DAY2
과제 3번 보고서

[매력적인 요약문으로 독자의 시선을 끌어 보세요. 일반적으로 요약은 문서의 내용을 간략하게 정리한 것입니다. 내용을 추가하려면 여기를 클릭하여 입력하면 됩니다.]

2023741024
로봇학부 박건후

목차

(1) 헤더 파일

```
#include <iostream>
#include <ctime>
namespace hw3{
extern int end_flag;
int absolute(int n);
class Monster{

    public:
    int HP;
    int x;
    int y;
    Monster();
    Monster(int x, int y, int HP);
    int Be_Attacked();

};
```

이번에는 hw3네임스페이스를 만들고 그 안에 선언을 했습니다. 전역 변수인 end_flag는 몬스터의 HP가 0이 되었을 때, 플레이어의 MP가 0이 되었을 때 프로그램을 종료하게 되는데 그 때의 플래그가 되는 변수입니다.

absolute함수는 정수값을 받아 해당 양수면 n값 리턴, 음수면 -n을 리턴하는 함수로써 절댓값을 구할 때 사용하기 위해 선언했습니다.

몬스터 클래스는 문제에서 요구하는대로 멤버변수, 멤버함수, 생성자, 소멸자를 선언하였습니다.

```
class Player{


    public:
    int MP;
    int x;
    int y;
    int HP;
    Player();
    Player(int x, int y);
    void X_move(int move);
    void Y_move(int move);
    void Attack(Monster& target);
    void Show_status();
};
}
```

Player 구조체 또한 문제에서 요구한 대로 선언하고 그에 맞게 구현하여 문제를 풀었습니다.

(2) 구현 파일

```
#include "hw3.hpp"
using namespace hw3;
namespace hw3{
    int end_flag=1;
    int absolute(int n){
     if(n>=0) return n;
     else if(n<0) return -n;
     else return 0;
    }
}
```

선언한 end_flag를 1로 초기화를 하고, absolute함수를 hw3 공간에서 구현하여 선언에서부터 구현을 찾을 때 충돌나지 않기 위해 h23에 선언하였습니다.

```
Monster::Monster():x(0), y(0), HP(50){};
Monster::Monster(int x, int y, int HP): x(x), y(y), HP(HP){};
int Monster:: Be_Attacked(){
    HP-=10;
    y++; //맞으면 뒤로 물러남
    return HP;
}
```

몬스터 클래스의 멤버함수를 구현한 부분입니다. 기본 생성자는 기본대로 초기화를 하고, 매개변수가 있는 생성자는 매개변수에 맞게 값을 초기화하는 형식이므로 멤버 초기화리스트를 통해 그대로 초기화하였습니다.

Be_Attacked메서드에서는 몬스터가 공격을 받은 것에 대한 동작을 해야 하므로 몬스터 내부에서 변화시킬 수 있는 것들을 작성하려고 했습니다. 공격당하면 10씩 HP가 줄어들게 하였고 공격을 당했을 때 같은 자리에만 있으면 사용자가 계속 A를 눌렀을 때 계속 맞게 되므로 공격받으면 좀 뒤로 물러나는 점을 본떠서 몬스터의 위치를 옮기도록 했습니다.

```cpp
Player::Player(){x=0;  y=0; HP=30; MP=10;};
Player::Player(int x, int y):x(x), y(y), HP(30), MP(10){};
void Player::X_move(int move){
    switch(move){
        case 1:
        x--;
        std::cout<<"X Position -1 moved!"<<std::endl;
        break;
        case 0:
        x++;
        std::cout<<"X Position 1 moved!"<<std::endl;
        break;
    }
}
void Player::Y_move(int move){
    switch(move){
        case 1:
        y++;
        std::cout<<"Y Position 1 moved!"<<std::endl;
        break;
        case 0:
        y--;
        std::cout<<"Y Position -1 moved!"<<std::endl;
        break;
    }
}
```

