텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**이종현**

**2021184029**

**STL**

**게임공학과**

**[ 목차 ]**

1. **서론**
   1. **과제의 목표**
   2. **과제의 배경 및 필요성**
   3. **보고서의 구성**
2. **본론**
   1. **개발 환경**
   2. **데이터 구조 및 파일 처리**
   3. **파일 입출력**
   4. **과제 해결 방법 및 결과**
      1. **파일을 읽어 컨테이너에 저장하기**
      2. **평균 점수 계산하기**
      3. **Id 비교, 파일에 저장하기**
      4. **메모리의 char을 정렬, 조건에 따라 출력**
      5. **LOOP를 통해 정렬/탐색하기**
3. **결론**
   1. **과제 해결 방법**
   2. **제출한 과제 해결 방법이 효율적이라고 주장하는 내용 / 개선할 부분**
   3. **과제를 하면서 느낀 점**

**1. 서론**

* **과제의 목표**

**C++STL을 활용하여 대용량 객체의 데이터를 효율적으로 처리하는 방법(컨테이너와 알고리즘)을 학습하고, 주어진 과제의 요구 사항을 지켜 해결한 과정을 체계적으로 정리하여 보고서를 작성하려고 한다.**

**다음과 같은 목표 달성에 중점을 두어 과제를 수행한다.**

* + 바이너리 파일로 저장된 Player객체 데이터를 적절한 STL 컨테이너에 효율적으로 읽어 들이고, 데이터 구조에 따른 파일 입출력을 명확히 이해하고 설명할 수 있다.
  + Player 데이터에 대한 다양한 연산(최대/평균 점수 계산, 정렬 등) STL의 다양한 기능을 적절히 활용하여 구현하고, 각 처리 방법과 결과를 이해할 수 있다.
  + 사용한 STL 컨테이너, 알고리즘의 선택의 이유와 구현상의 주요 고려사항(데이터 구조, 크기, 선택한 STL 등)을 충분히 생각할 수 있고, 활용할 수 있다.
  + 일련의 과정들을 논리적으로 재고하고 정리함으로써, 본 STL 강의에 대해 더 높은 이해도를 가지고자 노력한다.
* **주요 과제 내용**
  + 파일을 읽어 컨테이너에 저장하기
  + 평균 점수 계산하기
  + Id 비교, 파일에 저장하기
  + 메모리의 char을 정렬, 조건에 따라 출력
  + LOOP를 통해 정렬/탐색하기
* **핵심적인 해결 방법 및 주요 결과**
  + 데이터 용량이 정해져 있을 때, array를 사용하여 안정적인 접근
  + std::max\_element, std::accumulate 등 효율적인 알고리즘 사용
    - 이미 만들어진 알고리즘(이거보단 빠를 순 없을 걸?)을 파악
  + sort 와 predicate 등에 대해 이해, 알고리즘 사용 가능하도록
  + STL을 참고하여 각 상황에 맞는 라이브러리 사용하기

**2. 본론**

* **개발 환경**

**언어 및 도구**

**IDE :** Visual Studio 2022

**사용 언어 :** 미리보기 – 최신 C++ 초안의 기능(/std:c++latest)를 사용

**빌드** : x64 Release 모드에서 사용

**주의 사항:** using namespace std; 사용 지양

* + 교수님께서 본래 사용하는 것은 피하라고 말씀하셨던 것에 대해 직접 체험해보기 위해서 using namespace std는 사용하지 않았다.
* **데이터 구조 및 파일 처리**

**데이터**: 2025 STL 과제 자료 생성.exe로 생성된  
2025 STL 과제 파일 – 2021184029 파일  
**데이터 규모**: (Player 멤버의 포인터들 값까지 저장 추정) ↓텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**파일 처리:**

Player Class 구조 ( 총 64 바이트, 4바이트 패딩 포함 )

텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

사용 된 쓰기 함수:  
텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**과제 해결 방법 및 결과**

**1. 모든 Player 정보 읽어 컨테이너 저장 및 마지막 Player 정보를 출력한다.**

텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**고민한 내용 및 해결 방법:**

**1. 250만개의 명시된 수의 객체를 저장할 컨테이너, Array**

std::array를 사용하여 컴파일 시 정해진 값을 이용하여 컨테이너 생성  
Contiguous 메모리이므로 연속된 메모리 공간에 데이터 저장, 높은 캐시 히트율을 만들 수 있다.

