서는 Player의 크기만큼 한번 불러오고 p를 새로 할당 받은 다음에 num만큼 읽어오게 구현하였다.

# 문제 2

먼저 STL함수를 적극적으로 이용한 방식으로는 std::max\_element와 std::accumulate를 이용해서 각각 최댓값과 전체점수의 합을 구하는데 사용하였다. std::max\_element는 범위내의 값 중에서 가장 큰 값을 찾아 반복자를 반환해준다. Player는 클래스이기 때문에 비교의 기준을 설정해 주어야 하는데 여기서는 std::less를 작성하지 않고 람다를 통해 Player에 getscore를 이용하여 점수를 기준으로 비교할 수 있게 만들어 주었다. 다음으로 std::accumulate는 범위내의 값을 세번째 인자로 초기화된 값에 전부 합하여 반환해준다. 리턴된 값을 저장할 변수는 double로 설정하였는데 그 이유는 평균을 구할 때 형 변환을 거치지 않기 위해 실수형의 자료형을 설정하였고 score은 int이기 때문에 최대 10자리 숫자이고 25만개의 데이터를 합하면 최대 15~16자리 숫자가 나올 수 있는데 double만으로도 15자릿 수 까지는 저장할 수 있기 때문에 double을 사용하였다. 4번째 인자로 람다식을 이용해 Player의 어떤 값을 이용해 계산할 지를 설정해 주었는데 여기서도 getScore를 이용해 점수를 더해준다.

이렇게 먼저 구현하였는데 함수 두개에서 두번의 루프가 이루어지는 것이 비 효율적이라 생각해 for루프 한 번으로 최댓값과 전체 점수의 합을 구하는 방식을 구현해 보았고 이 방식에서는 최대 점수를 가진 player를 maxPlayer로 저장해두고 루프를 돌면서 비교하며 maxPlayer를 구하였고 이 과정에서 전체 합인 sum에 값을 더해주면서 진행했다.

이렇게 두가지 방식을 모두 구현하고 시간을 측정하며 비교해 보니 두번째 방식이 더 빨라서 이 방식을 채택하였다. 추가로 일반적으로 출력하면 소수점이 과학적 기수법으로 출력되는데 우리에게 익숙한 방식으로 표기되기를 원해서 std::fixed를 사용하여 일반적인 소수점 표기법으로 출력되게 만들었다.

# 문제 3

처음에는 단순하게 이중 루프를 이용해 아이디를 비교하며 찾는 것을 고려하였는데 시간이 너무 오래 걸려 탐색이 끝나지 않아 다른 방법으로 고려해야 했다. 그래서 생각한 것이 id기준으로 Player를 정렬하고 루프를 돌면서 i와 i-1을 비교하면 O(N)의 시간으로 같은 id를 찾을 수 있다고 판단하여 std::sort를 이용하여 먼저 id기준으로 정렬을 시행한 다음에 i=1부터 players.size까지 루프를 돌면서 i와 i-1을 비교하며 같은 id를 찾았다.

이때 연속된 같은 id중에서 처음 같은 값을 찾았다면 같은아이디.txt에 i와 i-1을 동시에 입력해야 하므로 bool변수(isFirstSame)를 이용하여 처음으로 같은 id가 발견되었으면 i-1을 입력하고 isFirstSame을 true로 바꾸어 이후에는 i-1을 입력하지 않게 하고 같은 값이 발견될 때는 항상 i를 입력한다. 같은 값이 발견되지 않았을 때에는 같은 값을 발견하여 입력하고 있었으면 줄바꿈(endl)을 입력하고 isFirstSame값을 false로 바꾼다.

같은 아이디의 개수는 size\_t countsame을 0으로 초기화하고 같은 같은아이디.txt에 값을 입력할 때마다 1씩 증가시킨다. 또한 값을 입력할 때에는 이름과 id를 getName, getId를 이용해 얻어와 입력한다.

# 문제 4

이 문제는 for루프를 이용해 모든 Player의 p에 저장된 char[]를 정렬하였는데 이때 p에 접근하는 것은 getp를 통해서 p가 관리하는 포인터를 얻어오고 getnum을 이용해 num개만큼 있다는 것을 표현하여 std::sort를 이용하여 정렬하였다. 이 경우에는 char을 오름차순 정렬하는 것은 따로 비교식을 코딩하지 않아도 가능하기 때문에 별도의 람다식이나 함수는 필요하지 않다.

정렬 이후에 a의 개수를 세는 것은 std::count함수를 이용하였다. Std::sort와 같은 방식으로 범위를 지정하고 마지막 인수로 찾을 값을 입력하는 일치하는 값의 개수를 리턴해준다. 이때 리턴받은 값은 개수를 세는 것이기 때문에 양수의 값을 담는 size\_t로 받았다.

