C++ language report

Deadline 2019

[201601639 컴퓨터공학부 홍승현] e-mail : cx\_nf@naver.com Contact : 010 – 4019 - 2514

목차

[클래스 다이어 그램 1](#_Toc24722286)

[Character.h 2](#_Toc24722287)

[protected 2](#_Toc24722288)

[int hp 2](#_Toc24722289)

[int mp 2](#_Toc24722290)

[int armor 2](#_Toc24722291)

[int damage 2](#_Toc24722292)

[솔루션 내 코드 2](#_Toc24722293)

[PUBLIC 2](#_Toc24722294)

[Character(const int hp, const int mp, const int armor, const int damage) : hp(hp), mp(mp), armor(armor), damage(damage) 2](#_Toc24722295)

[Virtual ~Character() 2](#_Toc24722296)

[bool HP\_isEmpty() const 2](#_Toc24722297)

[bool MP\_isUsable() const 3](#_Toc24722298)

[virtual void Attack(Character\* c) = 0; 3](#_Toc24722299)

[virtual void Skill(Character\* c) = 0; 3](#_Toc24722300)

[Get set 함수 3](#_Toc24722301)

[솔루션 내 코드 3](#_Toc24722302)

[정리 4](#_Toc24722303)

[Character 상속받는 Thives.h, Wizards.h, Warriors.h, Enemies.h 5](#_Toc24722304)

[public 5](#_Toc24722305)

[Thieves() : Character(120, 60, 7, 16) 또는 Warriors() : Character(150, 60, 10, 13) 또는 Wizards() : Character(120 80 7 10) 5](#_Toc24722306)

[~Thieves() 또는 ~Warriors() 또는 ~Wizards() 5](#_Toc24722307)

[void Attack(Character\* enemies) 또는 void Attack(Character\* main\_hero) 5](#_Toc24722308)

[void Skill(Character\* enemies) 또는 void Skill(Character\* trash) 5](#_Toc24722309)

[thieves.h 코드 6](#_Toc24722310)

[warriors.h 코드 7](#_Toc24722311)

[wizards.h코드 8](#_Toc24722312)

[Enemies.h 코드 9](#_Toc24722313)

[정리 9](#_Toc24722314)

[GameMain.h 10](#_Toc24722315)

[private 10](#_Toc24722316)

[int difficulty 10](#_Toc24722317)

[Character\* enemies 10](#_Toc24722318)

[Character\* main\_hero 10](#_Toc24722319)

[int Set\_Difficulty() 10](#_Toc24722320)

[int Set\_Job() 10](#_Toc24722321)

[int Set\_Menu() 10](#_Toc24722322)

[public 10](#_Toc24722323)

[GameMain() 10](#_Toc24722324)

[~GameMain() 11](#_Toc24722325)

[bool Error(const string& errorcode, const int size) 11](#_Toc24722326)

[void Menu() 11](#_Toc24722327)

[void Start() 11](#_Toc24722328)

[솔루션 내 코드 11](#_Toc24722329)

[정리 12](#_Toc24722330)

[GameMain.cpp 알고리즘, 코드 설명 13](#_Toc24722331)

[gamemain\_cpp 코드 13](#_Toc24722332)

[정리 14](#_Toc24722333)

[프로그램 간략도 15](#_Toc24722334)

[GameMain 코드 간략화 15](#_Toc24722335)

[게임시작(Start() 메소드) 코드 설명 및 간략화 16](#_Toc24722336)

[입력(Set\_Difficulty, Set\_Job, Set\_Menu) 코드 설명 및 간략화 16](#_Toc24722337)

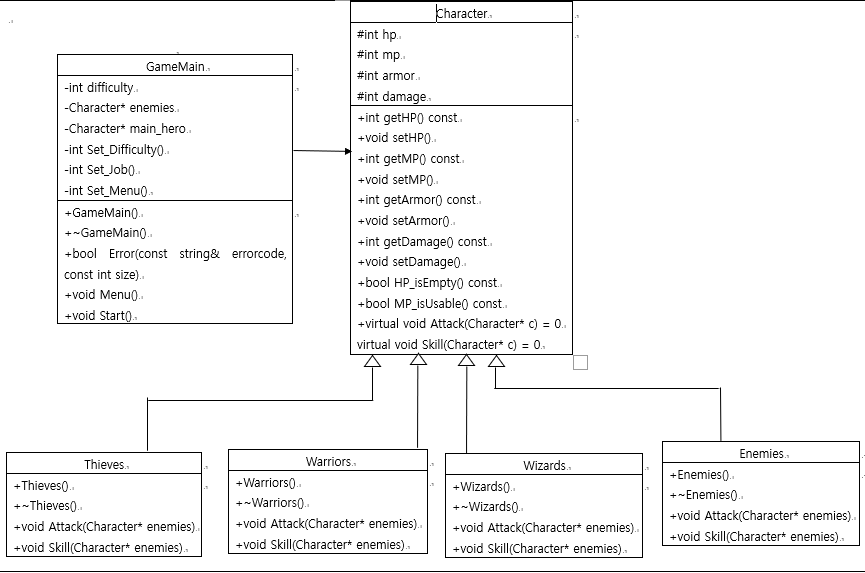
[올바른 입력인지 판별하는 오류판별(Error 메소드)코드 설명 및 간략화 17](#_Toc24722338)

[출력결과 18](#_Toc24722339)

[이 과제를 함으로써 느꼈던 점 20](#_Toc24722340)

[성장했다? 20](#_Toc24722341)

# 클래스 다이어 그램



# Character.h

## protected

### int hp

게임 내 체력을 담당하는 변수

### int mp

게임 내 마법력을 담당하는 변수, 스킬 메소드를 실행할 때 밀접한 관련이 있는 변수이다.

