

게임 서버 프로그래밍 - 라이브러리 연습

지산

February 5, 2024

Contents

I 시작하면서	5
1 왜 C++ 인가?	7
1.1 현재의 언어들을 간략하게 살피기	7
1.2 게임 개발에서 사용하는 언어들	7
1.3 C++의 중요한 변화	8
1.4 왜 여전히 C++인가?	8
1.4.1 빠른 코드 생성과 세밀한 제어가 가능하다	8
1.4.2 다양한 안정적인 라이브러리가 있다	8
1.5 C++을 도구로 포함하기	9
1.6 C++ 연습하기	9
1.7 안내하고 정리하기	9
2 연습을 준비하기	11
2.1 이해하고 연습하는 과정	11
2.1.1 단위 테스트로 이해하기	11
2.1.2 C++ 표준 문서 읽기	11
2.1.3 참고 자료들 검색	11
2.2 C++ 언어 표준은 무엇을 사용해야 하나요?	12
2.3 개발 환경 준비하기	12
2.3.1 비주얼 스튜디오 설치	12
2.3.2 vcpkg 설치	12
2.3.3 doctest 설치	13
2.3.4 첫 테스트 프로젝트	13

II	기반 라이브러리	15
3	무엇을 연습하나요?	17
3.1	C++ 표준 라이브러리	17
3.2	중요한 추가 라이브러리	18
4	container 라이브러리	19
4.1	솔루션 준비	19
4.1.1	프로젝트 생성	19
4.1.2	추가 설정	20
4.2	std::array	21
III	통신	23
5	통신 라이브러리	25
IV	게임 데이터와 데이터베이스	27
6	게임 데이터와 데이터베이스 라이브러리	29
V	서비스	31
7	서비스를 위한 라이브러리와 도구	33

Part I

시작하면서

Chapter 1

왜 C++ 인가?

1.1 현재의 언어들을 간략하게 살피기

llvm의 출현 이후 컴파일러의 마지막 단계인 코드 생성이 매우 수월해지면서 아주 다양한 언어들이 많이 개발되었습니다. 그리고 대체로 개발자 효율을 더 중요시 하고 있어서 go 언어, C#, java, kotlin 등 메모리 관리를 garbage collection 으로 자동화 한 언어들이 많이 쓰이고 있습니다.

C++은 C만큼 빠르게 실행되는 코드를 만들 수 있고, C와 호환되면서 클래스와 템플릿을 포함한 더 강력한 언어를 목표로 만들어졌습니다. 하지만, 언어가 오래 되면서 미묘하게 복잡한 부분들이 늘어나고, 템플릿 프로그래밍이 강력하기는 하지만 사람이 이해하기가 쉽지 않은 면이 있어 근래 시스템 프로그래밍을 하는 곳이 아니면 잘 배우지 않고 잘 쓰이지 않는 경향이 있습니다.

프로그래밍의 근간은 모두 동일합니다. 메모리를 직접 바꿀 수 있는 명령형(Imperative) 언어들은 가장 강력한 기능을 메모리를 바꾸는 사이드 이펙트를 통해 처리합니다. 그런 면에서 튜링 기계의 원시 모델에서 별반 앞으로 나아가지 못 했다고 할 수 있습니다.

Lisp, Ocaml, Haskell, F# 과 같은 함수형 언어들은 사이드 이펙트를 지양하고 순수한 함수들의 조합으로 처리하려고 합니다. 같은 입력에 대해 같은 결과를 돌려주는 순수 함수들로 프로그램을 구성하면 증명 가능하고 버그가 적은 결과물을 만들 수 있기 때문입니다.

이런 함수형 언어의 영향을 명령형 언어들도 많이 받고 있는데 C++의 템플릿 프로그래밍이 순수 함수형 언어의 하나이고, 람다와 같은 함수형 언어의 기능들이 많이 도입되어 있습니다.

그래서, C++이 더 복잡해진 원인이기도 합니다.

