**Computer Programming Project 1 Report**

**2014-16757 김보창**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* extra credit A, B 구현했습니다! \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Development/Implementation Environment**

**Hardware**

Cpu : i5-8265U

**OS** : Window 10

**C++**

IDE : VS CODE

Compiler version : 6.3.0

Used library header : <iostream>, <cstring>, <vector>, <fstream>, <regex>, <string>,

<iterator>, <exception>, <random>, <ctime>

**Specific explanation about the code**

**Class Definition**

전체 프로그램에서 큰 부분을 차지하는 클래스들은 다음 클래스들로 나뉜다.

1. 자동차가 주행할 환경을 저장하는 Environment Class와 Environment를 구성하는 Stage Class들.

2. 자동차를 나타내는 Vehicle Class와 그로부터 파생된 Car, Airplane, Submarine Class.

3. 블랙박스 정보를 담는 Blackbox Class

4. 기타 Event 처리를 위한 Exception Class 등등..

**1. Environment Class & Stage Class**

자동차가 주행할 map의 정보와 현재 자동차의 위치로 구성되는 클래스로,

현재 자동차가 어디에 있는지, 어떤 환경을 달리고 있는지를 나타낸다.

Environment Class는 환경의 정보가 저장된 문자열로부터 Road, Sky, Ocean, X, Y 환경을 받아와 각 환경을 해당하는 Stage Class의 Derived Class에 저장한다.

(문자열로부터 환경을 파싱하는것은 strtok과 regex를 적절히 조합하여 구현하였다.)

Stage Class에서 파생된 Class로는 Road, Sky, Ocean, X\_Stage, Y\_Stage 클래스가 있고,

각 클래스는 해당 환경의 정보 (습도, 온도, 각 타일이 몇 유닛인지..)들과

그 정보들을 받아오기 위한 member function을 가지고 있다.

이렇게 해당하는 클래스들을 생성한 다음, Environment Class 내부의 vector에

이렇게 생성한 Stage Class들을 저장하고, 현재 자동차의 위치를 0으로 초기화 한다.

Environment Class는 크게 세가지의 public method를 가진다.

Void onestep\_go() : 현재 자동차 위치를 1칸 전진시키는 method

Stage \*get\_next\_stage() : 다음에 진행할 stage 객체를 가져오는 method. 차가 끝까지 달렸다면 NULL 포인터를 리턴하여 더 이상 진행하는 일을 막는다.

Void print\_map() : extra point를 위한 method로, 현재 자동차의 위치와 맵을 출력한다.

먼저 get\_next\_stage()를 통해 진행할 stage의 정보를 받아오고,

이 정보를 통해 자동차의 상태를 어떻게 바꿀지 결정한 다음,

onestep\_go()를 실행하여 실제 map에서 자동차의 위치를 전진시키는 방법으로 사용하게 된다.

**2. Exception Class**

프로그램에서, 자동차에 문제가 생겼을 때의 구현은 Exception을 통해서 이루어지기 때문에,

Exception Class를 먼저 소개한다. 모든 Exception은 runtime\_error 클래스에서 파생되었다.

CriticalFailException : 디버그를 위한 exception으로, 프로그램 내부에서 치명적인 문제가 발생하면 throw된다.

EnvironmentFailException : 문자열 포맷이 잘못되었거나 등의 이유로 Environment를 생성하지 못할 때 throw한다.

VehicleException : Vehicle의 상태가 바뀔 때 throw되는 Exception들의 base class.

VehicleChagedException : Vehicle이 transform 될 때 throw된다. 자동차가 바뀌기 전 형태와 바뀐 후 형태에 대한 정보를 내부에 가지고 있다.

VehicleFinishedException : Vehicle이 마지막 목적지에 무사히 도착했을 때 throw된다.

VehicleAbortException : Vehicle이 더 이상 진행할 수 없을 때 throw되는 Exception들의 base class.

VehicleEnergyAbortException : Vehicle의 에너지가 0이면 throw된다.

VehicleOxygenAbortException : Vehicle의 산소가 0이면 throw된다.

VehicleDestroyAbortException : Vehicle이 망가져서 더 이상 진행이 불가능 하면 throw된다.

**3. Blackbox Class**

블랙박스 클래스는 자동차에서 블랙박스를 기록하기 위해 사용하는 클래스로,

다음 두가지의 public member function을 가진다.

Void Record(mode, energy, oxygen, speed) : 블랙박스에 해당 정보를 기록한다.

Void print() : 블랙박스에 기록된 정보를 출력한다.

**4. Vehicle Class & Derived Class (Car, Airplane, Submarine)**

자동자 자체의 상태를 저장하는 클래스로, 자동차의 상태를 기록하기 위해 Blackbox Class의

포인터와, 자동차를 실제로 달리게 할 Environment Class의 포인터를 멤버로 가진다.

굳이 Class 자체를 Composition으로 가지게 하지 않고, Class의 Pointer를 가지게 한 이유는, 자동차가 수시로 Transform을 하는데 그때마다 Blackbox와 Environment Class를 각각 생성하고 삭제하면 너무 비효율적이라 생각해서 포인터만 가지게 하였다.

자동차의 Transform은 각각의 Derived Class 사이의 변환을 통해 구현하였다.

**Vehicle Class**

Vehicle Class는 Car, Airplane, Submarine Class의 base class로,

모든 vehicle이 공통적으로 공유하는 member variable들을 가지고 있다.