### int armor

게임 내 방어력을 담당하는 변수, 공격을 받았을 때 데미지에 따라 방어력 을 감소한 만큼의 데미지가 들어가게 된다.

### int damage

상대를 공격할 때 들어가는 데미지를 담당하는 변수이다.

### 솔루션 내 코드



## PUBLIC

### Character(const int hp, const int mp, const int armor, const int damage) : hp(hp), mp(mp), armor(armor), damage(damage)

해당 클래스의 생성자로, 파라미터로 hp, mp armor, damage를 받았을 때 이를 클래스 내의 private 변수에 할당하는 역할을 한다.

### Virtual ~Character()

해당 클래스의 소멸자로, 가상상속을 통해 동적 바인딩을 이용하였다. (가상상속 사용 안할 시, 업캐스팅할 때에 하나의 소멸자밖에 실행이 되지 않음)

### bool HP\_isEmpty() const

체력이 없는지를 나타내주는 메소드이다. 게임 실행시 적을 죽이거나 유저가 죽을 때 이 메소드를 통해 승리와 패배를 나눈다.

### bool MP\_isUsable() const

MP가 사용 가능한지를 판단해주는 메소드이다. 유저가 스킬을 사용할 때 사용 여부를 판단하며, 사용이 불가능할 경우 턴을 넘기고 사용이 가능할 경우 스킬을 사용한다.

### virtual void Attack(Character\* c) = 0;

일반공격을 통해 상대에게 데미지를 주게 하는 메소드이다.  
메소드 실행시 상대의 hp가자신의 일반공격 - 상대의 방어력 만큼 깎인다.  
순수가상상속을 통해 선언하였으며, 파생클래스의 재정의에 따라 데미지가 달라지게 된다.

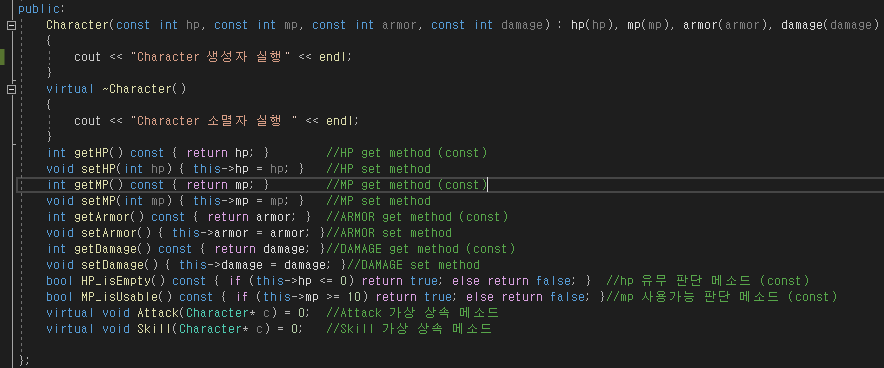
### virtual void Skill(Character\* c) = 0;

일반공격이 아닌 스킬을 사용하는 특수공격으로 상대에게 데미지를 주는 메소드이다.  
일반 공격과 마찬가지로 메소드 실행시 상대의 hp가 자신의 Skill 데미지 – 상대의 방어력 만큼 깎인다.  
순수가상상속을 통해 선언하였으며, 파생클래스의 재정의에 따라 데미지가 달라지게 된다.

### Get set 함수

getHP, setHP, getMP, setMP, getArmor, setArmor getDamage, setDamage 가 있으며 일반적인 get, set 함수이다. 이는 해당 클래스의 hp, mp, armor 그리고 damage를 설정해주는 메소드이다.

### 솔루션 내 코드



## 정리

상속을 목적으로 만든 추상클래스인Character 이다.  
유저가 직업을 선택할 전사, 마법사, 도적 그리고 적들이 이 클래스를 상속받고 있으며, 순수가상함수를 오버라이드 하였다. 이는 다음 코드에서 보여질 예정이다.

그렇기에 get, set 함수들과 HP가 있는지 없는지 판별하는 HP\_isEmpty 메소드, mp가 사용 가능한지 판단해주는 MP\_isUsable 메소드가 있고, Skill 뿐만 아닌 공통의 공격인 일반공격(Attack)도 순수가상함수로 선언해 놓았다.