이제 플레이어의 메서드를 구현하는 부분입니다. X_move, Y_move 구현 부분인데, 매개변수 값을 통해 1, 0에 대해 각각 동작을 달리하게 하여 플레이어의 위치를 변화시켰습니다.

```
void Player::Attack(Monster& target){
    if(absolute(target.x-x)<=1 && absolute(target.y-y)<=1){ //좌표상 거리가 1이하일 
        int monster_HP = target.Be_Attacked();
        MP--;
        std::cout<<"공격 성공!"<<std::endl<<"남은 체력: "<< monster_HP<<std::endl;
        if(monster_HP==0){
            std::cout<<"Monster_die!"<<std::endl;
            end_flag=0;
        }
    }
    else{
        std::cout<< "공격 실패!"<<std::endl;
        MP--;
    }
    if(MP==0){
        std::cout<<"MP 부족!"<<std::endl;
        end_flag=0;
    }
}
void Player:: Show_status(){
    std::cout<< "HP: "<< HP<<std::endl<<"MP: "<<MP<<std::endl<<"Position: "<<x<<
}
```

이제 플레이어가 공격을 하는 부분입니다. 공격이 먹히는 것에 대한 기준을 정할 때 우선 몬스터가 가까이에 있어야 하므로 몬스터로부터 x, y좌표상 각각 최대 1만큼 떨어진 거리 안에 있을 때 공격이 몬스터에게 먹히는 것으로 만들었습니다. 이 때 두 객체의 위치값의 차를 구하기 위해 absolute함수를 사용하였습니다. 공격이 먹혔을 경우에는 몬스터의 체력을 깍는 것을 target레퍼런스 변수를 통해 메서드를 호출하여 구현하였고 Be_Attacked가 int를 반환하는 함수이기에 HP값을 반환하게 하여 그 값으로 0일 경우 게임을 종료하도록 쉽게 만들기 위해 HP를 반환하도록 한 것입니다. 그리고 MP가 0이 되면 그 때고 게임이 종료되므로 마지막에 if문을 걸었습니다.

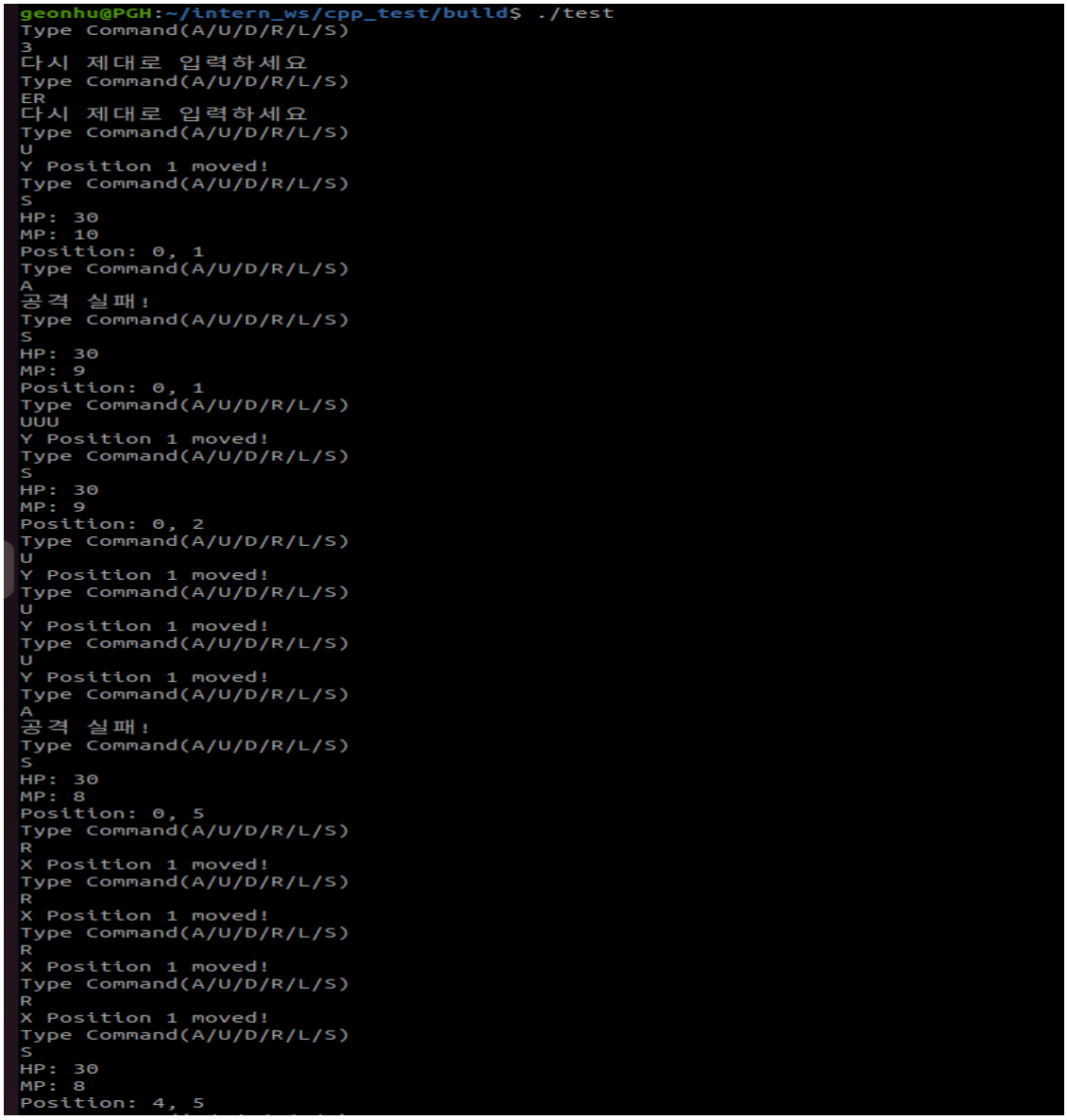
Show_status메서드는 과제의 예시에 맞춰 플레이어의 정보를 출력하게 했습니다.

