Unit Test Report

]

**팀: 13 강민호 고경모 문무현 안병현**

**제출 일시: 2024 - 12 - 06**

목차

[**1. 개요**](#_e9b76bxdlr6g)

[**2. 도구 선정 근거**](#_e9b76bxdlr6g)

[**3. 설치/실행 방법**](#_rf0rx98vpssx)

[**4. Test Cases 상세 설명**](#_e9b76bxdlr6g)

[**5. 테스트 결과**](#_rf0rx98vpssx)

[**6. 기존 테스트와의 차이점**](#_rf0rx98vpssx)

1. 개요

Unit test framework는 소프트웨어의 개별 구성 요소(함수 등)를 독립적으로 테스트할 때 사용할 수 있는 도구 모음이다. 코드의 특정 단위를 테스트하기 위한 구조와 도구를 제공하여 코드의 정확성을 검증할 수 있다.

Unit test란 코드가 목적대로 동작하는지 검증하는 작업을 말하며 단위의 정확성을 보장하기 위해 실행한다. 이를 위해 검증하려는 요소에 대한 Test Cases를 작성해야 한다.

C 언어에 사용할 수 있는 프레임워크로는 AceUnit, Check, Cmocka, CTest, CUnit, NovaProva, PicoTest, TPT, Unity, MinUnit 등이 있다.

출처 : <https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks>

1. 도구 선정 근거

|  |  |
| --- | --- |
| 프레임워크 | 특징 |
| AceUnit | JUnit 스타일의 테스트 구조, 임베디드 시스템에 적합함. |
| Check | POSIX 환경에서 동작함. |
| Cmocka | 표준 C 라이브러리 외 의존성 없음, C99 표준 지원. |
| CTest | CMake의 일부로 통합됨. |
| CUnit | xUnit 스타일의 테스트 구조. |
| NovaProva | 병렬 테스트 지원. |
| PicoTest | 최소한의 리소스 소비. |
| TPT | 실시간 시스템 테스트에 적합함. |
| Unity | 유연한 플랫폼 지원. |
| MinUnit | 단일 헤더 파일로 구현. |

다음의 프레임워크들을 조사하였으며 Cmocka, CUnit, MinUnit 세 가지의 프레임워크를 주로 고려하였다. 이 중에서 최종적으로 선정한 프레임워크는 Cmocka이다. Cmocka, CUnit, MinUnit을 고려한 이유는 C 언어에 대한 기본적인 단위 테스트를 지원하며, 가볍고 사용 방법이 어렵지 않기 때문이다. 이 중에서 Cmocka를 최종적으로 선정한 이유는 작은 규모의 프로젝트를 테스트하기에 적합하며, 다양한 플랫폼을 지원하기 때문이다. 또한 기존 작성한 코드를 C99 표준에 맞추어 작성하였으므로 C99 표준을 지키고 ANSI C에서도 사용이 가능한 Cmocka를 사용함으로써 표준에 충실한 코드 테스트를 진행할 수 있다고 판단하였다.

3. 설치/실행 방법

Ubuntu에서 Cmocka를 설치하고 실행하는 방법을 알아보자.

3.1. 설치 요구 사항

OS:Linux/Unix 기반 (Ubuntu 20.04 이상 권장)

컴파일러:GCC (버전 9 이상)

패키지 관리:apt 또는 yum

3.2. 우분투 Cmocka 설치 방법:

Cmocka를 설치하는 방법에는 크게 두 가지가 있다.

1. 우분투 패키지 관리자를 사용하여 설치 (apt 사용)

2. <https://cmocka.org/> 에서 설치 후 빌드

이 중 1번 방법을 사용하여 설치를 진행했다.