# Character 상속받는 Thives.h, Wizards.h, Warriors.h, Enemies.h

## public

### Thieves() : Character(120, 60, 7, 16) 또는 Warriors() : Character(150, 60, 10, 13) 또는 Wizards() : Character(120 80 7 10)

해당 클래스의 생성자로, 생성자 실행 전 부모클래스의 파라미터에 (HP, MP, ARMOR, DAMAGE)값을 넣어 할당시킨 후에 생성자를 실행시킨다. 부모클래스는 해당 파라미터를 통해 각 protected내에 있는 변수를 할당시킨다.

### ~Thieves() 또는 ~Warriors() 또는 ~Wizards()

해당 클래스의 소멸자이다. 소멸자 실행 후에 부모클래스의 소멸자도 실행된다.

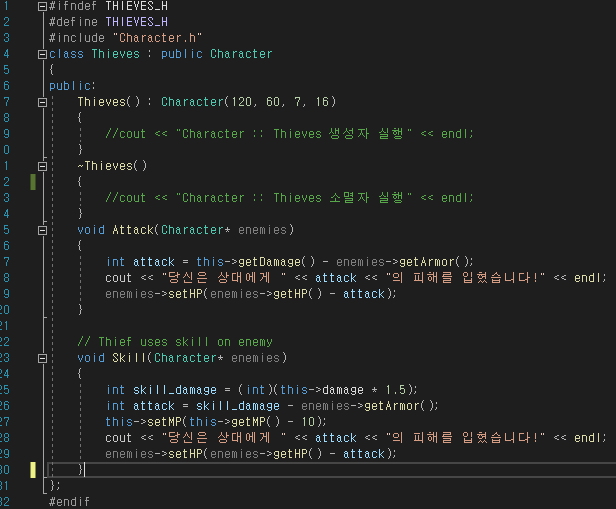
### void Attack(Character\* enemies) 또는 void Attack(Character\* main\_hero)

해당 클래스의 공격 메소드이며, 부모클래스의 메소드를 오버라이딩한 메소드이다. 각 직업군에 따라 일반 공격의 데미지가 다르며, 상대를 파라미터로 입력받아 상대의 피를 해당 클래스의 공격데미지에 상대의 방어력을 뺀 만큼의 데미지로 설정한다.

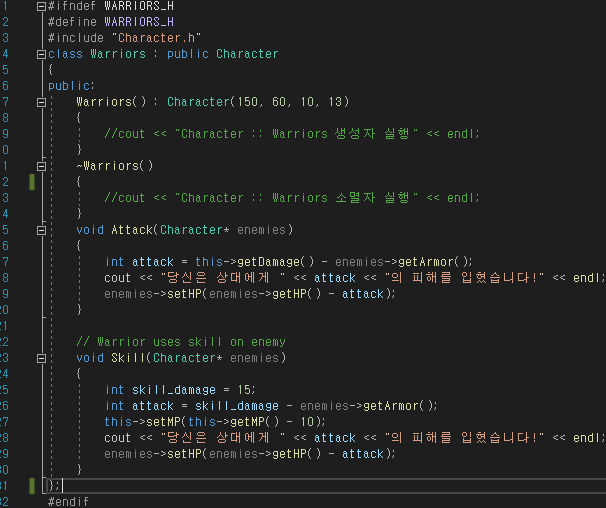
### void Skill(Character\* enemies) 또는 void Skill(Character\* trash)

해당 클래스의 특수공격인 스킬 메소드이며, 부모클래의 메소드를 오버라이딩한 메소드이다.  
Attack 메소드와 비슷한 코드 구성이나, 추가적인 코드로는 자신의 MP에 10을 빼는 코드 뿐이다.  
그리고 enemies 클래스에서는 Skill 사용을 하지 않기에 파라미터 역시 trash라는 작명으로 지었고 메소드 내에서도 아무런 구현도 하지 않았다.