1.2 게임 개발에서 사용하는 언어들

게임 개발에서는 클라이언트와 서버를 모두 포함하여 C++을 가장 많이 사용해 왔습니다. 지금도 C++로 개발한 게임들이 많고, 언리얼은 C++로 개발한 클라이언트 엔진입니다.

제가 알기로 거의 대부분의 엔씨소프트 게임들은 C++ 서버이고, 넥슨의 많은 서버들도 C++서버입니다. 웹젠의 게임들도 거의 모두 C++이고 일부 게임들이 C#으로 개발되었습니다.

마비노기를 만든 사람들은 C#을 극한까지 활용하여 멀티플레이도 C# 서버로 제공하고 있다고 알고 있습니다. 크레이지 아케이드 B&B도 C++ 서버이니 대부분의 게임들이 C++이라고 할 수 있습니다.

약간 상황이 바뀐 계기는 초기 모바일 게임들이 멀티플레이를 제공하지 않고도 성공한 경우들이 생기고, C++이 다루기 어렵고 버그가 생기면 치명적인 경우들이 많아 멀티플레이를 제공하지 않는 모바일 게임들은 node.js 기반으로 javascript를 쓰는 경우도 생겼습니다.

이런 게임들 중 대표적인 경우가 쿠키런 같은 게임들인데 개발사인 데브시스터즈가 현재 경영 위기를 겪고 있는 이유 중 하나가 너무 말랑한 게임들 위주로 포트폴리오를 구성한 것 때문일 수 있고, 근간에는 멀티플레이를 잘 제공할 수 있는 기술적인 기반의 준비가 약했던 면도 영향을 미쳤다고 생각합니다.

go로 멀티플레이 게임들 서버를 만들려는 시도들도 있었습니다. go는 가비지컬렉션을 갖는 언어이긴 하지만 컴파일된 코드가 매우 빠르게 동작하는 것으로 알려져 있고, 암호화폐 (이더리움) 에 사용해서 많은 암호화폐 프로젝트들이 사용하고 있고, 쿠버네티스를 포함한 다양한 도구 개발에도 사용하고 있는 좋은 언어입니다.

근래는 러스트로 서버를 만들려는 노력들이 있고 매우 빠른 웹 서버까지는 개발하여 여러 곳에서 서비스 하고 있지만, 아직 게임 서버 개발에까지 도입한 경우는 많지 않아 보입니다.

1.3 C++의 중요한 변화

C++은 현재 언어의 위기 상황을 어느 정도 넘어서는 상태로 보입니다. 가장 큰 위기는 수동으로 메모리 할당과 해제를 하면서 메모리 관련 오류가 발생하기 쉽고, 메모리 오류는 서버가 크래시 (프로세스가 멈춤) 되기 때문에 서비스에 치명적이라는 점에서 기인했습니다.

C++ 발전에 중요한 기여를 하고 있는 boost 프로젝트에서는 shared_ptr, unique_ptr이 오래전에 도입되었으나 언어에 채택되는데까지 너무 오랜 시간이 걸렸습니다.

템플릿 프로그래밍의 복잡도 문제까지 포함하여 위기가 심화되고 있었는데 이들은 C++ 11 표준에서 상당한 수준으로 해결했습니다. 이 때 많은 반성이 있었고, 지금은 3년마다 표준화를 진행하여 언어를 빠르게 발전시키고 있습니다.

여전히 언어가 복잡하고 사용이 어려운 면이 있다는 점은 분명하지만 개선 속도가 빨라져서 다른 언어들과 다시 경쟁할 수 있게 되었다고 봅니다.

1.4 왜 여전히 C++ 인가?

1.4.1 빠른 코드 생성과 세밀한 제어가 가능하다

C++은 시스템 프로그래밍 언어로 세밀한 제어가 가능하고 빠른 코드를 생성합니다. 그래서, 날카로운 칼이라고 할 수 있습니다. 잘 쓰면 유용하지만 잘못 쓰면 매우 위험합니다. 잘 쓰는 연습을 많이 해야 합니다. 그럼에도 불구하고 OS (Operating System, 운영체제)를 만들 수 있는 언어로 생성된 코드가 매우 빠릅니다. 리눅스는 C 언어로 개발되었지만 윈도우는 C++로 개발되었습니다. clang과 같은 컴파일러, SQL 서버와 같은 데이터베이스 등이 C++로 개발된 이유는 세세하게 제어할 수 있고 빠르기 때문입니다.

