**STL 보고서 과제**

2017184034 최정윤

**[ 과제 해결 방법 ]**

**1. 컨테이너 선택**

과제를 해결하기 위해 컨테이너를 선택하는 과정에서 정말 많은 고민을 하였다. 고려를 한 자료구조는 vector와 list 두가지인데, list는 중간 삽입 및 삭제가 빠르지만 vector에 비해 더 많은 메모리를 차지한다는 장단점이 있고, vector는 반대로 메모리 공간이 연속되어 있어 삽입 및 삭제가 list에 비해 힘들지만 list보다 메모리를 덜 차지한다는 특징이 있었다.

두 컨테이너의 장단점과 주어진 과제의 조건을 모두 고려하여 처음 선택한 것은 vector였다. “Player 객체는 추가 또는 삭제되지 않는다”는 조건이 있었기 때문에 삽입 및 삭제 기능이 필요하지 않았기 때문이다. 하지만 문제 4번에서 정렬을 여러 번 수행하는 점이 마음에 걸렸다. Algorithm 헤더에 포함된 std::sort 함수와 list의 내부에 있는 list::sort 함수 중 어느 함수가 더 효율이 좋은 지 알 수 없었기 때문이다. 그래서 vector와 list 두 컨테이너 모두 사용하여 프로그램을 두 번 구현하였다. Typedef 문법을 사용하여 수정이 용이하도록 하였고, 정렬을 하는 로직에는 시간을 출력할 수 있도록 구현하였다.

프로그램의 기본적인 기능을 모두 구현한 후, GetTickCount64 함수를 활용하여 밀리 세컨드 단위의 실행 소요 시간을 계산하였고, 다음 스크린샷은 그 결과이다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

좌측이 Vector와 std::sort를 사용한 결과이고, 우측이 List와 list::sort를 사용한 결과이다. 큰 차이가 나지 않지만, std::sort를 사용하는 편이 평균 소요 시간이 더 짧게 나왔으며, 결론적으로 굳이 시간도 더 많이 들고 메모리 공간을 더 차지하는 List를 사용할 이유가 없겠다고 판단하여 Vector를 사용한 프로그램을 구현하기로 결정하였다.

그 후 컨테이너에 담을 자료형에 대해서도 많은 고민을 하였다. 처음에는 그냥 Player 객체를 담도록 구현하였다. 하지만 Player의 소멸자에 구현해둔 p(메모리의 시작 번지를 담는 동적 문자 배열)의 메모리를 할당 해제하는 코드에서 에러가 발생하였다. 객체를 컨테이너에 담기 위해 임시 지역변수를 선언하였는데, 해당 선언을 한 함수를 벗어나면 지역변수의 메모리가 할당 해제되면서 소멸자가 호출이 되고, 그 과정에서 p 또한 메모리가 해제되었다. 그래서 추후 컨테이너에서 원소에 접근하거나 컨테이너의 메모리를 할당 해제할 때 p의 포인터가 유효하지 않아서 문제가 발생하였다. 이 문제를 해결하기 위해 Player 객체를 동적으로 할당하고, 컨테이너에는 그 포인터를 담는 방식으로 수정하였다.

**2. 파일 읽기 및 Player 클래스**

과제의 조건을 모두 확인하였을 때, Player 객체가 수행해야할 공통적인 기능이 몇 있어 Class 파일을 분리하여 구현하였다. 주어진 파일을 읽어오고 쓰는 기능, 그리고 정보를 출력하는 기능이 공통적으로 필요해보여 멤버 함수로 구현하였다.

과제에서 주어진 write 함수에 따르면, 각 객체 마다 write를 하였기 때문에, read도 유사하게 구현하였다. ReadFile 함수에서 ofstream을 선언하고, 멤버 함수로 그 레퍼런스를 넘기면 한 객체의 크기만큼 데이터를 read를 해오는 방식으로 구현하였다.

하지만 이 과정에서 객체가 문제에 주어진 백만개가 아닌, 백만개의 유효한 객체와 비어 있는 한 개의 객체가 더 추가되어 1’000’001개의 객체가 컨테이너에 담기는 문제가 발생하였다. Release mode의 특성상 디버깅이 어려워 원인을 밝히지는 못하였지만, 문제의 해결을 위해 함수를 호출한 객체의 데이터가 유효한지 확인하는 check 함수를 멤버 함수로 구현하였다. Check함수를 호출하여 해당 객체가 유효할 때에만 vector에 담도록 하였고, 그 결과 정상적으로 백만개의 객체가 담기는 것을 size 함수로 확인했다.

**3. 문제 해결**

Main 함수를 간결하게 하기 위하여 각 문제에서 요구하는 기능은 각각의 함수로 분리하였다. 함수의 인자로 main 함수에서 선언한 컨테이너의 레퍼런스를 받아오도록 하여 각 함수에서도 컨테이너에 접근할 수 있도록 구현하였다.