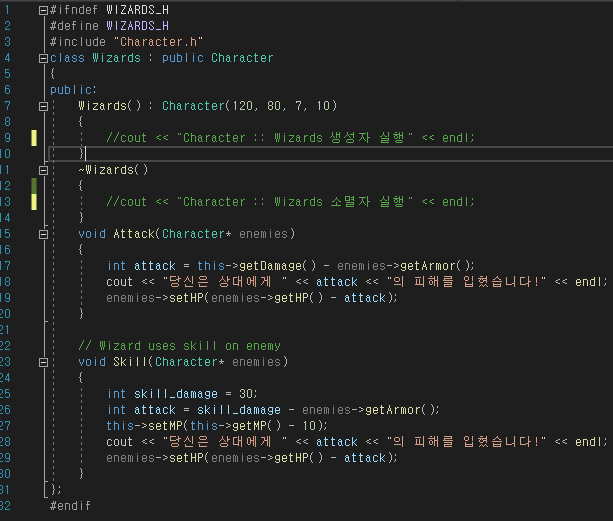
## thieves.h 코드



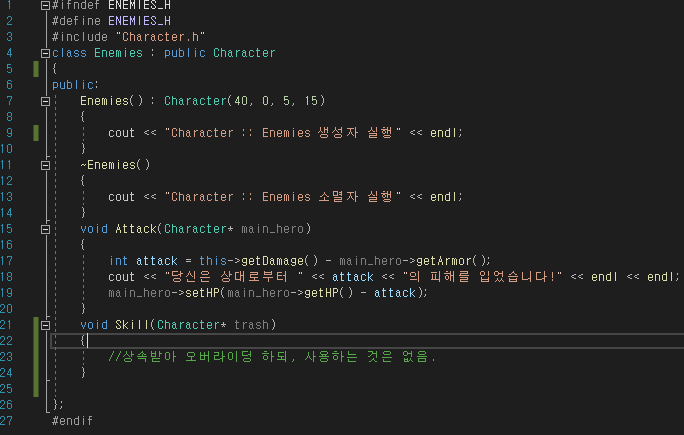
## warriors.h 코드



## wizards.h코드



## Enemies.h 코드



-

## 정리

Warriors, Wizards, Thieves, Enemies는 모두 Character를 상속받고 과제 조건표에 있는 기술표에 맞게끔 생성자에서 입력받고 있으나, Enemies 클래스만 Skill 상속에 대해 아무런 조건 및 설정이 없게 구현하였고, 다른 클래스들만 Skill에 대한 여러 조건들이 나열되어 있다. 이는 과제에 기재되어있다시피 적 클래스는 스킬을 사용 안하게끔 하기 위해서다.

# GameMain.h

## private

### int difficulty

난이도를 설정할 때 저장할 변수이다. 이 변수에 값에 따라 적의 수가 차이가 나게 된다.

### Character\* enemies

게임 내 적들을 구현시키기 위한 Character 포인터형 객체, 난이도 설정에 따라 그 수에서 2만큼 더한 값을 동적할당한다.

### Character\* main\_hero

게임 내 유저가 사용할 Character 포인터형 객체이다. 전사, 마법사, 도적을 선택할 시 그에 따라 업캐스팅 되어 게임에 사용된다.

### int Set\_Difficulty()

난이도를 설정하는 메소드이다. 입력된 값을 difficulty 변수에 넣어주게 된다.

### int Set\_Job()

유저의 직업을 설정하게끔 해주는 메소드이다. 3가지의 직업을 선택 후에 적용시킬 수 있다.

### int Set\_Menu()

전체 메뉴 설정을 입력받는 메소드이다.

1. 난이도설정
2. 직업설정
3. 게임시작

과 같은 입력을 받는다.

## public

### GameMain()

해당 클래스의 생성자로써, Menu() 메소드를 실행시킨다.

### ~GameMain()

해당 클래스의 소멸자이며, 자신의 변수인 enemies와 main\_hero의 동적할당이 해제가 되어있지 않은 경우에, 동적할당을 해제시킨다.

### bool Error(const string& errorcode, const int size)

입력받는 부분의 오류를 검출해주는 메소드로, 에러검출시 true를, 에러미검출시 false를 반환한다. 자세한 설명은 후에 나열되어있다.

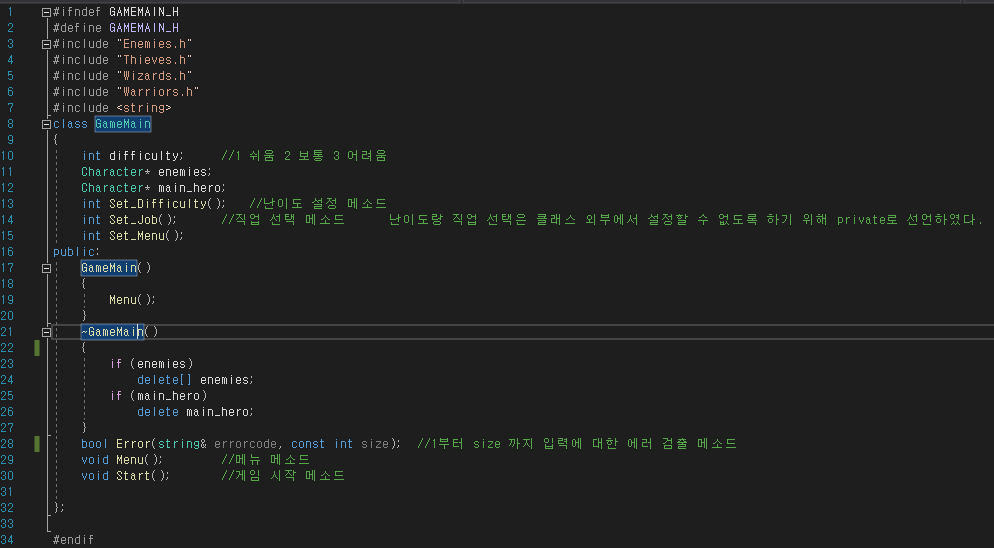
### void Menu()

전체적인 메뉴를 출력과 입력을 도맡아 한다. 이 메소드를 통해 입력을 받고 난이도 설정을 할 것인지, 직업을 설정할 것인지 게임을 시작할 것인지를 정한다.