게임 클라이언트와 서버는 모두 빠른 처리를 필요로 합니다. 그래서, 초기부터 현재까지 많이 쓰이고 있고 앞으로도 많이 쓰일 것입니다.

1.4.2 다양한 안정적인 라이브러리가 있다

게임 개발을 위해서는 다양한 라이브러리가 필요합니다. 2장에서 정리한 라이브러리 외에도 많은 라이브러리들이 있고 C++로 개발되어 오래 쓰이면서 안정된 것들이 상당히 많습니다.

유니티도 내부 엔진은 C++로 개발되었고 IL2CPP처럼 C++로 다시 변환한 후 컴파일해서 사용할 만큼 C++은 빠른 코드를 만들 수 있습니다. 언리얼이 C++로 개발된 이유이기도 합니다. 두 엔진 모두 다양한 외부 라이브러리를 사용하고 있습니다.

1.5 C++을 도구로 포함하기

C#이나 javascript, python으로 게임 서버에 필요한 기능을 빠르게 구현해보면서 다양한 실험을 할 수 있습니다. 모델링 언어로서 매우 좋습니다. 그리고, 아주 빠른 성능을 요구하지 않는 경우 서비스 할 수 있는 게임 서버를 제작하는데도 사용할 수 있습니다.

그래서, 하나의 모델링 언어이자 가벼운 게임을 만드는 도구로 쉽고 빠르게 개발 가능한 언어를 하나 갖고 있으면 좋습니다. C#은 유니티의 영향이 크겠지만 다양한 라이브러리가 잘 포팅되거나 사용 가능한 형태로 제공이 되므로 좋은 언어라고 할 수 있습니다.

하지만 여전히 C#으로는 만들기 어려운 게임들이 있습니다. C++과 실질적인 성능 차이가 몇 배라고 할 수 있을 정도로 아직은 많이 나기 때문입니다. 또 C++로 잘 작성하면 C#보다 더 잘 동작하는 경우도 많습니다. 템플릿이 어렵지만 잘 쓰면 매우 강력하기도 합니다.

그래서, C#과 C++ 또는 python과 C++ 같이 두 개의 짝을 이루는 언어를 나의 도구로 갖추고 있으면 성장을 지속하는데에 큰 도움이 됩니다.

1.6 C++ 연습하기

C++ 표준 문서는 수천 페이지에 달합니다. 이렇게 복잡한 언어를 프로그래머가 전부 다 이해하기는 불가능에 가깝습니다. 따라서, 효율적으로 이해하고 활용할 방법이 필요합니다.

회사에서 한번 진행한 방법으로 참고할만 합니다.

- Tour of C++ 읽으면서 코드 따라하기
- C++ 표준 라이브러리를 STL 중심으로 연습하기
- C++ 표준 라이브러리 코드 분석하기

Tour of C++은 C++ 개발자인 비야네 스트루스트롭이 여행 안내서처럼 쓴 책으로 대부분의 C++ 기능을 가볍게 안내하듯이 설명하고 있습니다. 씹어먹는 C++ 같은 좋은 사이트들을 참고하면서 제시된 코드들을 연습하면 좋습니다.

위 과정이 원활하지 않을 수도 있습니다. 특히, C++ 표준 라이브러리 코드 분석하기는 매우 어려울 수 있습니다. 최고의 C++ 프로그래머들이 오랜 시간 만들어낸 결과물이기 때문입니다. 아름다운 수학 증명이 매우 어려운 경우는 역사 속에서 매우 천천히 많은 노력들이 쌓인 결과물이기 때문인 것과 비슷합니다.

1.7 안내하고 정리하기

게임 서버 프로그래밍 책을 시리즈로 쓰려고 준비한 이유가 위의 내용들입니다.

많은 프로그래머들과 프로젝트들을 진행하면서 사실 우리가 잘 이해하지 못하거나 더 나은 방법이 있는데 잘 몰라서 어려움을 겪는 경우가 많기 때문입니다.