1번 문제에서 요구한 마지막 원소의 정보를 출력하는 기능은 LastElement 함수에 구현하였다. 처음에는 반복자를 활용하여 end의 위치에서 1을 뺀 위치의 객체의 정보를 출력하도록 하였다. 하지만 컨테이너의 멤버함수를 둘러보던 중 front와 back이 있는 것을 확인하고, 조금 더 간결한 코드를 위해 반복자를 사용하지 않고 back 함수로 마지막 객체에 바로 접근하도록 수정하였다.

2번 문제는 AverageScore 함수에 구현하였다. 각 객체의 점수가 모두 큰 숫자들이어서 총합 점수의 자료형을 int로 선언하였을 때 메모리를 초과하여 에러가 발생하였었다. 이는 최대 2147483647을 담을 수 있는 4 바이트 크기의 int보다 2배 크고 최대 9223372036854775807i64까지 담을 수 있는 8 바이트 정수 자료형인 long long을 사용하여 해결하였다.

3번 문제는 MemorySize 함수에 구현하였다. Player의 멤버 함수인 num이 500인지 확인하여야 했기에 find\_if 함수를 사용할까 하였지만, 해당하는 모든 객체를 출력해야 했기에 가장 먼저 발견된 원소를 반환하는 find 함수를 적합하지 않다고 생각하였다. 그래서 범위 기반 for문으로 모든 원소를 순회하고, num이 주어진 바이트 수와 같다면 멤버 함수인 write를 호출하도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

3번에서 실행한 결과에 따르면 메모리 바이트가 500인 객체 989개가 파일로 출력되었으며, 해당 파일의 크기는 534’060 바이트인 것을 확인하였다.

4번 문제는 RenderInfo에 구현하였고, 해당 함수에서 사용된 함수에는 임시로 생성한 컨테이너를 정렬해주는 CreateSortedVector와, 반복자를 인자로 넘기면 앞과 뒤의 원소의 정보를 출력하는 함수인 BeforeAfterInformation 함수가 있다. ID를 검색할 때마다 매번 컨테이너를 새로 정렬하는 것은 너무 효율이 떨어질 것 같아 3개의 임시 vector를 선언하였다. CreateSortedVector 함수의 인자에 원본 컨테이너와 임시 컨테이너, 그리고 어떤 기준으로 정렬할지를 나타내는 tag를 넘기면 임시 컨테이너에 원본 컨테이너의 내용을 복사하고, tag에 따라 다른 람다 함수를 사용하여 sort 함수를 호출하도록 하였다. 컨테이너의 복사를 위해 copy 함수를 사용하였는데, 처음에 복사가 되지 않고 임시 컨테이너가 계속 비어 있는 상태로 남아있는 문제가 발생하였었다. 하지만 인터넷 검색을 통해 resize 함수를 통해 복사를 받을 컨테이너의 크기를 재할당 해야 한다는 것을 알게 되어 문제를 해결하였다. ID를 검색하기 전에 정렬을 모두 끝내 둔 후, 조건대로 무한히 ID를 검색할 수 있도록 무한 루프 반복문을 사용하였다. 만약 잘못된 ID를 검색하면 반복문을 넘어갈 수 있도록 continue문을 사용하였다. 검색한 ID가 유효한지 확인되었으면 각각의 정렬된 임시 vector에서 find\_if 함수를 통해 주어진 ID를 찾도록 했다. Find\_if 함수를 사용하여 얻은 반복자를 BeforeAfterInformation 함수의 인자로 넘겨서 해당 원소의 정보를 출력하도록 했고, 만약 원소가 가장 처음 혹은 마지막인 경우 앞 또는 뒤 원소가 존재하지 않기 때문에 그에 대한 예외처리를 하도록 구현하였다.

위와 같은 과정을 통해 다음과 같은 실행 결과를 출력하게 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1번과제부터 3번과제까지의 해결 결과는 다음과 같다.

- 1번 문제의 해결 결과 : 이름 xiywmrralbh, 아이디 1000000, 점수 37392053, 자원수 117

- 2번 문제의 해결 결과 : 점수의 평균은 29992180

- 3번 문제의 해결 결과 : 989개의 객체, 534’060 바이트 파일

텍스트이(가) 표시된 사진

자동 생성된 설명

위는 4번 과제의 실행 결과 화면 일부이다.

**[ 요구사항 중 해결하지 못했거나 해결했으나 개선이 필요한 점 ]**

해결하지 못한 문제 사항 중 가장 신경이 쓰이는 것은 데이터를 읽어와서 컨테이너에 바로 담으면 백만 한 개의 원소가 담기는 것이다. 코드를 여러 번 검토해보았지만 어째서 마지막에 한 개의 빈 객체가 담기는지 찾아낼 수 없어서 아쉽다. 객체가 유효한지 확인하는 함수를 추가하여 문제를 해결하긴 하였지만, 문제의 원인을 찾아내지 못해서 완벽한 해결책은 되지 못했다고 생각한다. 추후 릴리즈 모드에서 디버깅이 가능하도록 하는 방법을 찾거나 디버그 모드에서 프로그램을 다시 구현하여 확인하여 해결해보고 싶다.