### void Start()

게임을 시작하는 메소드이다. 이 메소드를 통해서 유저와 적의 공격을 주고받게 되며 일반공격과 스킬공격을 선택하게 된다.  
서로의 공격을 주고 받는 와중, 상대가 모두 죽으면 게임을 승리하게 되고 유저가 죽을경우 게임에서 패배하게 된 후 메소드가 종료된다.

## 솔루션 내 코드

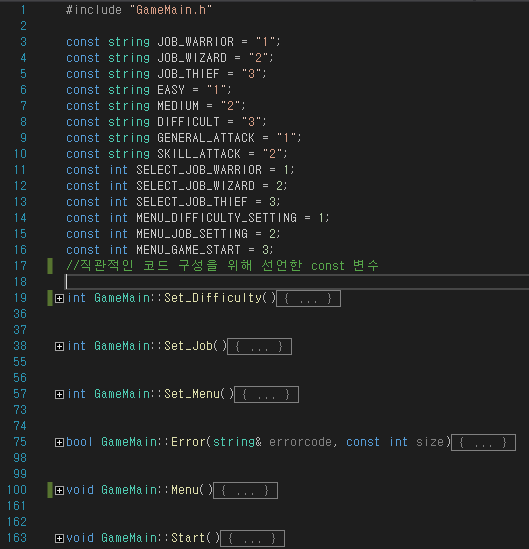


## 정리

모든 코드중에 핵심을 다루는 클래스로 게임에 필요한 여러 요소들을 넣었다(난이도, 난이도에 따라 달라지는 적을 만들 캐릭터 클래스의 포인터형 변수(enemies), 유저의 직업을 가리는 main\_hero, 그리고 이를 설정(Set) 시키는 메소드를 PRIVATE에 넣었다.), (생성자와 소멸자, 입력 에러 검출 메소드, 시작메뉴 출력과 입력 메소드, 게임 시작 메소드를 PUBLIC에 넣었다.)

# GameMain.cpp 알고리즘, 코드 설명

## gamemain\_cpp 코드



## 정리

프로그램을 실행할 시 메뉴에서 GameMain 인스턴스 객체를 선언한다.

선언 시 생성자가 실행 될 때 Menu() 메소드를 실행시킨다.

Menu메소드에서는 모든 메뉴를 볼 수 있으며, 입출력을 도맡고(입력은 Set\_Menu를 통해 입력받음) 설정이 완료되었는지 판별한다.

1. 난이도 설정(Set\_Difficulty)
2. 직업 설정(Set\_Job)
3. 게임 시작(Start)

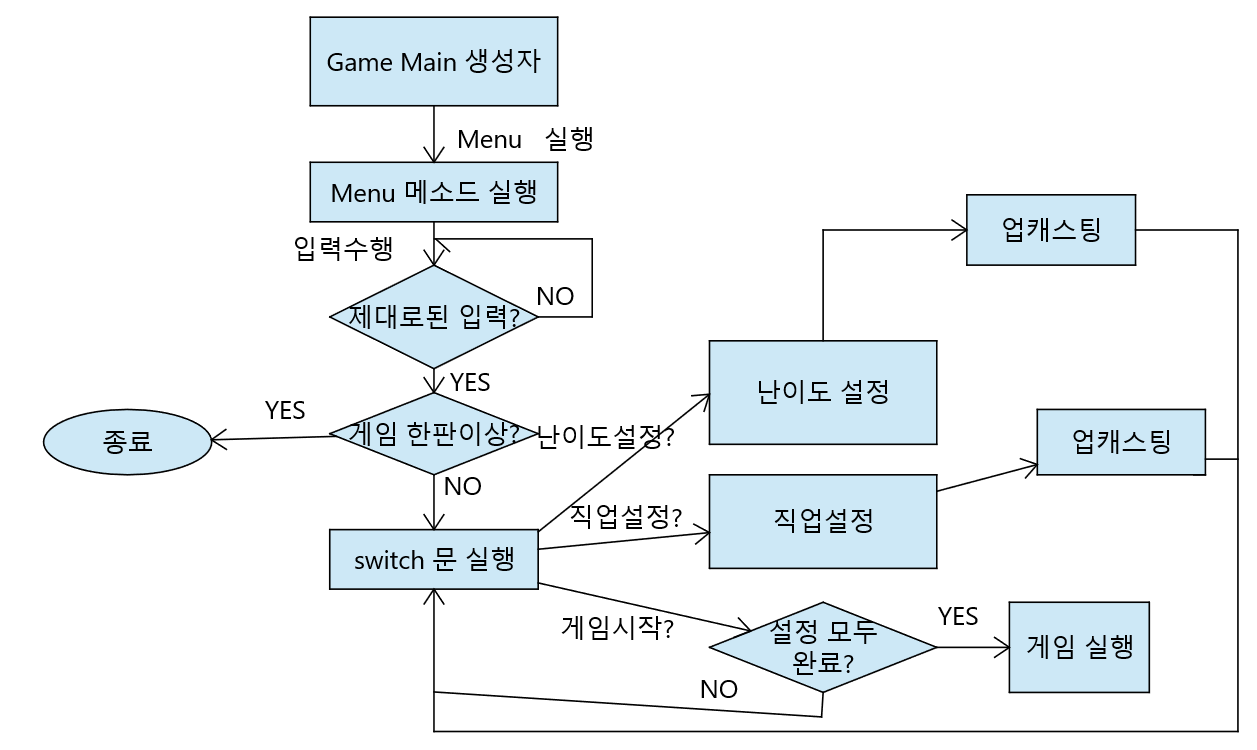
이 세개의 입력을 받게 되는데 문자나 범위 외의 숫자를 입력할 경우 Error 메소드를 통해 판별하여 오류 메시지를 출력하게 되고 제대로 된 입력을 받을 때까지 계속해서 입력을 받게 된다.

