P1.S6

작성일시	@2024년 8월 14일 오전 2:00
강의 번호	C++ 언리얼
유형	강의
복습	☐
날짜	@2024년 8월 14일

포인터 실습

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 포인터 실습

struct StatInfo {
    int hp;
    int attack;
    int defence;
};

void EnterLobby();
StatInfo CreatePlayer();
void CreateMonster(StatInfo* info);
bool StartBattle(StatInfo* player, StatInfo* monster);

int main() {
    EnterLobby();
    return 0;
}

void EnterLobby() {
    cout << "로비에 입장했습니다" << endl;

    StatInfo player;
```

```
    player.hp = 0xbbbbbbbb;
    player.attack = 0xbbbbbbbb;
    player.defence = 0xbbbbbbbb;

    // [매개변수][RET][지역변수(temp(c,c,c), player(b,b,b))] [매개
    // [지역변수(ret(100, 10, 2))] 이 값이 지역변수(temp(c,c,c)에 ㄱ
    // 중간 단계 temp를 확인 할 수 있다
    // StatInfo의 크기가 커지면 이렇게 중간중간 복사 하는 과정은 부하를
    player = CreatePlayer();

    StatInfo monster;
    monster.hp = 0xbbbbbbbb;
    monster.attack = 0xbbbbbbbb;
    monster.defence = 0xbbbbbbbb;

    // [매개변수][RET][monster(b,b,b))] [매개변수(&monster)][RET]
    // monster의 주소를 이용해서 monster(40,8,1)로 변경해줌
    CreateMonster(&monster);

    // 번외편1)
    // 구조체끼리 복사할 때 무슨 일이 벌어질까?
    // 아래와 같이 모든 변수들에 대해서 하나하나 복사해준다
    // 한줄짜리라해도 복사가 빠르지 않을 수 있다(큰 게임)
    //player = monster;

    //player.hp = monster.hp;
    //player.attack = monster.attack;
    //player.defence = monster.defence;

    bool victory = StartBattle(&player, &monster);

    if (victory)
        cout << "승리!" << endl;
    else
        cout << "패배!" << endl;

}
```

```
StatInfo CreatePlayer() {
    StatInfo ret;

    cout << "플레이어 생성" << endl;

    ret.hp = 100;
    ret.attack = 10;
    ret.defence = 2;

    return ret;
}

void CreateMonster(StatInfo* info) {
    cout << "몬스터 생성" << endl;

    info->hp = 40;
    info->attack = 8;
    info->defence = 1;
}

bool StartBattle(StatInfo* player, StatInfo* monster) {
    while (true) {
        int damage = player->attack - monster->defence;
        if (damage < 0)
            damage = 0;

        monster->hp -= damage;
        if (monster->hp < 0)
            monster->hp = 0;

        cout << "몬스터 HP : " << monster->hp << endl;

        if (monster->hp == 0)
            return true;

        damage = monster->attack - player->defence;
        if (damage < 0)
```

```
            damage = 0;

        cout << "플레이어 HP : " << player->hp << endl;