기본은 연습입니다. 목표를 정하고 필요한 것들을 찾아서 꾸준히 연습할 때만 완전해집니다. 인공지능의 시대에도 프로그래밍은 여전히 창의적인 일이고 함께 새로운 세계를 만드는 게임 개발은 지속될 것입니다.

이런 목표로 첫 걸음을 땡니다.

Chapter 2

연습을 준비하기

2.1 이해하고 연습하는 과정

게임 서버를 만들기위해 필요한 라이브러리와 도구는 항상 변화하고 발전합니다. 한 때는 boost를 사용하는 걸 꺼리던 적도 있고, 지금은 표준 라이브러리에 있는 `std::map`과 같은 일반화된 라이브러리 대신 직접 만들어 쓰던 적도 있습니다.

아직 언리얼 엔진을 보면 직접 컨테이너를 만들거나 공유 포인터를 직접 작업하고 쓰고 있습니다. 꼭 필요하다면 할 수도 있는 선택이지만 게임 서버는 대체로 표준화되고 안정적인 라이브러리를 사용합니다.

대신 게임의 차별성을 위해 꼭 필요한 기능들을 새롭게 만들어야 하고, 그런 기능에 집중하는 것이 대체로 낫습니다.

2.1.1 단위 테스트로 이해하기

대신 내부적인 구현이 어떻게 되어있는지, 성능 특성은 어떤지, 잘못 쓰기 쉬운 부분은 어떤지 등 잘 이해하고 써야 합니다. 그러기 위해서 이미 알려진 내용들을 확인하고 직접 단위 테스트 안에서 써보면서 이해합니다.

또 단위 테스트에서 디버깅을 하면서 실행 시간에 코드를 따라가는 연습을 하면서 코드에 대한 이해를 더 자세하게 할 수 있도록 합니다.

2.1.2 C++ 표준 문서 읽기

C++ 표준 문서는 형식이 매우 체계화되어 있고 내용이 방대하여 읽기가 어렵습니다. 하지만, 표준을 읽지 않으면 이해할 수 없는 내용도 많고 체계화 되지 않는 한계도 있습니다.

이를 보완할 수 있는 최적의 자료가 cppreference.com에서 제공하는 문서들입니다.

구글 검색을 해도 C++ 내용들이 많이 참조되지만 직접 사이트에 들어가서 검색해도 됩니다. 상당히 방대한 내용이 잘 정리되어 있고 표준 문서에서 불필요한 내용들도 잘 제외되어 있습니다.

2.1.3 참고 자료들 검색

인터넷에 매우 방대한 양의 자료들이 있습니다. `std::vector`로 검색해보면 매우 많은 내용들이 나옵니다. 그 중에서 cppreference.com, 씹어먹는 C++, [stackoverflow](http://stackoverflow.com) 등의 검색 내용은 거의 항상 신뢰할만한 결과를 제공합니다.

모르면 검색하고, 알려주는 내용을 단위 테스트 내에서 확인합니다.

2.2 C++ 언어 표준은 무엇을 사용해야 하나요?

생각보다 어려운 선택입니다. 현업에서도 그렇습니다. 현재 C++ 표준 모델이 기차 모델로 3년마다 출발하는 기차에 탄 표준 문서들을 표준화하는 방식으로 진행됩니다. 매우 적극적으로 표준화를 따라가는 스튜디오도 있고, 보수적으로 두 개 전의 표준을 따르는 경우도 있습니다. 어떤 경우에는 꼭 필요한 기능이 있어 최신 표준을 따르면서 일부 기능으로 제한하여 사용하도록 규약을 만드는 경우도 있습니다.

이 책에서는 C++20을 기준으로 삼고 C++23의 일부를 선택하여 사용하도록 합니다. TODO: 구체적으로 이것이 의미하는 바는 나중에 자세히 정리하도록 합니다.