이 중에서 게임시작을 입력할 시, 난이도와 직업 설정이 되어있는지 판별 후에 게임을 시작하게 된다.

게임을 시작하고나서 끝났을 경우 Menu메소드를 나오게 된 후 생성자 실행이 끝나게 된다. 즉, 게임을 실행하려면 GameMain의 인스턴스 객체만 만들어주면 알아서 게임 프로그램이 실행된다.

## 프로그램 간략도

### GameMain 코드 간략화

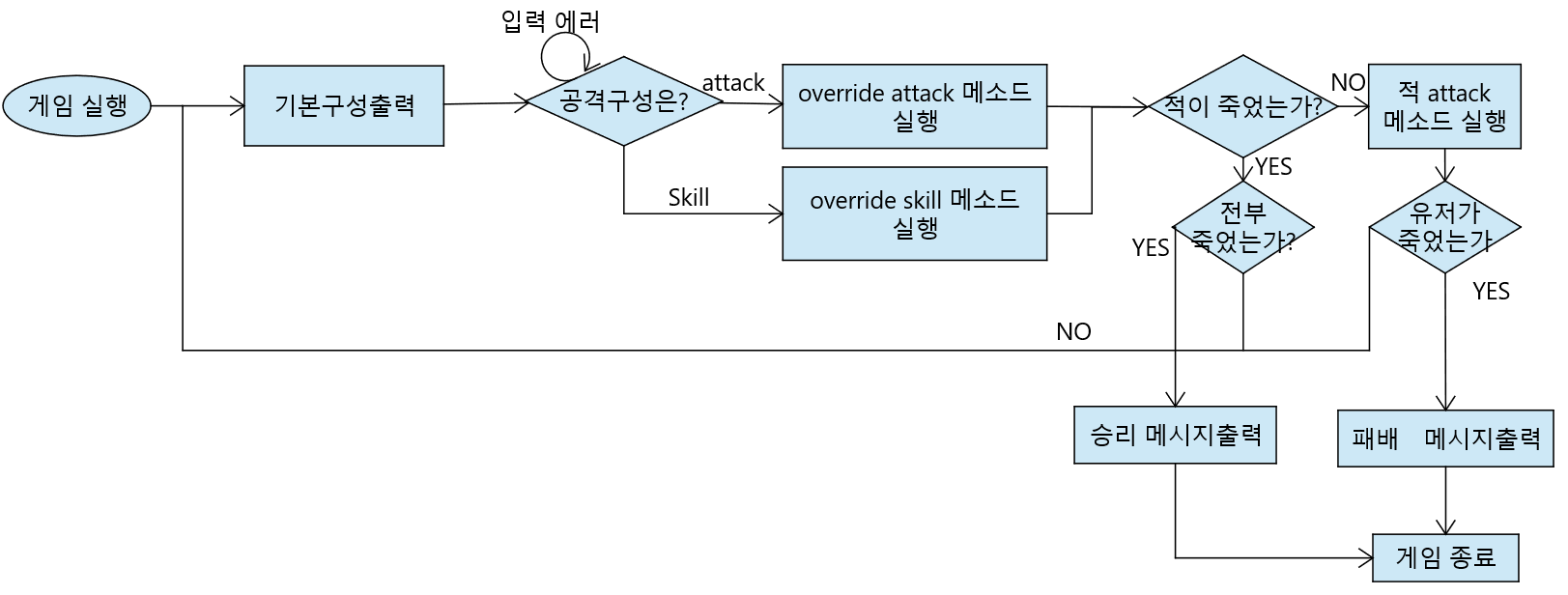


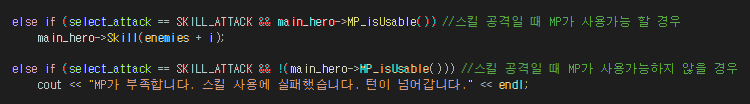
전반적인 코드 실행 부분이다.

본 설계도는 현 실행되는 프로젝트 코드들을 간략화하여 보기 쉽게 그려놓았다.