Speed, energy, distance, oxygen\_level(산소량), mode(현재 자동차가 무슨 모드인지),

blackbox, environment 멤버를 가지고 있으며,

또한 extra credit을 위해 solar\_panel\_defect\_extra 라는 멤버를 가지고 있다.

Solar\_panel\_defect\_extra가 만약 true라면, Car mode에서 solar panel recharge가 작동하지 않게 된다.

Vehicle Class는 각 member variable들의 get, set function들을 제외하면,

크게 5가지의 public method를 가진다.

Virtual void unit\_move() : specification의 first type move에 해당하는 이동을 하는 함수이다.

Virtual void chunk\_move() : specification의 second type move에 해당하는 이동을 하는 함수.

Virtual void print\_status() : 현재 vehicle의 상태를 출력한다.

Void print\_final\_status() : vehicle이 멈출때, vehicle의 상태를 출력한다.

Void throw\_exception\_vehicle\_status\_failed() : 현재 vehicle의 상태가 정상이 아니라면, 해당하는 상황에 맞는 exception을 던진다. 그렇지 않은 경우, 아무런 일도 하지 않는다. 에너지가 0인경우와 산소가 0인경우 exception을 던져주는 함수다.

여기서 virtual로 선언한 method들은 polymorphism을 이용해 쉽게 구현하고자 선언하였다.

또한 Vehicle Class는 명시적으로 Copy constructor를 가지고 있는데, 이는 후에 서술할 Mode change에서 유용하게 사용된다.

**Car, Airplane, Submarine Class**

이 클래스들은 Vehicle Class의 derived 클래스로, 해당 모드에 맞는 member variable들과

Public method들을 가지고 있다.

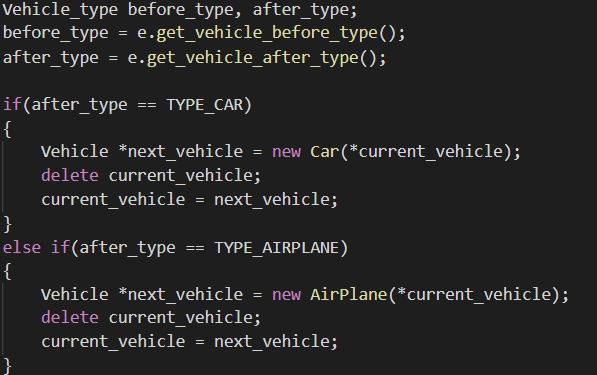
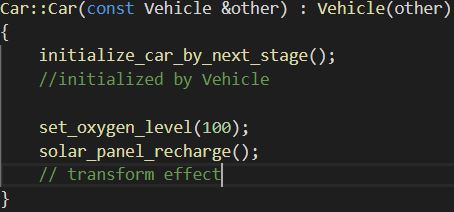
이 클래스들의 특이점으로는 Car class를 제외하면 다음과 같은 Constructor밖에 없다는 것이다.

Constructor(Vehicle other) : Vehicle(other) { class specified statement~~~~}

예를 들어, Airplane air라는 object가 있을 때, 이를 Submarine으로 변환하는 것은 다음과 같은 과정으로 빠르고 간편하게 이루어진다.

Submarine sub(air);

이러한 constructor를 통해, 다른 mode로 돌아가는 자동차 Class를 인자로 받아서, 편리하게 mode change를 할 수 있게 된다.



(Vehicle을 이용한 Car constructor와 다른 Vehicle을 사용해 Car를 initialize 하는 예시)

Car class의 경우에는, 프로그램이 시작할 때의 initialize를 위해 blackbox와 environment class의 포인터들을 받는 Constructor를 추가로 가지며, 이를 통해 초기 설정을 하게 된다.

Vehicle의 Derived Class들은 공통적으로 다음과 같은 public method를 가지게 된다.

Virtual void unit\_move() : first move mode를 실행한다.

만약 move 실행 도중, 자동차의 상태가 변화하면 해당하는 exception을 throw한다.

(mode change, arrived, energy fail, oxygen fail, destroyed)

Virtual void chunk\_move() : second move mode를 실행한다.

만약 move 실행 도중, 자동차의 상태가 변화하면 해당하는 exception을 throw한다.

Void print\_status() : 각 클래스에 맞는 정보들을 프린트한다.

이 클래스들의 내부 구현은, 다음과 같이 이루어진다.

초기화

1. Vehicle이 transform될 때, 그 전 Vehicle을 인자로 받아서 내부 vehicle에 필요한 기본적인 정보들을 만들어낸다.

2. 다음에 진행해야 할 Stage의 정보를 받아와서 해당 Stage의 정보들을 통해 현재 Vehicle을

초기화 한다. 예를 들어, Airplane이라면 Sky의 air density 등을 받아와서 초기화한다.

3. 초기화 후, 진행해야 할 작업들을 수행한다. 예를 들어, Car라면 transform 뒤에

산소를 100으로 채우고, solar panel을 통해 에너지를 모은다.

사용

Unit\_move와 chunk\_move는 내부에서 move\_one\_unit 함수를 이용해 실제 이동을 진행하게 된다.

Move\_one\_unit함수는 이름 그대로 하나의 유닛을 진행하는 함수로,

현재 Vehicle의 상태와 다음에 진행해야 할 Stage의 정보를 통해 자동차의 실제 이동을 담당하게 된다.

Move\_one\_unit 함수는 다음과 같은 과정으로 진행된다.