(3) main 파일

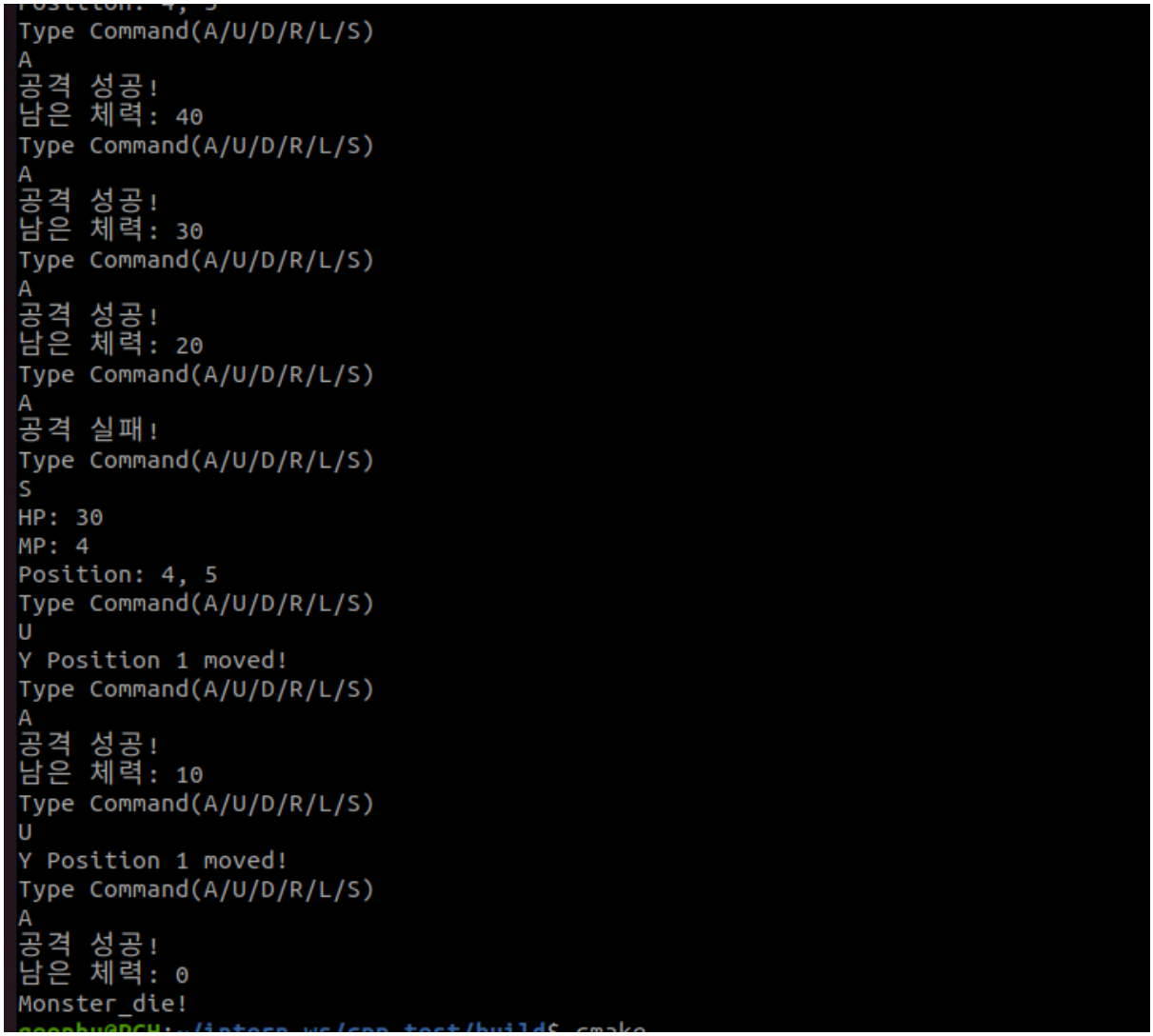
```
int main(){
    srand((unsigned)time(NULL));
    Player player(0,0);
    Monster monster(5,4,50);
    char ch=0;
    while(end_flag){
        std::cout<<"Type Command(A/U/D/R/L/S)"<<std::endl;
        std::cin>> ch;
        if(ch=='A'){
            player.Attack(monster);
        }
        else if(ch=='S'){
            player.Show_status();
        }
        else if(ch=='U'|| ch=='D'){
            if(ch=='U') player.Y_move(1);
            else player.Y_move(0);
        }
        else if(ch== 'L'||ch == 'R'){

            if(ch=='L') player.X_move(1);
            else player.X_move(0);
        }
        else{
            std::cout<<"다시 제대로 입력하세요"<<std::endl;

        }
        std::cin.clear(); //오류 상태 초기화
        std::cin.ignore(1000, '\n'); //버퍼에 들어간 문자 무시
    }
    return 0;
}
```

객체를 생성한 후 입력된 것에 따라 메서드를 호출하게 한 것입니다. 이번에는 char 변수에 입력을 받는 것이므로 문자에 대한 예외처리가 필요하지 않았고 A, U, D, R, L, S가 아닌 경우에 대해 else로 빼서 다시 제대로 입력하라는 문구를 출력하는 것으로 예외처리를 하였습니다. 그리고 AAAA처럼 연속으로 입력하는 것에 대해 4번 공격하는 것을 방지하기 위해 else 아래 마지막에 cin의 버퍼를 비우는 코드를 넣었습니다.