P. S) 필자가 난이도설정->업캐스팅 부분에서 입력에러 판별 구문을 넣는다는 것을 깜빡했다. ( 이는 코드 상에서 구현되어있다.)

### 게임시작(Start() 메소드) 코드 설명 및 간략화



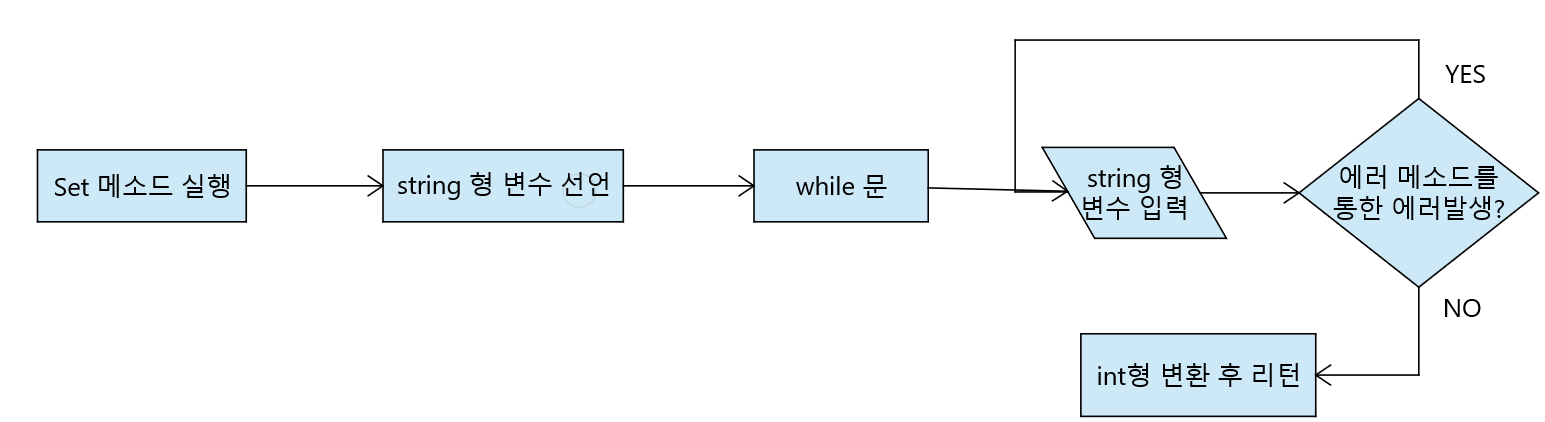
위 표에서 Skill에 대해 자세히 설명을 하자면

Skill공격을 선택(입력)했을 경우 두 가지의 조건으로 나뉜다.

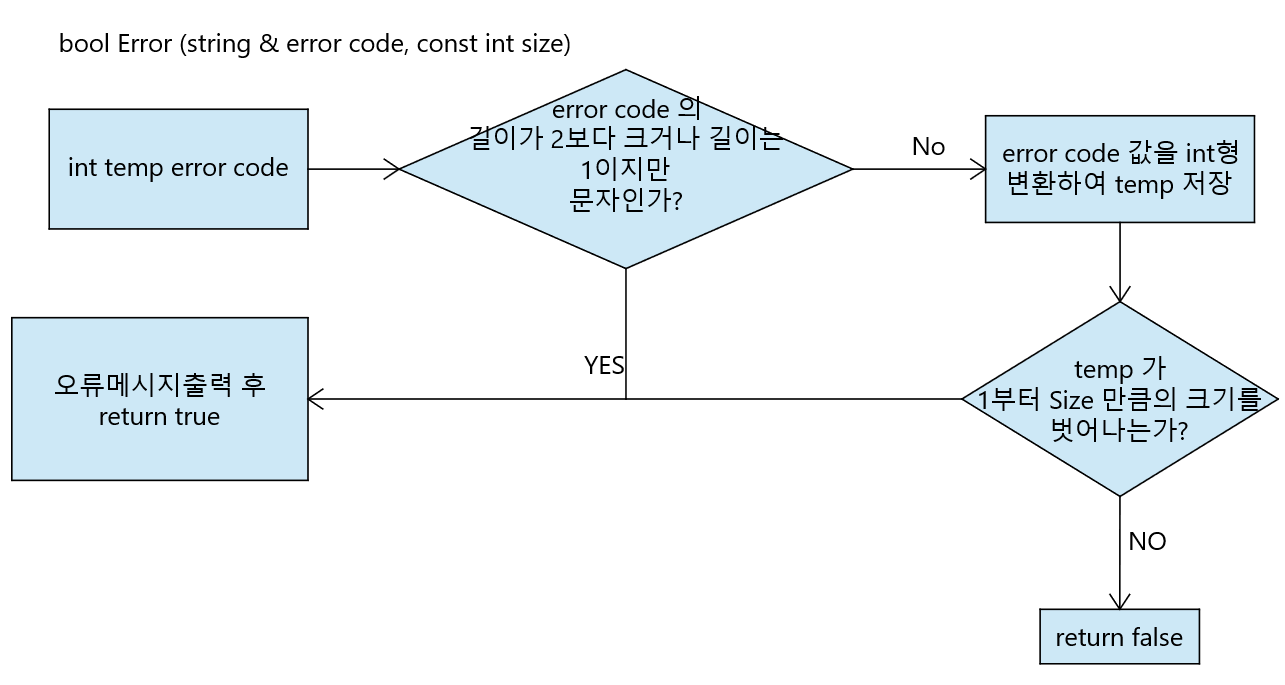
1. 스킬을 사용할 MP가 충분히 있는가
2. 스킬을 사용할 MP가 충분치 못한가

이를 통해 MP가 사용가능하면 Skill메소드를 실행, 스킬 데미지를 적에게 주어 공격하며, MP가 사용불가능일 경우에는 사용실패 메시지와 함께 아무런 행동도 취하지 않은채로 다음 턴으로 넘어간다.