sudo apt update

패키지 업데이트

sudo apt install build-essential cmake git

cmake와 git 설치

sudsu

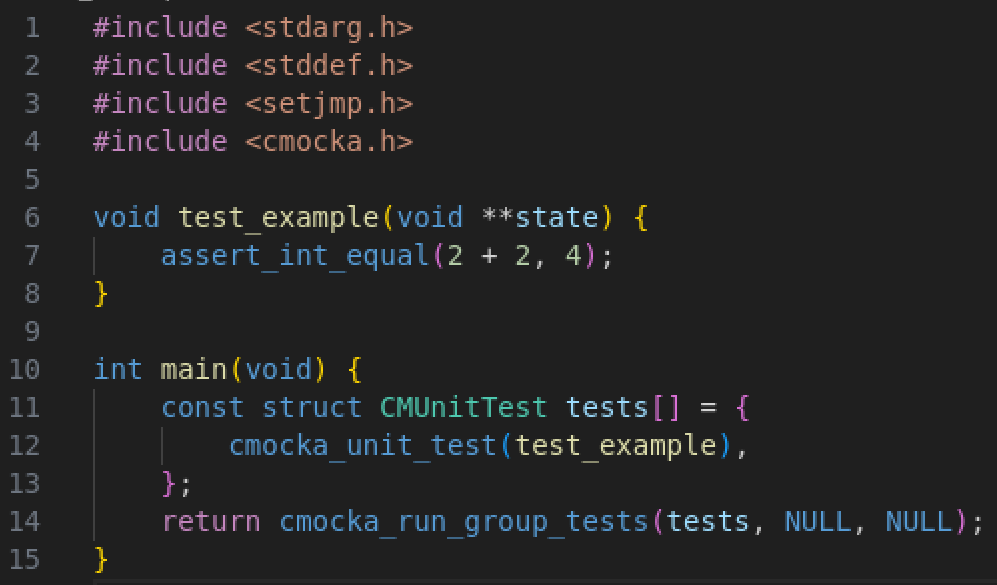
저장소 활성화

sudo apt install libcmocka-dev

Cmocka 설치

3.3. 예제 코드 실행

테스트용 예제 코드 생성



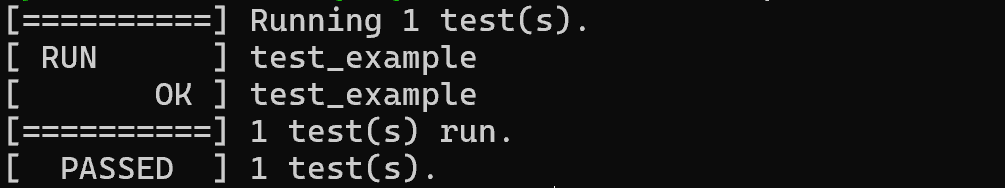
cmocka.h와 setjmp.h를 포함해야한다.

3.4. 컴파일 및 실행

gcc -o test\_example test\_example.c -lcmocka

./test\_example

예제 테스트 실행 결과



4. 테스트 케이스

4.1 DetermineObstacleLocation() 가 제대로 작동하는지 확인하는 case

4.1.1 rv[] 에 제대로 값을 저장하는지 확인하는 case

4.2 DetermineDustLocation() 가 제대로 작동하는지 확인하는 case

4.2.1 0,1을 제대로 반환하는지 확인하는 case

4.3 initqueue() 및 enqueue() 가 제대로 작동하는지 확인하는 case

4.3.1 queue의 init과 enqueue가 제대로 작동하는지 확인하는 case

4.3.2 queue가 가득 찼을 경우 작동하는 case

4.4 isEmpty() 및 isfull() 가 제대로 작동하는지 확인하는 case

4.4.1 queue가 비어 있을 경우 제대로 작동하는지 확인하는 case

4.4.2 queue가 비어 있지 않을 경우 제대로 작동하는지 확인하는 case

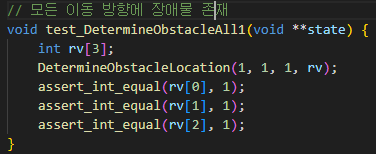
4.5 obstacle[], dust 에 의하여 제대로 작동하는지 확인하는 case

4.5.1 obstacle[]에 의하여 제대로 작동하는지 확인하는 case

4.5.2 dust에도 의하여 제대로 작동하는지 확인하는 case

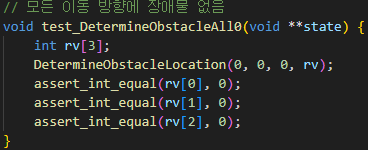
5. 테스트

1) Test Case 4.1.1



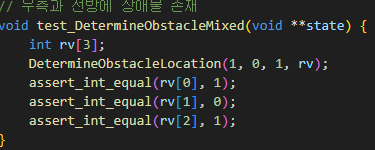


2) Test Case 4.1.1





3) Test Case 4.1.1





4) Test Case 4.1.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



5) Test Case 4.1.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

6) Test Case 4.1.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



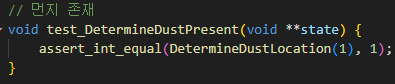
7) Test Case 4.1.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