컴파일 타임에 배열 크기가 정해지기 때문에, 반복자 무효화가 일어나지 않는다. 벡터의 경우 반복자를 이용 중 배열 크기가 변해 메모리를 이사해야 하는 경우가 생기면, 반복자가 무효화가 발생할 수 있다. (기존 메모리를 가리키는 반복자)

뿐만 아니라 크기가 고정되어 있으므로 벡터를 사용할 이유는 딱히 없다.

(확장을 위한 capacity, size등을 유지)

vector를 reserve하여 사용하는 것도 생각해봤는데, 크기가 고정되어 있는 경우  
위의 이유들에 의해 array가 적절한 컨테이너임을 이해할 수 있었다.

성능을 위해서 잘 사용하지는 않지만, array::at을 사용하여, 배열 원소로의 안전한 접근이 가능하다.

**2. 파일을 읽기위한 방법, read**

처음에는 사용된 쓰기함수를 참고하여 write를 read로 치환하여 구현해 보았으나, 오류가 계속 발생하였다.  
write함수의 경우, Player의 정보를 저장하면서, Player내부의 p 포인터 주소를 그대로 저장하는 것을 알았고, 그렇기에 p가 가리키는 free-store의 주소를 추가로 파일에 저장하는 것임을 깨닫을 수 있었다.

문제의 원인은, 이후 파일에서 read할 때 이전에 write에 사용했던 p의 주소를 그대로 가져오기 때문에, 이미 read해온 주소는 유효하지 않은 메모리였음을 알 수 있었다.

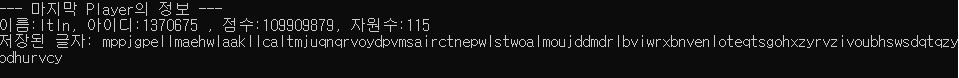
그러나 이러한 문제점을 알아도 해결하는 데에는 꽤 큰 시간이 걸렸다.  
정답은 바로 ‘스마트 포인터의 구조’에 있었다.

p = std::make\_unique<char[ ]>(num)을 통해 p에 새로운 메모리를 할당하면,  
기존 p가 가지고 있던 주소에 대해 참조가 없어지게 되고, 이러한 이유로 스마트 포인터의 구조상 delete 동작을 수행하게 된다.  
그러나 앞서 말했듯이 유효하지 않은 메모리에 대한 delete는 오류를 초래한다.  
그렇기 때문에 발생했던 오류였고, 수업시간에 교수님께서 자주 언급하셨던

p.release()를 통해, 기존에 p가 가지고 있던 포인터의 참조를 반환하고 nullptr로 만들어 유효하지 않은 메모리 주소에 대해 delete하는 문제를 수정할 수 있었다.

최종적으로 읽고 마지막 객체를 출력하는 방식은 print를 이용하여 출력해보기로 하였다, 강의 시간에 배운 내용을 복습하고 싶었다.

출력 방식은 <<operator를 구현할 수도 있었지만, show()로 구현하였다.



**2. 점수가 가장 큰 Player를 탐색, 출력  
Player의 평균 점수를 계산하여 출력**

**텍스트, 폰트, 화이트, 서예이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**고민한 내용 및 해결 방법:**

**1. 가장 큰 점수를 가진 Player를 탐색? 이거보단 빠를 순 없을 걸?, std::max\_element**

물론 직접 탐색하여도 되긴 하겠지만, STL은 이미 만들어진 이유가 있다.  
교수님께서 항상 언급하시던 말, 너가 아무리 잘 만들어도 ‘이거보단 빠를 순 없다’ => STL의 표준 라이브러리 중 하나인 std::max\_element가 단번에 떠올랐다.  
직접 탐색을 구현하고 하는 것보다 일단 명확하게 기능을 알 수 있고, 그렇기에 기억도 잘 남은 것 같다.

뿐만 아니라 코드를 바꾸어야 할 때, 예를 들어 갑자기 300만 객체를 쓴다고 하면, 탐색 코드를 바꾸어야 하는 경우가 있을 수 있으나, max\_element의 인자로 사용되는 begin(), end() 반복자와 predicate(람다를 사용하였다)은 바꿀 필요가 없다.  
또한 나보다 똑똑하신 분들이 만들어 놓고 유지 보수하는 코드를 사용하지 않을 이유가 없다. 따라서 다음과 같이 구현하였다.