### 입력(Set\_Difficulty, Set\_Job, Set\_Menu) 코드 설명 및 간략화

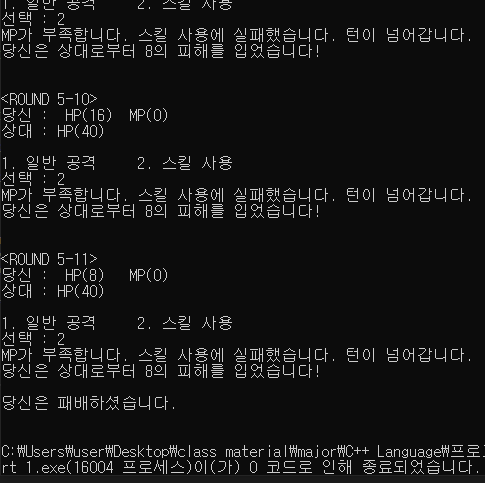
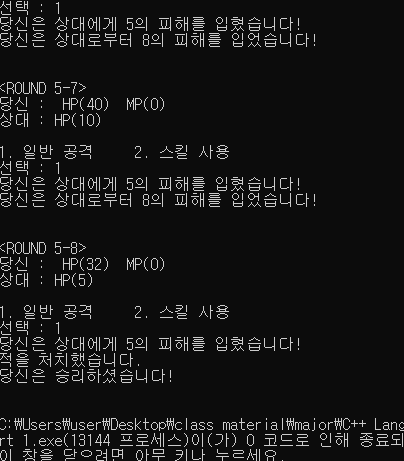
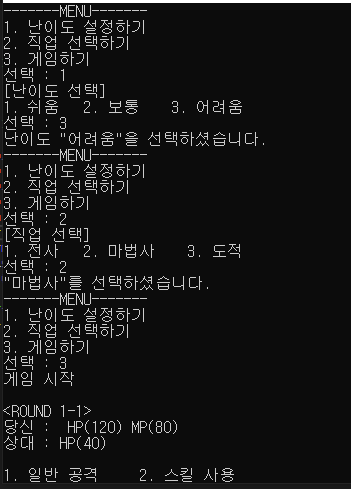


### 올바른 입력인지 판별하는 오류판별(Error 메소드)코드 설명 및 간략화



위 메소드를 통해 지정되지 않은 숫자나 문자, 숫자와 문자가 섞인 문자 모두 걸러낼 수 있다. Error이면 true를 반환하고 Error가 발생되지 않으면 False를 반환하기 때문에 모든 Error메소드를 실행할 때 앞에 괄호() 후 !를 하여 조건이 참이면 반복문을 나가는 식으로 구현해 놓았다.

## 출력결과



## 이 과제를 함으로써 느꼈던 점

### 성장했다?

1. 첫 과제(경마게임)를 할 때 난관에 빠져 코드를 못 짠던 적이 있었다. Int형을 입력받을 때 문자를 사용하면 무한반복이 되었기 때문이였다. 이는 어느정도 구글링 + 조교님의 도움으로 해결되었으나 여전히 해결 안된 부분이 “3abc”와 같이 숫자+ 문자로 입력을 받을 때였다.  
   이는 첫 입력을 3으로 받아 뒤의 abc 문자로 인한 오류 출력과 동시에 숫자 입력을 받았다고 인식하여 반복문을 탈출하여 넘어간 것이다.  
   이를 이번 과제 때 string과 형변환을 사용, 해결하여 한껏 성장한 느낌이 들었다.
2. 던전 프로그래밍을 할 때 유저가 선공, 적이 후공을 하는 방식으로 보았다. 하지만 결과예시에서는 동시공격이였고, 출력문으로써는 유저가 선공인 것 처럼 보이기에 유저가 선공으로 적을 죽일 시 적은 유저를 공격하지 못하고 다음 라운드로 넘어가게끔 코딩해보았다.  
   두 가지 예를 다 해보았는데 역시 현실적으로 선공후공이 나은 듯 하다..^^..  
   예전에는 이러한 코드를 입맛대로 바꾸는 것이 쉽지 않았는데 되돌아보면 참 신기하다는 마음이 든다. 불과 3~4달 전 까지만 해도 C언어 기초공부하면서 뺀질했던 기억이 나는데 마음 다 잡고 지금까지 온 나에게 참으로 자랑스럽다.

프로젝트 1(과제 1) 보고서 종료

작성자 : 201601639 홍승현