        player->hp -= damage;
        if (player->hp == 0)
            return false;
    }
}
```

참조 전달

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 참조

struct StatInfo {
    int hp;
    int attack;
    int defence;
};

// [매개변수][RET][지역변수(info)][매개변수(&info)][RET][지역변수]
void CreateMonster(StatInfo* info) {
    info->hp = 100;
    info->attack = 8;
    info->defence = 5;
};

void CreateMonster(StatInfo info) {
    info.hp = 100;
    info.attack = 8;
    info.defence = 5;
};

// 값을 수정하지 않는다면 양쪽 다 일단 문제없음
```

```
// 1) 값 전달 방식

void PrintInfoByCopy(StatInfo info) {
    cout << "--------------------" << endl;
    cout << "HP : " << info.hp << endl;
    cout << "ATT : " << info.attack << endl;
    cout << "DEF : " << info.defence << endl;
    cout << "--------------------" << endl;
}

// 2) 주소 전달 방식

void PrintInfoByPtr(StatInfo* info) {
    cout << "--------------------" << endl;
    cout << "HP : " << info->hp << endl;
    cout << "ATT : " << info->attack << endl;
    cout << "DEF : " << info->defence << endl;
    cout << "--------------------" << endl;
}

// StatInfo 구조체가 1000바이트짜리 대형 구조체라면?
// -(값 전달) StatInfo로 넘기면 1000바이트 복사
// -(주소 전달) StatInfo*는 8바이트
// -(참조 전달) StatInfo&는 8바이트


// 3) 참조 전달 방식
// 값 전달처럼 편리하게 사용하고
// 주소 전달처럼 주소값을 이용해서 진퉁을 건드리는 일석이조 방식

void PrintInfoByRef(StatInfo& info) {
    cout << "--------------------" << endl;
    cout << "HP : " << info.hp << endl;
    cout << "ATT : " << info.attack << endl;
    cout << "DEF : " << info.defence << endl;
    cout << "--------------------" << endl;
}
```

```
int main() {
    // 4바이트 정수형 바구니를 사용할거야.
    // 앞으로 그 바구니 이름을 number라고 할게
    // 그러니깐 number에서 뭘 꺼내거나, number에 뭘 넣는다고 하면
    // 찰떡같이 알아듣고 해당 주소(data, stack, heap)에 1을 넣어주면
    int number = 1;

    // * 주소를 담는 바구니
    // int 그 바구니를 따라가면 int 데이터(바구니)가 있음
    int* pointer = &number;
    // pointer 바구니에 있는 주소를 타고 이동해서, 그 멀리 있는 바구니에
    *pointer = 2;


    // 로우레벨(어셈블리) 관점에서 실제 작동 방식은 int*와 똑같음
    int& reference = number;

    // C++ 관점에서는 number라는 바구니에 또 다른 이름을 부여한 것
    // number라는 바구니에 reference라는 다른 이름을 지어줄게!
    // 앞으로 reference 바구니에다가 뭘 꺼내거나 넣으면, 실제 number바
    reference = 3;

    // 그런데 굳이 똑같은걸 다른이름으로 붙인 이유는?
    // 그냥 number = 3 이라고 해도 똑같은데...
    // 참조 전달 때문!
    // 일반적인 복사와 비슷한 방식으로 사용해서 편리 + 진퉁을 건드리는 편

    StatInfo info;
    CreateMonster(&info);

    PrintInfoByCopy(info);
    PrintInfoByPtr(&info);
    PrintInfoByRef(info);

    return 0;
}
```

포인터 vs 참조

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 포인터 vs 참조

struct StatInfo {
    int hp;
    int attack;
    int defence;
};

// [매개변수][RET][지역변수(info)][매개변수(&info)][RET][지역변수]
void CreateMonster(StatInfo* info) {
    info->hp = 100;
    info->attack = 8;
    info->defence = 5;
};

StatInfo globalInfo;

void PrintInfoByPtr(const StatInfo* const info) {
    // 널인지 아닌지 체크해서 크래시를 막아줌
    if (info == nullptr)
        return;

    cout << "--------------------" << endl;
    cout << "HP : " << info->hp << endl;
    cout << "ATT : " << info->attack << endl;
    cout << "DEF : " << info->defence << endl;
    cout << "--------------------" << endl;

    // 별 뒤에 붙인다면?
    // StatInfo* const info
    // info라는 바구니의 내용물(주소)을 바꿀 수 없음
    // info는 주소값을 갖는 바구니 -> 이 주소값이 고정이다!
    //
```

```
    // 별 이전에 붙인다면?
    // info가 '가리키고 있는' 바구니의 내용물을 바꿀 수 없음
    // '원격' 바구니의 내용물을 바꿀 수 없음