추가로 루프를 빨리 끝내는 방식으로 구현한다면 시간이 단축될 것이라 예상하고 for루프를 이용해 a가 끝나는 지점에서 바로 break를 이용해 루프를 탈출하는 형태도 구현해 보았는데 이 방식은 오히려 break분기로 인하여 컴파일러에서 최적화를 할 수 없어 더 느린 결과를 보여주었다.

# 문제 5

이 문제를 풀 때 고려한 것은 3개의 배열을 만들어 각각 id, name, score를 기준으로 정렬해 탐색알고리즘을 이용하여 문제를 해결하는 것과 한 개의 배열을 이용해 매 탐색 전에 기준을 바꿔가며 정렬을 실행하는 것 중에 어떤 것이 더 효율적인지 고려하였다.

먼저 3개의 배열을 따로 정렬해두는 것은 처음에는 기존 players배열을 복사해서 생성하는 방법을 고민하였는데 그렇게 된다면 25만개의 배열을 3개 복사하는 것이라 복사하는 것도 오래 걸릴 것이고 메모리도 효율적이지 않다고 생각했다. 그 다음 생각한 것은 복사하는 것이 아니라 포인터의 배열을 만들어 실제 복사가 아닌 포인터를 이용해 정렬된 배열을 관리하는 것이다. 이렇게 한다면 복사에 들어가는 비용과 3개의 배열을 추가한다는 이 방식의 단점을 어느정도 완화 가능하다. 이 방식에서의 탐색 방법은 std::lower\_bound를 이용했는데 이 함수는 지정된 범위에서 인자로 전달한 값보다 작지 않은 첫 원소의 위치를 반복자로 리턴해준다. 그렇게 리턴된 값을 기준으로 뒤쪽으로 같지 않은 값이 나올때까지 end반복자를 이동시켜 [start,end)의 범위를 탐색한다. 이후 prev를 이용해 찾은 범위의 앞의 값을 출력, 루프를 이용해 찾은 범위내의 값들을 출력, end값은 찾은 범위의 다음 값에 해당하므로 end를 출력해 id범위의 값들과 그 앞, 뒤 값까지 출력할 수 있다. 이 상태에서 찾은 id범위를 루프를 돌면서 id의 name을 targetName으로 저장한 뒤 그 targetName을 기준으로 id탐색과 같은 방법으로 lower\_bound와 end탐색을 통해 범위를 찾고 출력을 진행했고 score에 대해서도 같은 방법으로 진행하였다.

다음으로 1개의 배열을 매번 재 정렬하면서 std::equal\_range를 이용한 방법을 고려해 보았다. 먼저 equal\_range는 정렬된 상태에서 입력한 값과 일치하는 값들의 범위를 std::lower\_bound, std::upper\_bound의 값을(반복자를) std::pair의 형태로 반환한다. 출력하는 방식은 위의 3개 배열에서 사용한 것과 유사하게 prev와 for루프, equal\_range로 얻은 pair의 second(위 방식에서의 end에 해당한다.)를 이용해 출력하였다. 이후에 name과 score를 탐색할 때에는 찾은 id의 name과 score를 set자료구조에 따로 저장하고 이렇게 저장한 값들을 기준으로 equal\_range를 사용해 탐색하였다. set을 사용한 이유는 혹시 모를 중복 때문에 같은 값이 여러 번 출력되는 것을 피하기 위해 중복이 허용되지 않는 set을 이용하였다.

이 두가지 방법을 고려할 때 처음에는 3개의 배열을 저장하는 방식이 속도는 빠르더라도 메모리 측면에서 불리하다고 생각했는데 포인터를 이용해 배열을 관리하는 것과 배열 1개 방식에서 set을 사용해 값을 저장하는 것을 고려하면 처음에 25만개의 Player배열 3개를 따로 복사하는 방식과 배열 1개 방식을 비교할 때보다는 메모리 효율이 크게 차이 나지 않을 것이라 생각했다. 무엇보다 메모리 부분에서 크게 차이 나지 않는다면 3개의 정렬된 배열을 따로 저장하는 방식이 매번 정렬하는 것보다 압도적으로 실행속도가 빠르기 때문에 3개의 배열을 사용하는 방식을 채택하였다.

# 효율적이라고 주장하는 내용 및 개선할 부분

먼저 효율적이라고 생각하는 부분은 전체적으로 쓸데없는 루프문 없이 STL함수를 적절히 이용하여 컴파일러가 최적화할 수 있게 설계하였다. 3번 문제에서 for루프를 사용한 부분은 STL함수인 equal\_range나 lower\_bound, find같은 함수를 사용해 같은 id 값의 범위를 찾아내고 이후에 또 저장하기 위한 루프를 실행하는 것 보다 한번의 루프에서 같은 값이 있으면 저장하는 방식으로 구현하는 게 더 효율적이라 생각해서 이 부분에서는 STL함수를 사용하지 않았다. 또 5번문제에서는 메모리와 실행속도 측면에서 여러 방식을 비교하였고 기존 방식을 보완하면서 찾아낸 해결책이기 때문에 효율적인 방식이라고 생각한다.