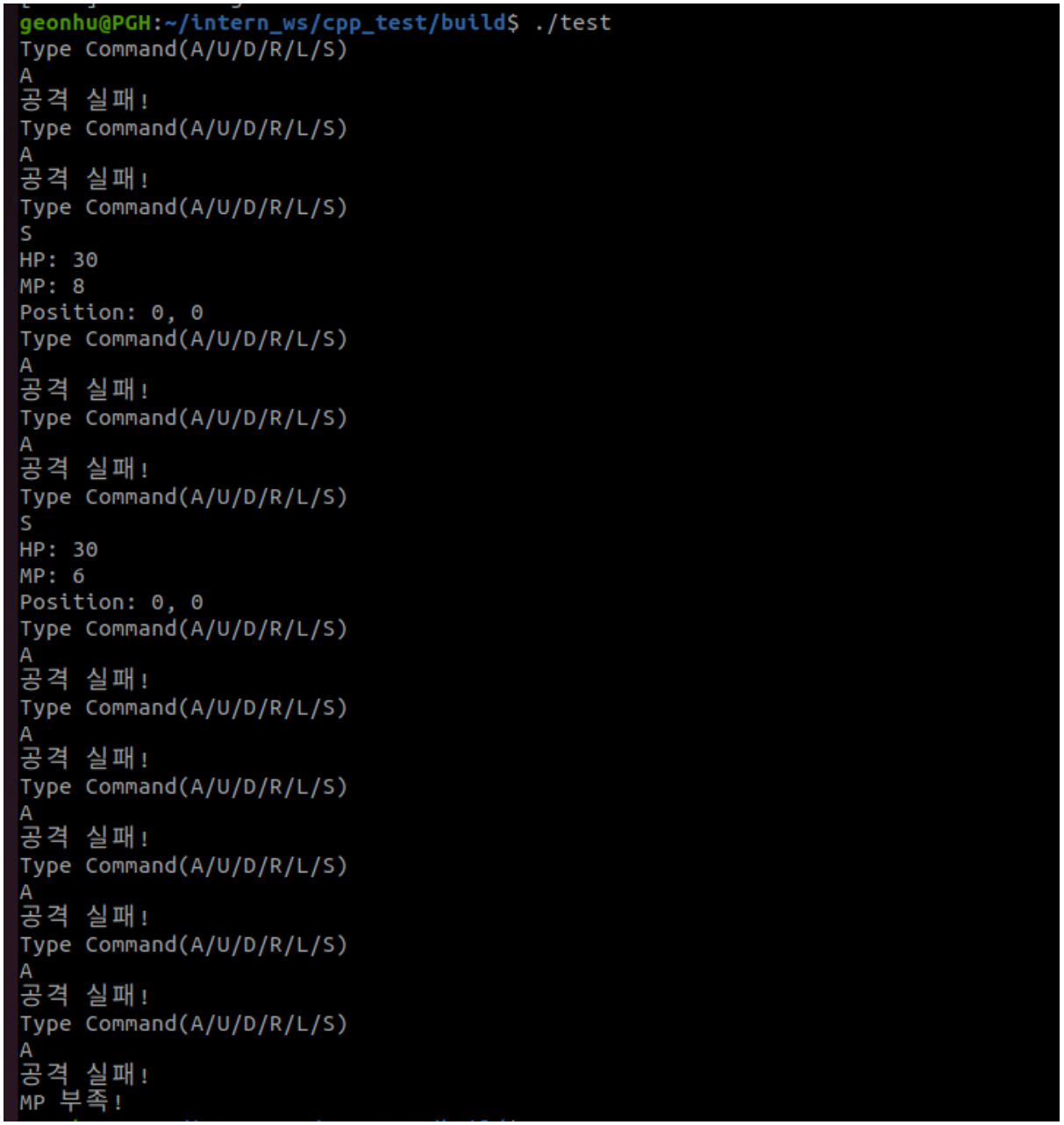
(4) 실행 사진



3이나 AAAA등의 예외를 잘 처리하는 모습을 담았고 기본적인 A, S, 움직임에 대한 것을 보여드리려 했습니다.



공격이 성공했을 때 계속 공격하였지만 공격이 실패되는 것을 볼 수 있습니다. 그 때 위로 이동하여 공격했을 때 다시 공격이 성공하는 것을 볼 수 있습니다. 그리고 몬스터가 죽으면 게임이 종료됩니다.



이 부분은 MP가 부족하여 게임이 종료되는 부분입니다.