Player클래스 내에 비교함수를 구현할 수도 있었지만, 일단 그건 다음 과제에서 구현하기도 하고, 강의 중에 언급되었던 람다가 빠른 이유가 무엇일까 생각해보기 위해 람다로 구현해보았다. (인라인화 등으로 오버헤드 제거) (비교 시에는 getter 사용, setter는 금지!)



**2. 평균을 구한다…모든 값의 합을 구한다…=> std::accumulate**

이 경우에도 바로 사용한 STL라이브러리를 떠올리긴 했지만 사실 조금 난감했다.

지금까지 accumulate는 사용자 정의 클래스가 아니라 기본 자료형으로 이루어진 컨테이너에 대해 사용하기만 했기 때문이다.

따라서 Player등 사용자 정의 클래스로 사용할 때는 주로 람다를 사용하였기 때문에 관련 방법이 없나 많이 고민하였다. 아니나 다를까 accumulate의 인자로 함수 객체(callable)을 받을 수 있음을 알 수 있었다.

[**cppreference**](https://en.cppreference.com/)를 참고한 결과, 다음과 같은 사용법이 있음을 발견하였다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

물론 이해하기는 힘들었지만, AI의 도움을 받아 return 등의 관계로 어렴풋이 이해해볼 수 있었다. 이에 관해서는 수업에서 한번 설명해 주셨으면 좋을 것 같다.

또한 accumulate의 병렬 버전으로 reduce의 존재 또한 알 수 있었다. 그러나 아직 reduce에 대한 필요성은 느끼지 못해서 자세히 알아보진 않았다. 아마 개선한다면 이런 점이 될 것 같다.

출력 결과는 다음과 같다.



**3. id가 서로 같은 객체를 찾아 “같은 아이디.txt”에 기록, id가 같은 객체는 모두 몇 개인지 출력,  
파일에는 정해진 형식으로 기록한다.**

**텍스트, 폰트, 타이포그래피이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**고민한 내용 및 해결 방법:**

**1. 찾는다..find? 그럼 근데 find를 몇 번을 해야 하지?..**

일단 id가 서로 같은 객체를 찾아야 한다는 점에서 std::find를 사용하는 방법을 고려했었다. 그러나 객체 수는 매우 많고, 항상 처음 만난 원소의 반복자를 반환하는 점,

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.혹시나 해서 찾아본 다른 버전의 find에 대해서도 첫번째 원소만을 반환한다는 점에서도 find를 반복하는 건 썩 좋은 방법은 아니라는 생각이 들었다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

마땅한 방법이 생각나지 않아서 AI와 고군분투하면서 알아낸 결과, AI는 unordered\_map을 이용하는 방법을 제안했다.  
결국 목표는 ‘이해’이기 때문에 AI가 제시한 코드를 분석하여 보았다.  
1. 중복되는 id를 가진 객체들을 저장하기 위한 size\_t(중복되는 id) – vector<size\_t>(그 인덱스들)pair의 벡터를 구성한다. ( 물론 reserve가 필요할 것이다. )

2. id를 key로 하고 해당 id를 가진 Player들의 인덱스 목록을 value로 하는 맵을 생성한다.  
3. 순회하며 id 별 그룹화를 한 뒤, 또 다시 순회하며 2개 이상의 중복되는 id를 가진 플레이어들의 인덱스를 vector에 pushback 한다.  
4. 이 id에 따른 vector의 크기가 곧 id가 중복되는 플레이어들의 수를 나타낸다.

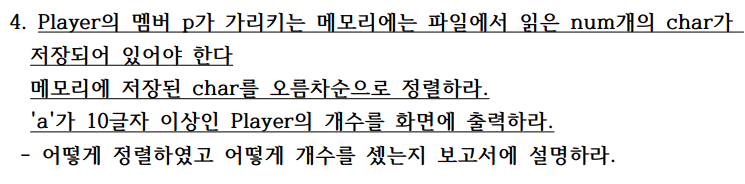
여기서 느낀 점은, 2개 이상의 id, 즉 중복되는 조건을 판명하는 기능을 ranges라이브러리의 std::views::filter를 통해 구현할 수 있을 것 같다는 생각이 들었다.

이를 활용하여 std::views::filter를 이용하여 다시 작성해 보았는데, 이렇게 하면 if문과, 그 if문을 위한 조건 변수 등을 설정할 필요가 없어 코드가 더 간결해지고, 더욱 명확하게 무엇을 하고자 하는지 나타낼 수 있었다.