    // info[ 주소값 ]    주소값[ 데이터 ]
    //info = &globalInfo;
    //info->hp = 10000;
}

void PrintInfoByRef(const StatInfo& info) {
    cout << "--------------------" << endl;
    cout << "HP : " << info.hp << endl;
    cout << "ATT : " << info.attack << endl;
    cout << "DEF : " << info.defence << endl;
    cout << "--------------------" << endl;

    // 신입이 왔다
    //info.hp = 10000;
}

#define OUT

void ChangeInfo(OUT StatInfo& info) {
    info.hp = 1000;
}

int main() {
    StatInfo info;

    // 포인터 vs 참조 세기의 대결
    // 성능 : 똑같음!
    // 편의성 : 참조 승!

    // 1) 편의성 관련
    // 편의성이 좋다는게 꼭 장점만은 아니다
    // 포인터는 주소를 넘기니 확실하게 원본을 넘긴다는 힌트를 줄 수 있는데
    // 참조는 자연스럽게 모르고 지나칠 수 있음!
    // ex) 마음대로 고친다면?
```

```
// const를 사용해서 이런 마음대로 고치는 부분 개선 가능

// 참고로 포인터도 const를 사용가능
// * 기준으로 앞에 붙이느냐, 뒤에 붙이느냐에 따라 의미가 달라진다

// 2) 초기화 여부
// 참조 타입은 바구니의 2번째 이름(별칭?)
// -> 참조하는 대상이 없으면 안됨
// 반면 포인터는 그냥 어떤~ 주소라는 의미
// -> 대상이 실존하지 않을 수도 있음
// 포인터에서 '없다'는 의미로? nullptr
// 참조 타입은 이런 nullptr
StatInfo* pointer = nullptr;
pointer = &info;
PrintInfoByPtr(pointer);

StatInfo& reference = info;
PrintInfoByRef(reference);

// 그래서 결론은?
// 사실 팀바팀 정해진 답없음

// 구글오픈소스는 포인터 사용, 언리얼에선 레퍼런스도 사용

// - 없는경우도 고려해야 한다면 pointer(null 체크 필수)
// - 바뀌지 않고 읽는 용도면 const ref&
// - 그 외 일반적으로 ref(명시적으로 호출할 때 OUT을 붙인다)
// -- 단, 다른 사람이 pointer를 만들어놓은걸 이어서 만든다면, 계속
ChangeInfo(OUT info);

// Bonus) 포인터로 사용하던걸 참조로 넘겨주려면?
// pointer[ 주소(&info) ]    ref, info[ 데이터 ]
// 포인터는 주소값을 가지고 있는거고, 참조값은 그냥 두번째 이름
//StatInfo& ref = *pointer;
//PrintInfoByRef(ref);
PrintInfoByRef(*pointer);

// Bonus) 참조로 사용하던걸 포인터로 넘겨주려면?
```

```
    // pointer[ 주소 ]    reference, info[ 데이터 ]
    //StatInfo* ptr = &reference;
    //PrintInfoByPtr(ptr);
    PrintInfoByPtr(&reference);