1. 현재 Stage를 한 스텝 진행한다. Car라면 Road, Submarine이라면 Ocean을 한 스텝 진행한다.

만약 한 스텝 진행 후에 자동차에 문제가 생긴다면, 해당 Exception을 throw 한다.

문제의 파악과 exception throw는 env\_progress\_onestep() 이라는 내부 함수에서 담당한다.

2. 한 스텝 진행 후, 다음 step에 진행할 Stage가 [X], [Y]인지를 파악하고, 그렇다면 X, Y를 처리한다. X,Y 처리중 자동차에 문제가 생겨 자동차를 진행할 수 없다면, 해당 Exception을 throw한다.

3. 그 후, 다음 step에 진행할 Stage가 다른 형태의 Stage로 바뀌는지 파악한다.

만약 바뀐다면, VehicleChangedException을 throw해 현재 상태를 바꿔줘야 함을 외부에 알린다.

4. 모든 경우에서 문제가 없다면 함수는 정상적으로 종료된다.

Unit\_move와 chunk\_move는 만약 move\_one\_unit함수에서 exception이 발생한다면,

현재 상태를 블랙박스에 저장하고, 해당 exception을 다시 throw해 상위 함수에 문제를 떠넘겨

상위 로직에서 문제를 잘 처리할 수 있도록 한다.

5. Main logic

메인 로직은 다음과 같은 과정으로 이루어진다.

1. Extra mode를 할지 말지 사용자 입력을 받는다.

2. 사용자로부터 어떤 file을 사용할지 입력 받는다. 0을 입력받으면 프로그램을 종료한다.

3. 해당 file을 토대로 environment object를 생성하고, blackbox object를 생성한다.

4. environment와 blackbox object의 포인터를 토대로 기본 Car object를 생성한다.

5. 루프를 통해 사용자로부터 입력을 받아 자동차를 움직여 나간다. 입력을 받으면서 자동차의 상태와 지도를 출력한다. 중간에 자동차의 상태에 문제가 생기면 6으로 이동한다.

6. 자동차의 형태가 바뀌어야 하면 자동차의 형태를 바꾸고 5로 이동한다.

만약 자동차가 더 이상 진행할 수 없는 상태라면 더 이상 진행할 수 없음을 출력하고,

현재 할당된 자원을 모두 해제하고 2로 이동한다.

**Troubleshooting points while implementing your code**

이번 숙제의 경우 효율적이고 general한 코드를 짜기 위해 많은 고민이 필요했다.

Vehicle의 형태가 계속해서 변형되는데다, vehicle이 현재 진행할 수 있는지 없는지도 지속적으로 파악해야 했기에 충분히 생각하지 않고 코드를 짜면 유지보수하기가 매우 힘들어 질 것 같았다.

따라서 수업시간에 배운 Error handling과 polymorphism을 적극적으로 활용하여 코드를 짜게 되었다.

Vehicle의 형태가 계속해서 변형되는데, 이러한 변형이 일어나도 speed, energy 등등은 유지됨에 착안해서 base class인 Vehicle에 이러한 정보를 저장하고, Vehicle의 특정한 형태인 Car, Airplane, Submarine등은 base class의 정보를 이용해 최대한 간편한 방법으로 initialize가 가능하게 만들었다.

따라서 사용자는 Vehicle의 transform이 필요할 때, 어떤 형태로 transform 해야 하는지 만 알고 있다면 base class인 Vehicle의 핸들만 가지고 있으면 새로운 객체를 간편하게 생성할 수 있다.

또한 virtual method를 이용해 실제 object의 타입에 관계없이 필요한 연산을 수행할 수 있게 하여 사용자의 부담을 덜었다.

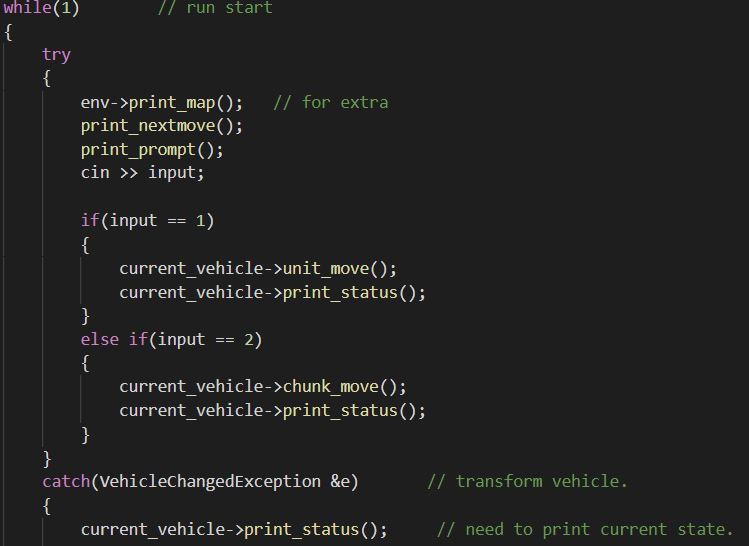
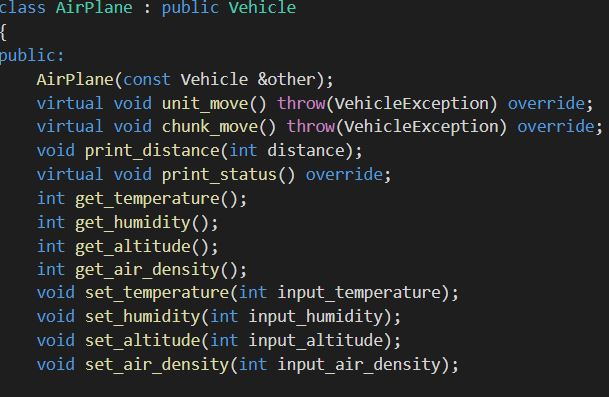
Vehicle이 진행하다가, 문제가 생기는 경우 이 문제를 어떻게 전달하고 나타내느냐 도 굉장히 고민을 많이 했었는데, vehicle이 움직인 후, 특정 값을 리턴 해서 그 값을 보고 문제가 일어났는지 안 일어났는지 판단하는 것은 너무나 비효율적인 작업이 될 것 같았다.