2.3 개발 환경 준비하기

wsl(Windows Subsystem for Linux)를 쉽게 윈도우에서도 사용할 수 있어서 clang+cmake+vscode 조합을 사용할 까도 고민했습니다. 우리나라 개발환경은 대체로 Linux를 사용하는 경우에도 데스크톱 환경은 윈도우인 경우가 많고 또 가정에서는 거의 대부분 윈도우를 사용하고 있으므로 비주얼 스튜디오와 비주얼 C++, vcpkg 조합으로 정했습니다.

2.3.1 비주얼 스튜디오 설치

비주얼 스튜디오 커뮤니티 배포판은 무료로 쓸 수 있고 기능 제한도 거의 없습니다. 이를 받아서 설치하는 과정을 이미지를 따서 적고 설명할 수도 있겠으나 인터넷에 자료가 이미 많이 있으므로 직접 진행할 수 있으리라 믿습니다.

이와 같이 이미 알려진 방법들이 있는 경우에는 자료를 활용하도록 안내하려고 합니다.

비주얼 스튜디오 버전은 2022를 사용합니다. 현재 C++20은 거의 대부분 구현되었고 C++23부터는 일부만 구현되어 있습니다.

2.3.2 vcpkg 설치

vcpkg는 마이크로소프트에서 제공하는 cmake와 비주얼 스튜디오에서 바로 사용할 수 있는 형태로 제공되는 C++ 라이브러리 패키지들입니다.

vcpkg 또한 인터넷 검색으로 설치 방법을 확인할 수 있습니다. 처음에 익숙하지 않은 몇 가지 개념들이 있어 사용법 설명은 필요합니다.

Listing 2.1: vcpkg 받기

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
```

Listing 2.2: 비주얼 스튜디오 연동하기

```
./vcpkg.exe integrate install
```

2.3.3 doctest 설치

세쌍(triplet)이라고 불리는 x64-windows-static 과 같이 지정하여 라이브러리를 x64용으로 윈도우에서 정적으로 링크할 라이브러리를 지정하여 설치할 수 있습니다.

Listing 2.3: doctest 64비트 정적 라이브러리 설치

```
./vcpkg.exe install doctest:x64-windows-static
```

게임 서버를 만들 때는 주로 위와 같이 64비트, 정적 링크를 사용하는 경우가 많습니다.

2.3.4 첫 테스트 프로젝트

비주얼 스튜디오를 열고 C++ 콘솔 프로젝트를 생성합니다. 콘솔 프로젝트 생성 방법도 인터넷 검색으로 찾을 수 있습니다.

몇 가지 기본 설정은 안내할 필요가 있고, 아래와 같이 따라서 진행합니다.

Part II

기반 라이브러리

Chapter 3

무엇을 연습하나요?

C++ 표준 라이브러리는 C++ 컴파일러들과 함께 배포되고 있고, 이들 라이브러리 개발자들은 최고 수준의 C++ 프로그래머들입니다. 따라서, 제공되는 인터페이스와 설명 문서로 잘 사용하는 연습을 통해 여러 곳에서 활용할 수 있을뿐만 아니라 C++ 언어에 대한 이해를 높이는데 매우 효과적입니다.

다양한 라이브러리 구성 요소들이 있습니다. 그 중에서 가장 많이 사용하는 컨테이너 자료구조와 알고리즘을 어떻게 쓰는지 배우는 것에서 시작합니다. 필요한 라이브러리 사용법들을 추가로 좀 더 살펴보고 연습한 후에 코드를 읽는 연습을 합니다.

좀 익숙해지면 빅오 (Big O) 기호라고 불리는 알고리즘의 성능 특성을 실제 사용 코드와 프로파일링을 통해 확인합니다. 그 다음은 C++이 어떻게 얼마나 최적화를 잘 하는지 어셈블리 코드를 읽으면서 확인합니다.

3.1 C++ 표준 라이브러리

이해하고 연습할 라이브러리 목록은 다음과 같습니다.

- container 라이브러리 : 여러 자료 구조를 사용할 수 있습니다.
 - `std::array`
 - `std::vector`
 - `std::map`
 - `std::unordered_map`
 - `std::set`
 - `iterator`
- input/output 라이브러리 : 파일과 콘솔 입출력을 압니다.
 - `fstream`
 - `iostream`
 - `filesystem`
- thread 라이브러리 : 쓰레드와 락을 다룰 수 있습니다.
 - `atomic`

- thread
 - mutex
 - condition_variable
 - future
- 메모리 : 메모리 관리와 공유 포인터를 이해합니다.
 - memory
 - new
 - scoped_allocator

3.2 중요한 추가 라이브러리

tbb의 중요한 자료구조입니다.

- concurrent_queue
- concurrent_map

메모리 관리 라이브러리들도 중요합니다.

- tcmalloc
- mimalloc

Chapter 4

container 라이브러리

4.1 솔루션 준비

4.1.1 프로젝트 생성

gsp_library C++ 콘솔 프로젝트를 만듭니다. 해당 폴더의 파일들을 적절하게 정리합니다.

저는 gsp_library 솔루션과 프로젝트를 만들었습니다. gsp_library.cpp 파일의 main을 doctest에서 제공하는 main의 내용으로 교체합니다.

내용은 다음과 같습니다.

Listing 4.1: doctest main 내용