또한 std::sort 함수와 list::sort 함수의 내부 작동의 차이점에 대해서도 더 많은 조사가 필요하다고 생각한다. 처음 인터넷 검색에 의해 list 내부의 sort 함수가 속도가 더 빠르다는 글을 읽어 실행 시간 측정을 했을 때도 똑 같은 결과가 나올 것으로 예상하였지만 실제 결과는 반대로 나왔다. 두 함수의 비교에 대해서 검색하였지만 명쾌한 답을 주는 글을 찾지 못하여 아쉬웠다.

그리고 정렬한 내용을 담은 임시벡터 3개가 신경이 쓰였다. 반복된 정렬 연산을 하지 않기 위해서 백만개의 객체가 담긴 vector를 3개나 선언했기 때문이다. 현재의 프로그램은 성능을 위해 메모리를 포기하였지만, 두가지 모두를 잡을 수 있는 방법이 있는지 알아보고 싶다.

**[ 과제를 하면서 느낀 점 ]**

수업시간에 다양한 컨테이너와 그 사용법에 대해서 배웠지만, 실제 프로그램을 구현할 때 적절한 컨테이너를 선택하는 것은 많은 연습이 필요하다고 느꼈다. 이번 과제에서 List와 Vector 중에 어떤 컨테이너를 선택할지에 대해 자료 조사를 하던 중, 랜덤 액세스라는 용어를 보게 되었다. List는 랜덤 액세스가 불가능하지만, Vector는 가능해서 탐색의 속도가 더 빠르다고 하였는데 아직 정확히 어떤 개념인지 감은 안 잡힌다. 그래서 각 컨테이너의 내부 작동 방식에 대해 더 많은 조사를 하여 각 컨테이너가 어떤 상황에서 효율적이고 어떤 상황에서 비효율적인지 판단할 수 있는 능력을 기르고, 랜덤 액세스 같은 STL 관련 용어들에 대해 더 많은 조사를 하기로 결심하였다.

또한, 존재하지만 알지 못해서 쓰지 못한 제공 함수가 있는지에 대해서도 조사가 필요하다고 느꼈다. 예를 들어, 4번 문제에서 앞과 뒤의 원소에 접근하는 부분을 -- 연산자를 사용하여 반복자를 직접 이동시켜서 구현하였지만, 수업시간에 교수님께서 하신 상상할 수 있는 기능은 이미 거의 다 구현되어 있다는 말씀처럼 이미 제공되는 함수가 존재할 것 같다. C++ 레퍼런스 사이트에서 제공되고 있는 함수들이 어떤 것이 있는지 더 자세하게 조사할 필요성이 있다고 생각했다

그리고 무엇보다 코드를 많이 치는 것이 많은 연습이 될 것 같다고 느꼈다. 위에서 언급한 내용들에 대해서 다양하게 조사를 하여도, 직접 코드에 적용하는 것이 어려우면 소용이 없다고 생각한다. 앞서 부족했던 내용에 대해서 공부를 하고, 궁금했던 내용들과 새로운 내용들을 모두 정리하고 난 이후에는 교수님께서 수업시간과 과제로 알려주신 코드들을 변형하거나 추가하는 방식으로 공부를 하기로 결심하였다. 다른 컨테이너를 사용해서 재구현하고 실행 속도를 비교한다든가, 새로 발견한 STL 제공 함수를 사용하여 새로운 기능을 추가하는 방식으로 공부하면 다른 프로그램을 작성할 때 STL을 더 효율적으로 사용할 수 있을 것 같다.

지난학기에 컴퓨터 그래픽스 수업을 들으면서, 기말 프로젝트로 간단한 게임을 만들었었다. 맵에 나무를 심는 코드를 구현하였었는데, 그 당시에 STL 컨테이너에 원소를 넣고 빼는 간단한 기능만 알고 있었다. 그 프로젝트는 Vector를 사용하여 구현하였었는데, 이번 과제를 하면서 Vector를 사용한 것은 효율적이지 못한 선택이었다는 것을 알게 되었다. 나무는 게임 도중에 새로 심고 없애는 일이 빈번하였었는데, 이번 과제를 통해 컨테이너들에 대해 조사를 하면서 그때 List를 사용했다면 삽입과 삭제의 연산이 더 빠르지 않았을까 하고 후회했다. 또한 윈도우 프로그래밍 과목의 기말 프로젝트를 구현할 때 플레이어의 아이템을 정렬해야 하는 일도 있었는데, 인벤토리를 1차원 배열로 구현하여서 정렬 코드를 버블 소트 알고리즘을 통해 직접 구현하였었다. 만약 STL 컨테이너와 Sort 함수를 사용했다면 시간도 단축되고 코드도 간결하였을 것 같다. 그래서 STL을 상황에 맞게 사용하는 요령을 기른 후에는 지금까지 만들어왔던 프로젝트들을 STL을 써서 재구현을 하기로 결심하였다. 제공 함수의 존재를 모르고 직접 구현했던 기능들이 많은데, 그런 부분을 모두 고치면 코드도 간결해지고 성능도 올라갈 것으로 기대가 된다.