따라서 이를 해결하기 위해 수업시간에 배운 error handling 을 사용하였다.

사용자 exception class를 정의하고, 자동차 주행을 할 때마다 리턴 값을 받는 것이 아니라,

문제가 생겼을 때만 exception을 throw 하고, exception이 throw 된 경우 각 상황에 따라서

처리하게 하여 효율성을 높였다.



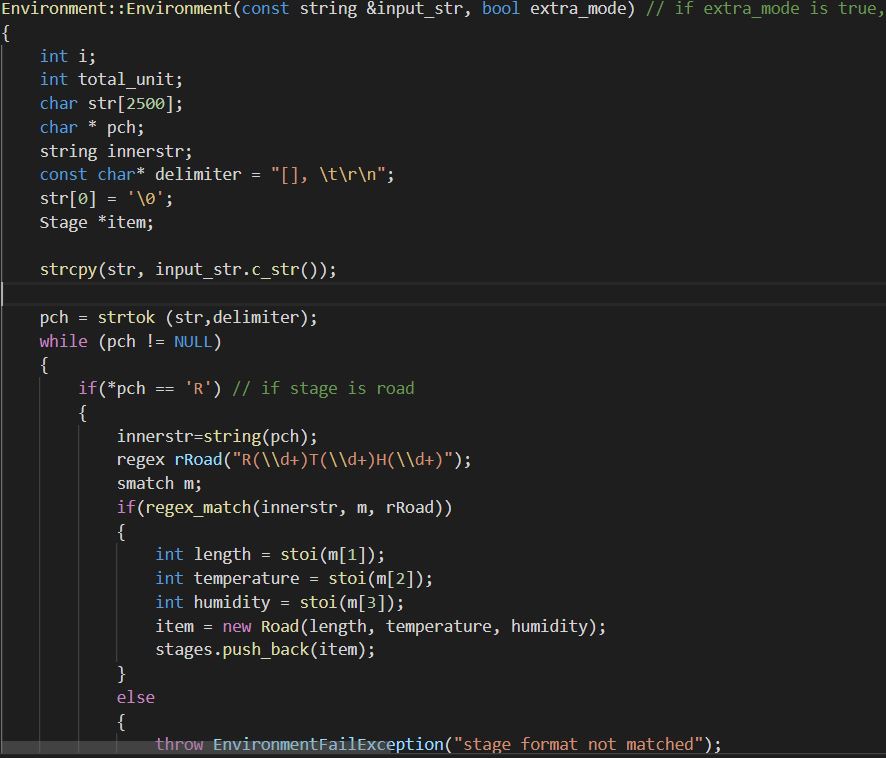
(airplane 클래스의 interface와, 메인 함수에서 try~catch를 통해 예외처리하는 코드)

프로그램에서 스트링의 데이터를 파싱 하고, 필요한 맵을 만드는 부분은

Environment 클래스 내부에서 strtok과 regex를 이용해 해결하였다.

다만 strtok의 경우, 원본 string의 상태를 바꿔버리는 문제가 있으므로

strtok에 string을 집어넣기 전, strcpy를 통해 새로운 스트링을 복사하는 방법을 사용하였다.

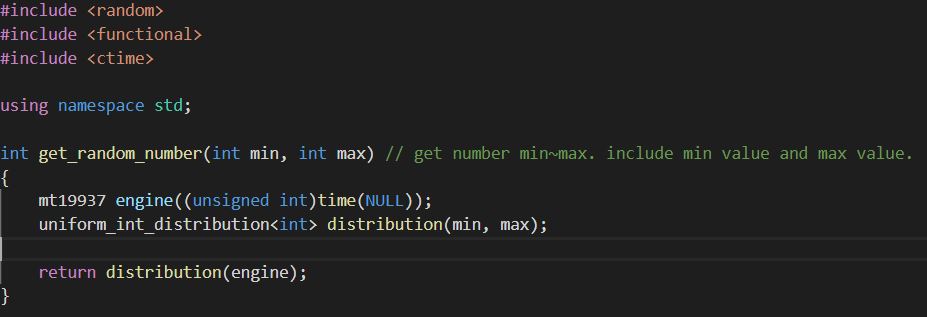


(string을 통해 environment를 생성하는 코드)

또한 X와 Y를 처리하는 부분에서 random하게 event를 진행해야 하는 부분이 있었는데,

이를 처리하기 위해 C++11 에서 추가된 random class를 사용하게 되었다.

Random class는 메르센-트위스터 엔진을 통해, 기존 C에 존재하던 rand와 srand를 이용한 난수생성보다 더욱 정교하게 난수 생성을 할 수 있다고 하여 사용하게 되었다.