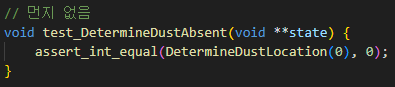


8) Test Case 4.2.1



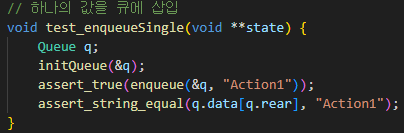


9) Test Case 4.2.1



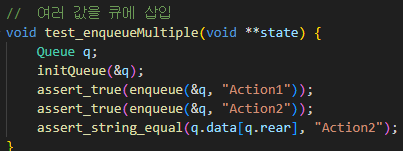


10) Test Case 4.3.1



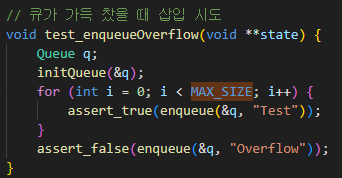


11) Test Case 4.3.1





12) Test Case 4.3.2





13) Test Case 4.3.2

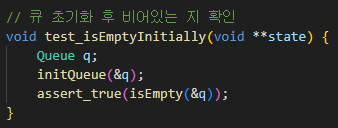
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

14) Test Case 4.4.1





15) Test Case 4.4.1

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명



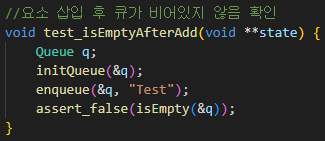
16) Test Case 4.4.2

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



17) Test Case 4.4.2





18) Test Case 4.5.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

19) Test Case 4.5.1

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

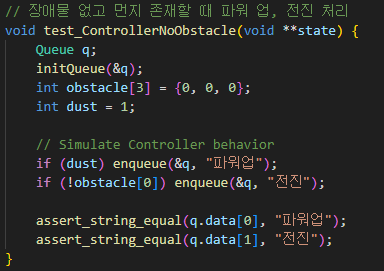
20) Test Case 4.5.1

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

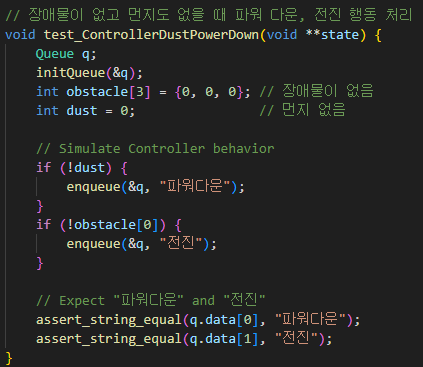


21) Test Case 4.5.2





22) Test Case 4.5.2





6. 기존 테스트과의 차이점

HW3 에서는 컴파일 후 만들어진 output을 사용해서 test를 진행했다. 결과물이 원하는 대로 출력되어 전체적인 프로그램의 완성을 확인할 수 있었다.

HW4에서는 cmocka를 사용해 test를 진행해보았다. HW3보다 컴포넌트 별로 test를 할 수 있어 조금 더 프로그램을 자세히 test할 수 있었다.

케이스를 나누는 것에서도 차이가 있었는데 우리가 직접 실행시킨 파일에서는 장애물의 유무와 먼지의 유무에 중점을 두고 test case를 만들었다. Cmocka를 사용해서 컴포넌트 별로 test를 해야하니, 장애물이 어디 있는지보다 각 컴포넌트 별로 해야할 일을 확인하는 case들을 만들었다.

레고로 비유하자면 cmocka를 사용한 test는 결과물을 만들기 위해 각 파츠들을 맞게 붙이는 느낌이 들었고, 사용하지 않은 test는 만들어진 레고의 팔이 잘 움직이는지, 목이 잘 움직이는지 확인하는 느낌이었다.