그에 비해 개선이 가능할 것이라 생각하는 부분은 4번 문제에서 두개의 함수에서 각각 루프를 돌기 때문에 비 효율적이라 생각하고 가능하다면 2번 문제의 해결책처럼 한 번의 루프에서 정렬하는 것과 동시에 a의 개수를 세는 것이 가능하다면 개선 가능하다고 생각한다.

# 느낀 점

먼저 굉장히 단순해 보이는 문제여도 그 해결방법은 무수히 많고 그만큼 최적이라고 말할 수 있는 방법도 여러 개여서 코딩할 때에는 항상 어떤 것이 우선순위가 높은 지 고민해야 한다는 것을 깨달았고 또한 내가 열심히 고민하는 문제들은 과거에 누가 이미 고민했던 문제고 STL함수에는 그러한 사람들이 만들어 놓은 함수가 굉장히 많다는 것을 실감할 수 있었다.

그러한 STL 함수들은 직접 코딩하는 것보다 함수를 이용하는 방법이 대부분의 경우에서 성능적으로도 효율적이고 편한 방법이지만 2번 문제의 경우처럼 편하다고 STL함수를 아무런 고민도 하지 않고 이용하면 성능이 오히려 떨어질 수 있기 때문에 항상 코딩할 때에는 고민을 멈추지 말아야 한다는 것을 느꼈다.

최종적으로 STL을 배우는 이유는 그저 암기해서 적어내기 위한 것이 아니라 함수의 원리를 파악해서 그 함수를 사용할 최적의 장소를 찾아내기 위한 것이라고 생각했다.

# 출력 결과

**==========1번==========**

**이름:ltzg , 아이디:330264 , 점수:105344238 , 자원:559**

**elqjlvaoijtheyspynmwoxydiwdijtfljqukeytuamspadumedqwxraaxnvvkbhssagoduzvxruvkkpuamausxhmecydvrfcdeluzyzurnpvcvkedfwnhactsnhyybucccthmeolxiwtoaiaoqmqboyghpwukcxnpccodqahfsfasgibxqpbsqlmegsemfyxfdnhuoazqtaqggxuonsoawhrktcmyoghrkpwoahohkeoavbnxvkigtjclvupkqldnjtkpaupesnwojiekozauaemedrjfpasbvzpiceuqfsveymulchvnlsxkyxfneujszhtsextzoclmqqjnhkkmhvezvwmchqzmschxlqeextijrmxruaxishhkodxpixxgdqlvstwbpctenhomixhvzqjpcqccjmeqlposiiiebojvhfwlljnxeoawqatxfixbtsomirfwtgalugvadhnmamskxxmgkfgxyhrjfoyuklvkdofoezlpskmchvuqclqtnqtsmirwogghblcxdampilhwmqnnzcgawnuzttccorhqlv**

**==========2번==========**

**이름:lcfddwjqunghkz , 아이디:55048 , 점수:423456521 , 자원:590**

**hxcupmzdoihiyzxrqbceloblimfvgyjarzhleydcggoeyoxnpxgqxwcaxcxlelaudbbljawlpymdkgsjpmvlyivedqmmrlzxfgjcrkfhilnqhnkukrmvjjfqvcxcrlbeovhvwpihsrkrolmpnnmqtefjjfervjlpubqkcufevfrzciwxohhbgizxmpqpsvlkcksnvwfvkzzqnvjfhnxbnkdofpxqoulyxobyvcosavqsdteumoxczaddjmndoymhwosmfnaasewbobgbduwaszcpinmncwrhktrvfjufgusutwgyycnazgqguvlyuvsfnkqwdoekuawfqxhwtsxvhkudptwweiutsrnmuycjlaujcduqxdxnaqfcelvntqgydwcibyqpxvxplrviqhecmvqubhtqffgbdcwwoeudogtjoufmxmbfhtewgkxlrhvxffhpfkhpixwzdkmqdrjoqkhaqswadhfigefoukqpesfylbcshuqnvvpcrjtcifzxinopumdaqorpbkclmwgkpqufiejinubsbcminndzmbqrcodweilixutkbivpslrdohjapmpmvrwmiv**

**평균 점수:123507194.811554**

**==========3번==========**

**같은 id를 가진 객체 수: 1640050**

**==========4번==========**

**a가 10글자 이상인 객체 수: 1443508**

**텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**