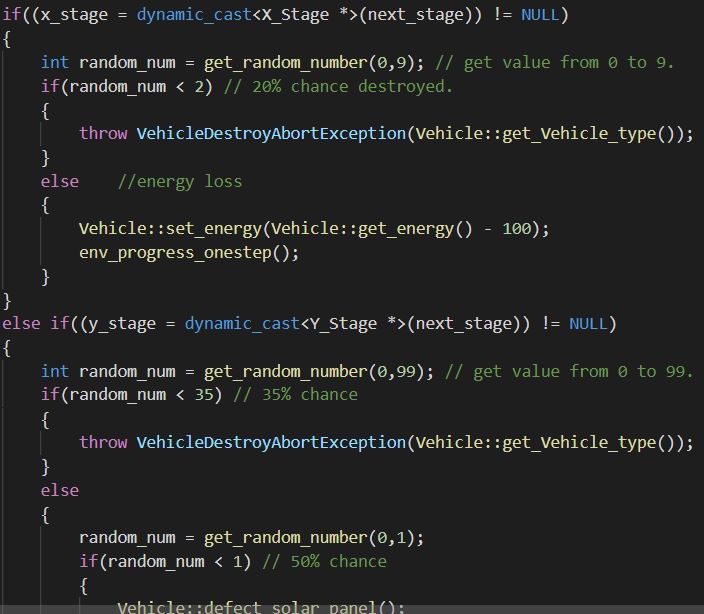


(min~max 값을 포함하는 integer random number를 만들어내는 함수. 각 number의 생성 확률은 같다.)

이때 프로그램을 실행할 때마다 이벤트가 일어나는 결과가 달라야 하므로,

random class의 엔진의 시드 값으로 현재 시간을 주어 실행할 때마다 다른 결과가 나오게 하였다.

이를 통해 X와 Y 이벤트를 처리할 때 특정 확률로 일어나는 이벤트를 처리할 수 있게 하였다.



(X와 Y를 확률적으로 처리하는 코드.)

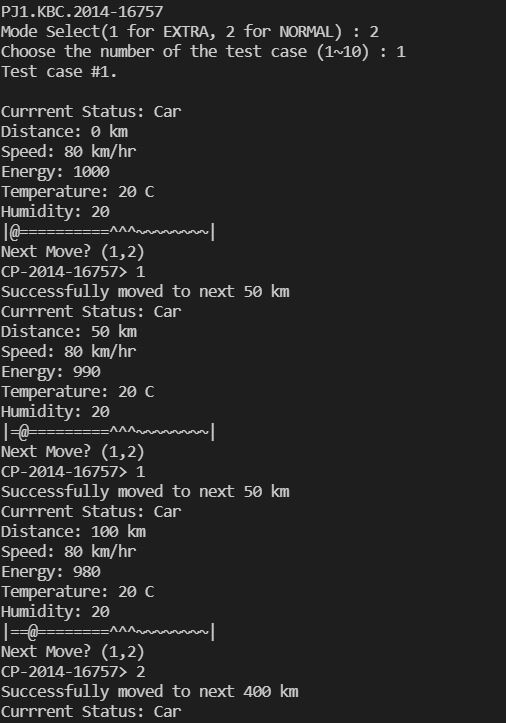
**Screenshot of the program working**

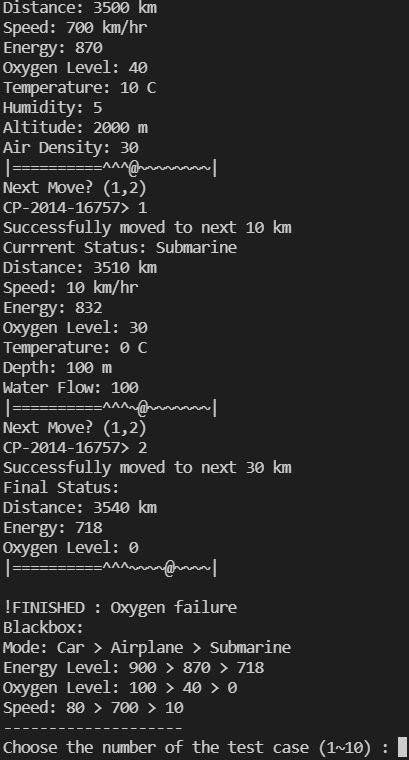
TC1 = "[R500T20H20],[X],[S3000T10H5A2000D30],[Y],[O80T0D100W100]"

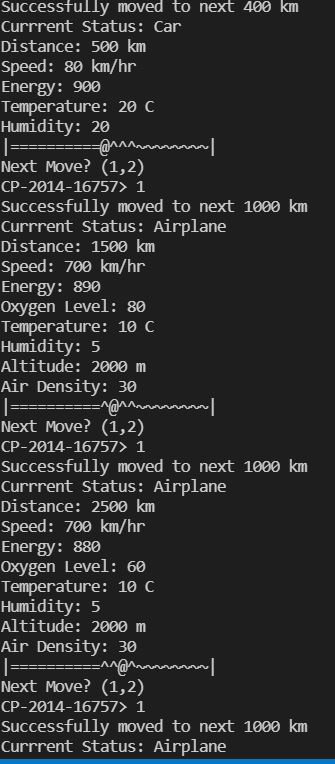
TC5= "[R500T20H20],[Y],[S1000T10H5A2000D30],[R500T20H20],[Y]

,[S1000T10H5A2000D30],[R5000T20H20]"