    return 0;
}
```

배열 기초

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 배열 기초

struct StatInfo {
    int hp = 0xAAAAAAAA;
    int attack = 0xBBBBBBBB;
    int defence = 0xDDDDDDDD;
};

int main() {
    // TYPE 이름[개수]

    // 배열의 크기는 상수여야 함 (VC 컴파일러 기준)
    const int mosnterCount = 10;
    StatInfo monsters[mosnterCount];

    // 여태껏 변수들의 [이름]은 바구니의 이름이었음
    int a = 10;
    int b = a;

    // 그런데 배열은 [이름] 조금 다르게 동작한다
    //StatInfo players[10];
    //players = monsters;

    // 그럼 배열의 이름은 뭐지?
    // 배열의 이름은 곧 배열의 시작 주소
```

```
// 정확히는 시작 위치를 가리키는 TYPE* 포인터
auto WhoAmI = monsters;

// 주소 [ (100, 10, 1) ] StatInfo[ ] StatInfo[ ] StatInfo[
// monster_0[ 주소 ]
StatInfo* monster_0 = monsters;
monster_0->hp = 100;
monster_0->attack = 10;
monster_0->defence = 1;

// 포인터의 덧셈! 진짜 1을 더하라는게 아니라, StatInfo 타입 바구니
// StatInfo[ ] 주소 [ (200, 20, 2) ] StatInfo[ ] StatInfo[
// monster_1[ 주소 ]
StatInfo* monster_1 = monsters + 1;
monster_1->hp = 200;
monster_1->attack = 20;
monster_1->defence = 2;

// 포인터와 참조는 자유자재로 변환 가능하다
// StatInfo[ ] StatInfo[ ] monster_2,주소 [ (300, 30, 3) ]
StatInfo& monster_2 = *(monsters + 2);
monster_2.hp = 300;
monster_2.attack = 30;
monster_2.defence = 3;

// [주의] 이거는 완전 다른 의미다
// StatInfo[ ] StatInfo[ ] 주소[ 내용물 ] StatInfo[ ] StatIn
// temp[ 내용물 ]
StatInfo temp;
temp = *(monsters + 2);
temp.hp = 400;
temp.attack = 40;
temp.defence = 4;

// 이를 좀 더 자동화한다!
for (int i = 0; i < 10; i++) {
    StatInfo& mosnter = *(monsters + i);
    mosnter.hp = 100 * (i + 1);
```

```
        mosnter.attack = 10 * (i + 1);
        mosnter.defence = 1 * (i + 1);
    }

    // 근데 *(monsters + i) 이건 너무 불편
    // 인덱스!
    // 배열은 0번부터 시작. N번째 인덱스에 해당하는 데이터에 접근하려면 배
    // *(monsters + i) = monsters[i]

    monsters[0].hp = 100;
    monsters[0].attack = 10;
    monsters[0].defence = 1;

    for (int i = 0; i < 10; i++) {
        StatInfo& mosnter = monsters[i];
        monsters[i].hp = 100 * (i + 1);
        monsters[i].attack = 10 * (i + 1);
        monsters[i].defence = 1 * (i + 1);
    }

    // 배열 초기화 문법 몇가지
    int numbers[5] = {}; // 다 0으로 밀어버린다
    int numbers1[10] = {1, 2, 3, 4, 5}; // 설정한 애들은 설정한 값
    int numbers2[] = {1, 2, 3, 4, 11, 24, 124, 14, 1 }; // 데이
    char helloStr[] = {'H', 'e', 'l', 'l', 'o', '\0'};
    cout << helloStr << endl;

    // 배열 요약
    // 1) 선언한다
    int arr[10] = {};
    // 2) 인덱스로 접근해서 사용
    arr[1] = 1;
    cout << arr[1] << endl;

    return 0;
}
```

포인터 vs 배열

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 포인터 vs 배열

// 배열은 함수 인자로 넘기면, 컴파일러가 알아서 포인터로 치환한다(char[]-
// 즉 배열의 전체 내용을 넘긴게 아니라 시작주소(포인터)만 넘긴 것
void Test(char a[]) {
    a[0] = 'x';
}

int main() {

    // .data 어딘가에 hello world를 저장해두고
    // 시작주소를 test1에 부여한것
    // test1[ 주소 ]
    const char* test1 = "Hello World";

    // .data 주소[H][e][l][l][o][][W][o][r][l][d][\0]
    // [H][e][l][l][o][][W][o][r][l][d][\0]
    // test2 = 주소
    char test2[] = "Hello World";

    // 포인터는 [주소를 담는 바구니]
    // 배열은 [닭장] 즉, 그 자체로 같은 데이터끼리 붙어있는 '바구니 모음'
    // - 다만 [배열 이름]은 '바구니 모음' 의 [시작 주소]