```
#define DOCTEST_CONFIG_IMPLEMENT
#include <doctest/doctest.h>

int main(int argc, char** argv)
{
    doctest::Context context;

    // exclude test cases with "math" in their name
    context.addFilter("test-case-exclude", "*math*");

    // stop test execution after 5 failed assertions
    context.setOption("abort-after", 5);

    // sort the test cases by their name
    context.setOption("order-by", "name");

    context.applyCommandLine(argc, argv);

    // overrides
    // don't break in the debugger when assertions fail
```

```

// context.setOption("no-breaks", true);

int res = context.run(); // run

// important - query flags (and --exit) rely on the user doing this
if (context.shouldExit())
    return res;          // propagate the result of the tests

int client_stuff_return_code = 0;

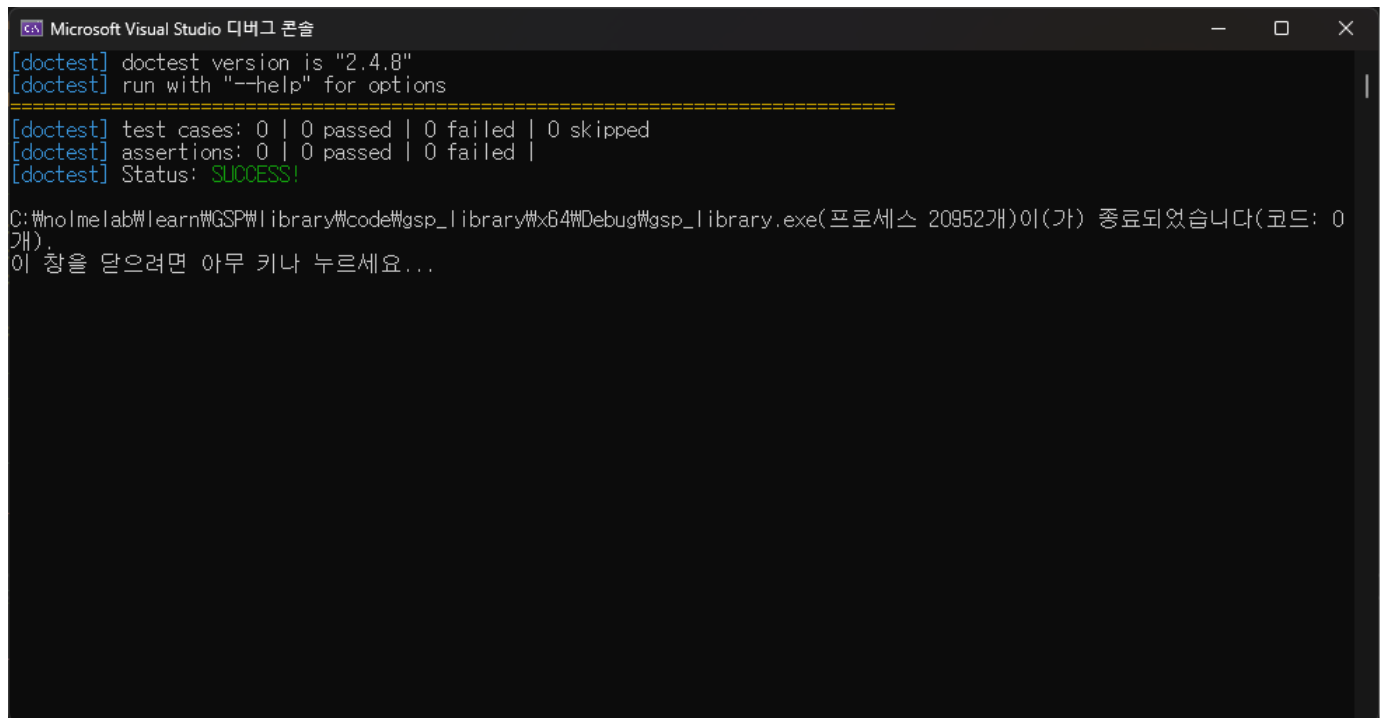
// the result from doctest is propagated here as well
return res + client_stuff_return_code;
}

```

4.1.2 추가 설정

프로젝트 속성의 vcpkg 항목에 Use Vcpkg, Use Static Libraries가 모두 예(Yes)로 설정되었는지 확인해야 합니다. 또 C++ → 코드 생성에 DLL이 아닌 멀티쓰레드로 설정되었는지 디버그와 릴리스 구성 모두 확인합니다. 왜냐하면 정적 라이브러리로 vcpkg에 설치하기 때문입니다.

빌드하고 실행해서 정상으로 동작하는지 확인합니다.