1. TC1, normal-mode

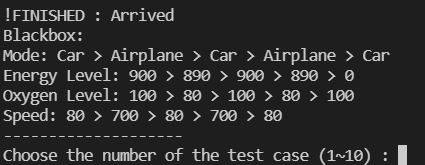
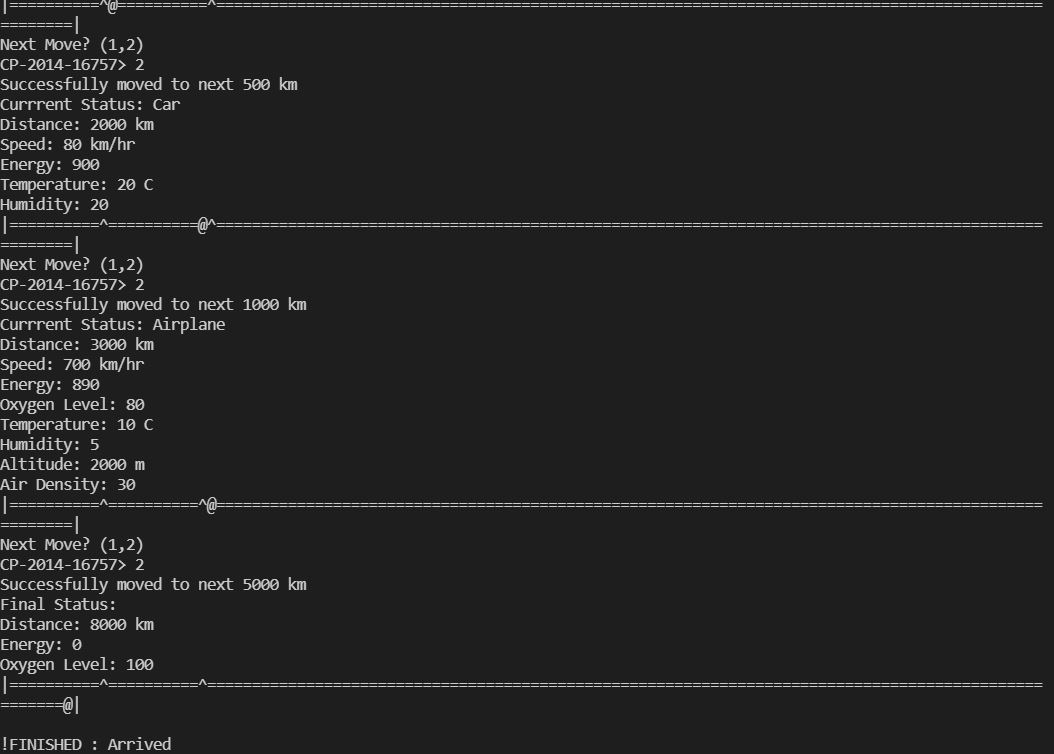
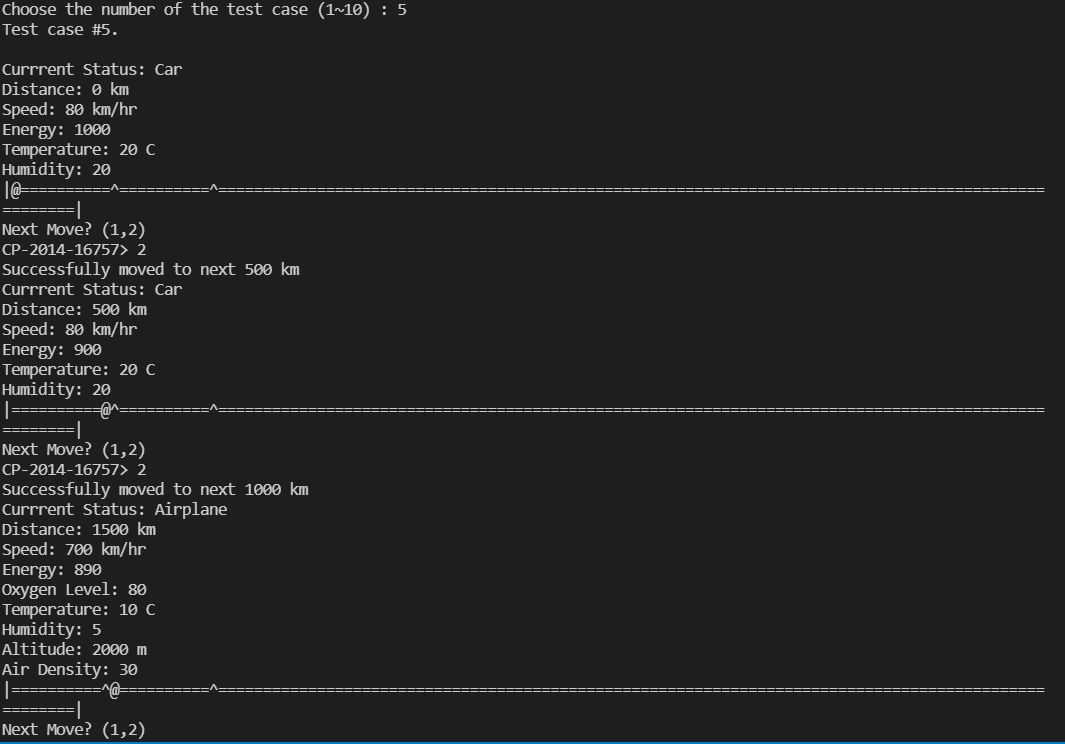