    // 배열을 함수의 인자로 넘기게 되면?
    // test2가 바뀐다!
    Test(test2);
    cout << test2 << endl;

    return 0;
}
```

로또 번호 생성

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 로또 번호 생성기

void Swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

void Sort(int numbers[], int count) {
    int min = 0;

    for (int i = 0; i < count; i++) {
        min = i;

        for (int j = i + 1; j < count; j++) {
            if (numbers[min] > numbers[j])
                min = j;
        }
        if(i != min)
            Swap(numbers[i], numbers[min]);
    }
}

void ChooseLotto(int numbers[]) {
    srand((unsigned)time(0));

    int count = 0;
    while (count != 6) {
        int randValue = 1 + (rand() % 45);

        bool found = false;
        for (int i = 0; i < count; i++) {
            if (numbers[i] == randValue) {
                found = true;
                break;
```

```
            }
        }

        if (found == false) {
            numbers[count] = randValue;
            count++;
        }
    }

    Sort(numbers, 6);
}

int main() {
    // 1) Swap 함수 만들기
    int a = 1;
    int b = 2;
    Swap(a, b);

    // 2) 정렬 함수 만들기 (작은 숫자가 먼저 오도록 정렬)
    int numbers[6] = { 1, 42, 3, 15, 5, 6 };
    Sort(numbers, sizeof(numbers) / sizeof(int));

    // 3) 로또 번호 생성기
    ChooseLotto(numbers);

    for (int i = 0; i < 6; i++) {
        cout << numbers[i] << endl;
    }

    return 0;
}
```

다중포인터

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 다중 포인터
```

```
void SetNumber(int* a) {
    *a = 1;
}

void SetMessage(const char* a) {
    a = "Bye";
}

void SetMessage(const char** a) {
    *a = "Bye";
}

void SetMessage2(const char*& a) {
    a = "Wow";
}

int main() {
    int a = 0;
    SetNumber(&a);
    //cout << a << endl;

    // .rdata Hello주소[H][e][l][l][o][\0]
    // msg[ Hello주소 ] << 8바이트
    const char* msg = "Hello";

    // 주소2 [ ] = .rdata Hello주소[H][e][l][l][o][\0] << 1바이트
    // 주소1[ Hello주소 ] << 8바이트
    // pp[ &msg ] << 8바이트
    const char** pp = &msg;

    // [매개변수][RET][지역변수(msg(Hello주소))] [매개변수(a(&msg ms
    // .rdata Hello주소[H][e][l][l][o][\0]
    // .rdata Hello주소[B][y][e][\0]
    // msg[ Hello주소 ] -> msg[ Bye주소 ]
    SetMessage(pp); // = SetMessage(&msg);
    cout << msg << endl;
```

```
    // 다중 포인터 : 혼동스럽다?
    // 그냥 양파까기라고 생각하면 된다
    // *을 하나씩까면서 타고 간다고 생각하면
    const char** pp2;

    // 참조도 내부적으로는 포인터와 같기때문에 넣어줄수있음
    SetMessage2(msg);
    cout << msg << endl;

    return 0;
}
```

다차원 배열

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 다차원 배열

int main() {


    int apartment2D[2][5] = { {4, 2, 3, 4, 1}, {1, 1, 5, 2, 2

    for (int floor = 0; floor < 2; floor++) {
        for (int room = 0; room < 5; room++) {
            // apartment2D + (floor * 20) + 4 * room를 한 주소
            int num = apartment2D[floor][room];
            cout << num << " ";
        }
        cout << endl;
    }

    int apartment1D[10] = { 4, 2, 3, 4, 1, 1, 1, 5, 2, 2 };
    for (int floor = 0; floor < 2; floor++) {
        for (int room = 0; room < 5; room++) {
            int index = (floor * 5) + room;
            // apartment1D + (floor * 20) + 4 * room를 한 주소
```

```
            int num = apartment1D[index];
            cout << num << " ";
        }
        cout << endl;
    }

    // 2차 배열은 언제 사용할까? 대표적으로 2D 로그라이크 맵
    int map[5][5] =
    {
        {1, 1, 1, 1, 1},
        {1, 0, 0, 1, 1},
        {0, 0, 0, 0, 1},
        {1, 0, 0, 0, 0},
        {1, 1, 1, 1, 1}
    };
    for (int y = 0; y < 5; y++) {
        for (int x = 0; x < 5; x++) {
            int info = map[y][x];
            cout << info;
        }
        cout << endl;
    }

    return 0;
}
```