이후 같은 id를 가진 Player의 인덱스를 저장하는 vector의 크기를 accumulate로 합산하여 중복되는 id를 가진 Player수를 구할 수 있었다.   


이렇게 구한 플레이어의 id와 인덱스 정보를 이용하여 동일한 id를 가지는 객체의 이름을 알 수 있었고, 이를 통해 파일에 저장할 수 있었다.

**4. Player의 멤버 p가 가리키는 메모리는 파일에서 읽은 num개의 char가 저장되어 있어야 한다.  
메모리에 저장된 char를 오름차순으로 정렬하고,  
‘a’가 10글자 이상인Player의 개수를 화면에 출력**



**고민한 내용 및 해결 방법:**

우선 각 Player 객체의 멤버인 p가 가리키는 메모리에 저장된 num개의 char문자를 오름차순으로 정렬해야 했고, 강의시간에 배웠던 std::sort가 컨테이너의 begin()으로부터 정렬할 타입의 정보와 시작주소를 알 수 있음을 기억했다. 그리고 begin()으로부터 생성되는 반복자는, 최근 강의 시간에 언급되었듯이 “포인터를 일반화 한 것”이고, 포인터가 할 수 있는 일을 더욱 일반화 해 놓은 것이라고 할 수 있다. 따라서 포인터의 동작을 수행한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

또한 최근에 우리가 string class를 작성하며 구현한 begin()과 end()에는 각각 p.get()과  
p.get() + \_len을 반환하도록 되어 있다.  
이러한 흐름 속에 Player의 sort 또한 (p.begin(),p.end())에 대해서 다음 변환이 가능하다.  
sort(p.get(), p.get() + num)  
따라서 player객체의 멤버 함수로 자신의 멤버 p가 가리키는 메모리 주소의 char들을 정렬하는 함수 sort\_chars\_in\_p함수에 위 내용을 구현하여 정렬하도록 하였다.

이후에 a의 개수를 세는 알고리즘이 무엇이 있을까 고민하여 보았고, 좋은 아이디어가 없을 지 AI에게 묻기도 해보며 찾은 결과,  
cppreference에서 std::count 알고리즘을 찾을 수 있었다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.  
지정된 범위(range) 내에서, 규정한 기준(predicate)를 만족하는 원소의 개수를 알 수 있는 알고리즘이었다.  
equal to value 인 경우 개수를 센다고 지정되어 있어 지정할 value ‘a’를 주어 비교 연산자를 통해 개수를 세도록 하는 함수를 작성할 수 있었다.  
이 함수를 통해 개수를 세었을 때, 10글자 이상이면 10글자 이상인 글자의 개수를 세는 변수의 값을 증가시키도록 하여 개수를 셀 수 있었다.

**5. LOOP를 통해 id를 입력 받아, 존재하는 id라면  
각각의 멤버에 따른 정렬을 하고, 앞 뒤 객체의 정보를 조건에 맞게 출력한다.**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**고민한 내용 및 해결 방법:**

일단 고려해야 할 사항 중 하나는, While라는 점이다.  
사실상 무한 반복 할 때, 성능의 저하가 없도록 하려면 어떤 방법을 사용해야 할지 고민이 많이 되었다.  
id를 입력 받을 때 마다 탐색한다는 건 While Loop 구성에 적합하지 않다.  
이는 3번 항목에 구현한 id\_map을 통해 id에 따른 인덱스를 바로 반환 받아 해결할 수 있다.  
그러나 이렇게 인덱스를 받아온다 하더라도 name에 따라, 또 score에 따라 매번 정렬해서 그 인덱스 인근의 정보들을 확인하는 건 효율적이지 않다고 판단하였다.

따라서 고민한 결과, 우선 id에 대한 player의 name과 score가 있을 때,   
player의 인덱스들을 name과 score에 따라 정렬한 컨테이너를 만든다.  
그리고 기존 3번 항목에 저장해 놓은 id\_map의 id를 입력했을 때 반환되는 인덱스(들)를 넣어 정보를 출력하도록 하는 아이디어를 세웠다.  
이러한 아이디어에 따라 인덱스 vector를 한번만 정렬 한 후 while에선 이미 정렬된 인덱스 vector를 사용해 효율적으로 객체에 접근할 수 있었다.