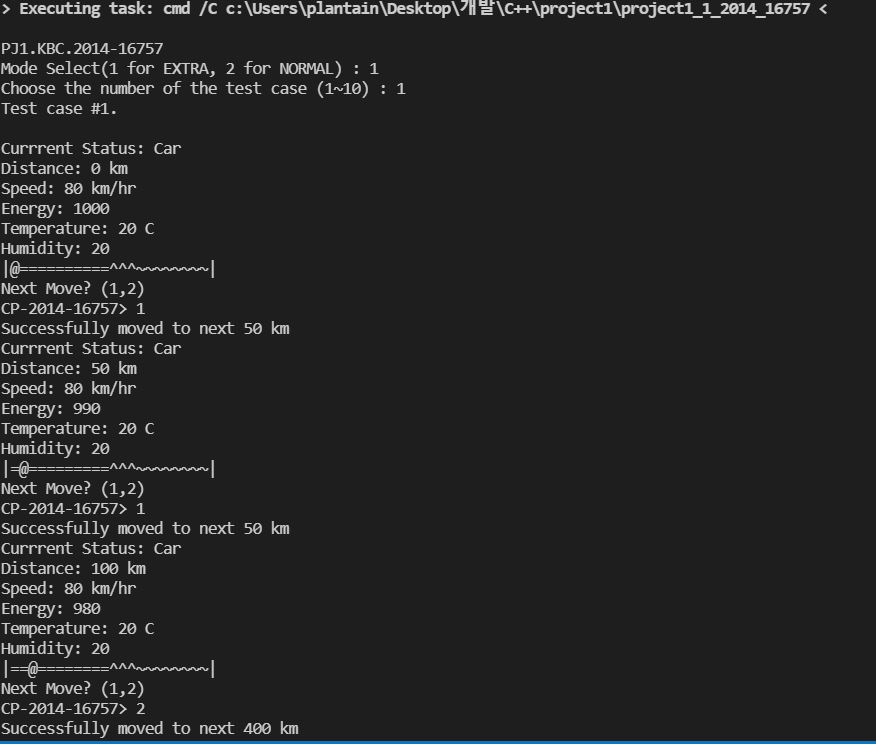
(PROJECT1 문서의 예시와 똑같이 끝난다.)

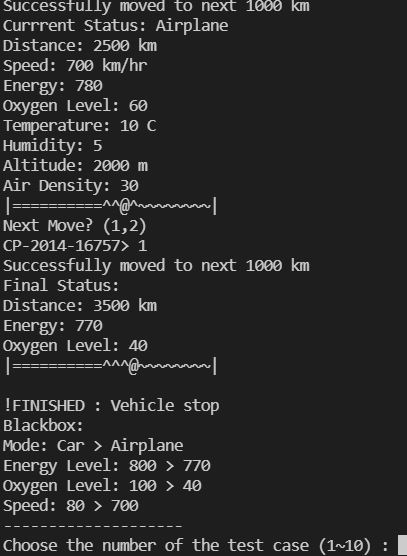
2. TC5, normal mode

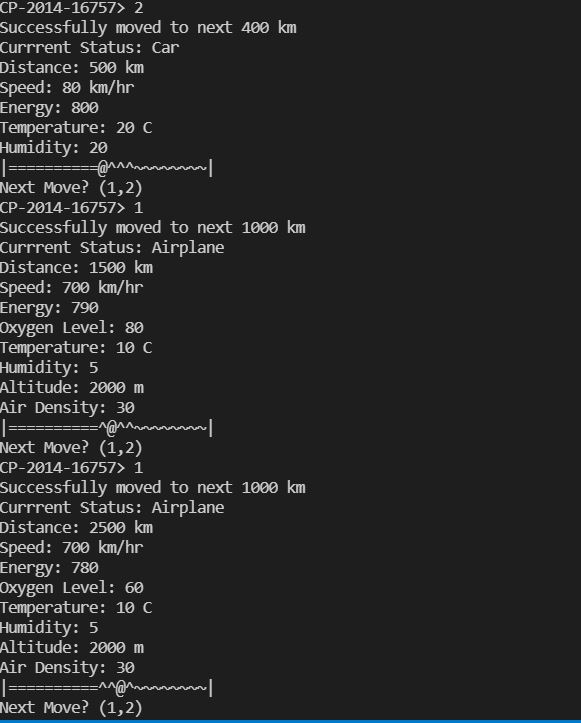


(arrived와 energy failure가 동시에 일어날 때, arrived가 우선적으로 판별된다.)