```

Microsoft Visual Studio 디버그 콘솔
[doctest] doctest version is "2.4.8"
[doctest] run with "--help" for options
=====
[doctest] test cases: 0 | 0 passed | 0 failed | 0 skipped
[doctest] assertions: 0 | 0 passed | 0 failed |
[doctest] Status: SUCCESS!
C:\nolmelab\learn\GSP\library\code\gsp_library\x64\Debug\gsp_library.exe(프로세스 20952개)이(가) 종료되었습니다(코드: 0
개).
이 창을 닫으려면 아무 키나 누르세요...

```

4.2 std::array

컨테이너들을 연습할 폴더 container를 만들고 폴더 안에 learn_array.cpp 파일을 추가하고 gsp_library 프로젝트에 적절한 필터를 추가해서 넣습니다. 필터 명으로는 폴더와 같은 container도 괜찮을 듯 합니다.

Listing 4.2: 시작 골격

```
#include <doctest/doctest.h>

#include <array>

TEST_CASE("std::array")
{
    SUBCASE("basic interface")
    {

    }
}
```


Part III

통신

Chapter 5

통신 라이브러리

C++로 TCP 서버를 만들어 클라이언트나 다른 서버와 통신을 할 수 있고, 인증과 같이 웹 호출이 필요한 경우 사용할 수 있는 정도를 준비합니다.

- asio - TCP와 UDP 통신 라이브러리
- cpr - curl 웹 호출 C++ 인터페이스
- sodium - 암호화 라이브러리
- crypto++ - 암호화 라이브러리
- flatbuffrs - 직렬화(Serialization) 라이브러리

flatbuffers를 선택한 이유는 가볍고 빠르기 때문입니다. thrift나 protocol buffers도 선택할 수 있는 직렬화 라이브러리들입니다.

crypto++은 상당히 많은 암호화와 인코딩 라이브러리를 갖고 있고 매우 오래된 안정적인 라이브러리입니다. 라이브러리의 선택은 vcpkg에 포함되었는지 여부도 영향을 미칩니다.

Part IV

게임 데이터와 데이터베이스

Chapter 6

게임 데이터와 데이터베이스 라이브러리

- RapidCsv
- nlohmann json
- nanodbc
- redis

Part V

서비스

Chapter 7

서비스를 위한 라이브러리와 도구

- prometheus
- grafana
- exporter 들
- elasticsearch 로그 수집과 분석 도구