뿐만 아니라 더 생각해야 할 점은 같은 ID를 가진 Player들이 같은 이름이나 점수를 가지는 경우에 중복 출력이 발생할 수 있다는 점이었다.  
과제에는 앞뒤 한 명의 Player의 정보만 출력하면 된다고 했지만, 이름이나 점수가 같고I아이디까지 같은 경우에는 여러 번 출력될 수 있는 문제가 있었다.  
이 문제에 대해서는 해결 방법을 생각해보다가 결국 AI로 찾아본 결과 unordered\_set을 통해 중복을 제거하는 방법을 찾을 수 있었다. 중복 제거에 대해 조금 더 생각하게 되었던 것 같다.

그리고 처음에 구현할 때는 막상 앞뒤만 출력하면 되겠다 생각해서 쉬울 줄 알았는데, ID가 중복이 되기 때문에 생각보다 어려운 점이 많았다.  
ID가 똑같으면서 이름과 점수가 다르면 해당 Player들의 앞 뒤를 모두 출력하는 방법을 선택하였다. 생각보다 함수 인자와 for문의 상호작용에 대해 생각할 것이 많아서 많이 헷갈리는 부분이었다.  
그리고 과제에서는 현재 가리키는 id를 가진 객체를 출력해야 할 지 애매해서 고민하다가, 결국 명확한 구분을 위해서 현재 객체도 출력해서 정확히 출력되는 것인지 확인하도록 하였다.

다음은 대망의 5번 항목의 실행결과 화면이다. (아이디는 2014503을 사용하여 확인했다.)

**[Player를 id 기준 오름차순으로 정렬하였을 때]**

**텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

위에 나온 것과 같이 [앞]에 존재하는 id가 또 중복되는 경우가 있기 때문에 생각할 점이 많았다.

**[Player를 name 기준 오름차순으로 정렬하였을 때]**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

동일한 id들이 갖는 name 각각에 대해서, name의 앞 뒤를 출력하도록 구현하였다.

**[Player를 score기준 오름차순으로 정렬하였을 때]**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Score 또한 동일한 아이디들에 대해 다른 score값을 갖기 때문에, 각각의 score값에 대한 앞과 뒤를 구하였다.

**3. 결론**

**제출한 과제 해결 방법이 효율적이라고 주장하는 내용**

기본적으로 때와 상황에 맞는 컨테이너를 찾아 사용하고자 노력하였고, 단순히 구현하기보다 STL에 존재하는 라이브러리들 std::sort, acuumulate 등 상황에 맞는 알고리즘을 찾아 cppreference등을 참고하기도 하며 구현해보기 위해 노력하였다.  
효율적이라고 주장하는 내용은 대부분 **고민한 내용 및 해결 방법**에 주로 적어 표현하였다.

**개선할 부분**

아무래도 조금 더 빨리, 게으름 피우지 말고 더 여유로운 상태에서 시작했다면, 더 다양한 컨테이너의 적재적소 사용법과 다양한 알고리즘을 탐색하여 사용할 수 있었을 것이고, 이러한 사용법을 통해서 어떻게 동작하고 어떤 상황에 써야하는지 더 명확하게 파악할 수 있었을 것 같다.  
특히 for문이 일관성없는 느낌을 받았고 filter 등 range라이브러리의 내용들도 많이 사용하지 못한 것 같다.  
또한 vector등 reserve를 할 때, 적절한 용량을 찾는 방법 등의 일련의 과정을 좀 더 깊고 세밀하게 경험하고 싶었으나, 앞서 말한 게으름에 의해 시간이 부족하여 많이 생각하지 못한 점도 아쉽다.

**과제를 하면서 느낀 점**

정말 알고 알아도 C++은 너무 광활한 것 같다.  
까면 깔수록 계속 나오고 적절한 알고리즘들을 사용하는 상황이 많다는 것을 알 수 있었다.  
그럼에도 교수님께서 알려주신 컨테이너, 알고리즘 등 라이브러리는 STL 중에서 정말 중요하고 쓰임새가 많이 필요한 [정수]로만 구성되어 있다는 것을 과제를 하면서 직접 많이 써보며 알 수 있었다.  
1학년 때 배웠던 C++도 그렇지만, 매번 강의때마다 모르는 것을 알아가는 것이 너무 즐겁고 성장하는 느낌이 너무 좋다.  
기회가 된다면 더욱 많은 유용한 기능 등을 알아가며 게임 제작에 있어 활용할 날이 얼른 왔으면 좋을 것 같다.

**읽어 주셔서 감사합니다. 항상 건강하시길 바라며 마치겠습니다.  
2021184029 – 이종현 드림.`**