3. TC1, extra mode

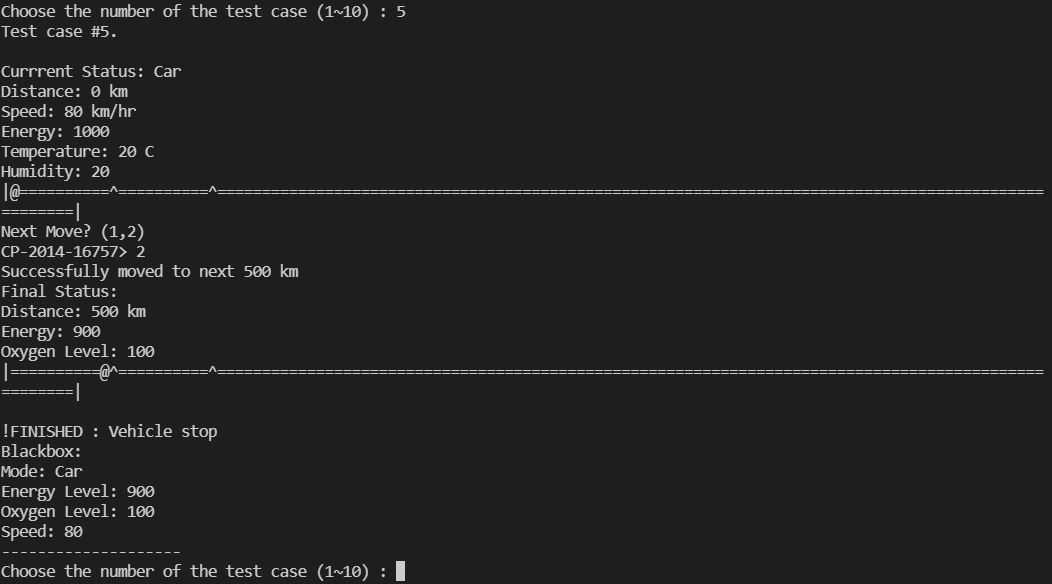






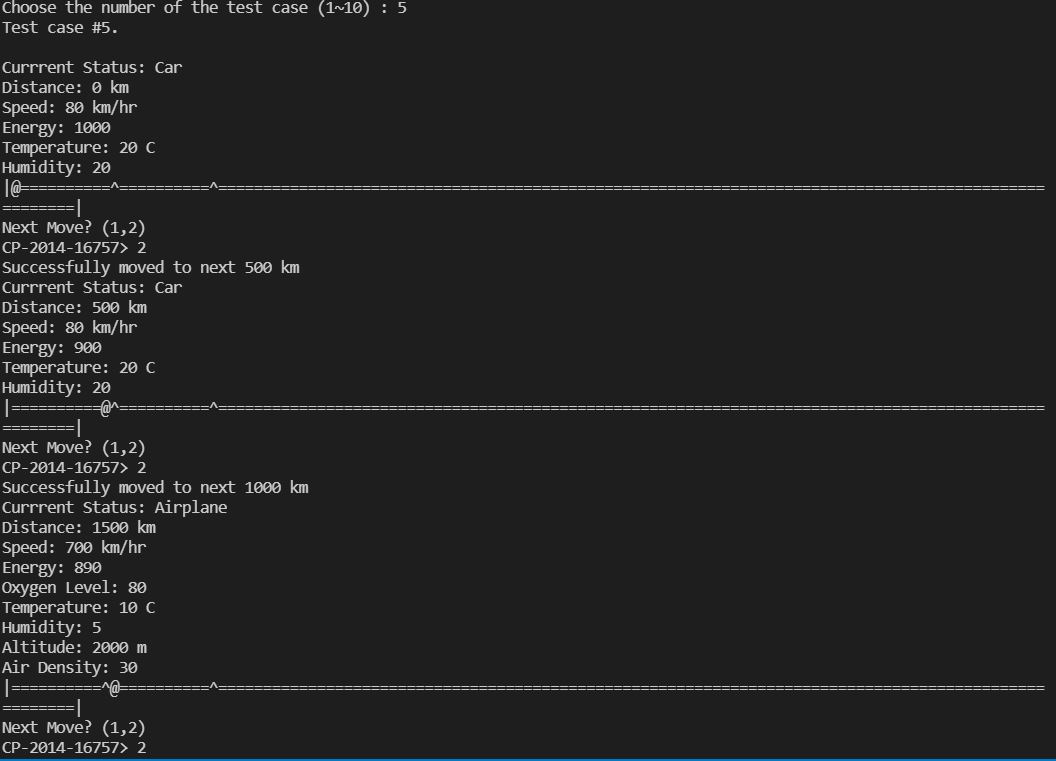
(event X때문에 energy가 100 감소하고, event Y때문에 차가 망가졌다.)

4. TC5, extra mode

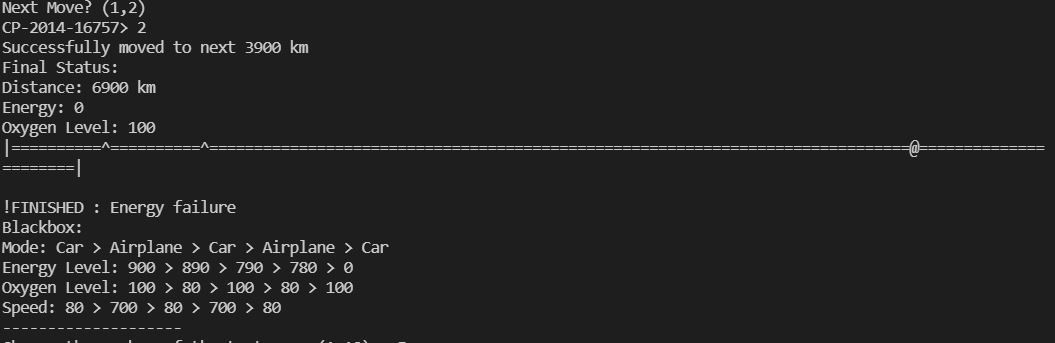


(event Y때문에 차가 망가졌다.)

5. TC5, extra mode another example



(event Y때문에 solar panel이 망가짐)



(solar panel이 망가진 것 때문에 원래대로라면 도착했을 거리도 도착을 하지 못했다.)