포인터 마무리

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 포인터 마무리

// 1) 포인터 vs 배열 2탄
// 2) 주의사항 (마음가짐

int& TestRef() {
    int a = 1;
```

```
    return a;
}

int* TestPointer() {
    int a = 1;
    return &a;
}

void TestWrong(int* ptr) {
    int a[100] = {};
    a[99] = 0xAAAAAAAA;
    // 엉뚱한 곳에 주소를 넣어줌
    *ptr = 0x12341234;
}

int main() {
    // 주소를 담는 바구니
    // 진퉁은 저~ 멀리 어딘가에 있음
    // p는 단지 그 곳으로 워프하는 포탈

    int* p;

    // 진짜배기! 원조 데이터
    // 닭장처럼 데이터의 묶음(엄청 많고 거대함)
    int arr[10] = { 1, 2, 3, 4, 5, 6, 7, 8 };

    // 그런데 상당히 많은 사람들이 [ 배열 = 포인터 ] 라 착각하는 경향이

    // - [배열의 이름]은 배열의 시작 주소값을 가리키는 TYPE* 포인터로 변
    p = arr;

    // - [TYPE형 1차원 배열]과 [TYPE*형 포인터]는 완전히 호환이 된다
    cout << p[0] << endl;
    cout << arr[0] << endl;
    cout << p[5] << endl;
    cout << arr[5] << endl;
    cout << *p << endl; // p[0]
    cout << *arr << endl; // arr[0]
```

```
    cout << *(p + 3) << endl;
    cout << *(arr + 3) << endl;

    // 지옥을 보여드리겠습니다. (2차원 배열 vs 다중 포인터)

    int arr2[2][2] = { {1, 2},{3, 4} };

    // [1][2] [3][4]
    // TYPE name[개수]
    int(*p2)[2] = arr2;
    cout << (*p2)[0] << endl;
    cout << (*p2)[1] << endl;
    cout << (*(p2 + 1))[0] << endl;
    cout << (*(p2 + 1))[1] << endl;

    cout << p2[0][0] << endl;
    cout << p2[0][1] << endl;
    cout << p2[1][0] << endl;
    cout << p2[1][1] << endl;


    // [매개변수][RET][지역변수]
    int* pointer = TestPointer();

    TestWrong(pointer);
    *pointer = 123;

    return 0;
}
```

TEXT RPG3

```
#include <iostream>
using namespace std;

// 오늘의 주제 : TextRPG2
```

```
// main
// - EnterLobby (PlayerInfo)
// -- CreatePlayer
// -- EnterGame (MonsterInfo)
// --- CreateMonsters
// --- EnterBattle

enum PlayerType
{
    PT_Knight = 1,
    PT_Archer = 2,
    PT_Mage = 3,
};

enum MonsterType
{
    MT_Slime = 1,
    MT_Orc = 2,
    MT_Skeleton = 3,
};

struct StatInfo
{
    int hp = 0;
    int attack = 0;
    int defence = 0;
};

void EnterLobby();
void PrintMessage(const char* msg);
void CreatePlayer(StatInfo* playerInfo);
void PrintStatInfo(const char* name, const StatInfo& info);
void EnterGame(StatInfo* playerInfo);
void CreateMonsters(StatInfo monsterInfo[], int count);
bool EnterBattle(StatInfo* playerInfo, StatInfo* monsterInfo)

int main()
{
```

```
    srand((unsigned)time(nullptr));
    EnterLobby();
    return 0;
}

void EnterLobby()
{
    while (true)
    {
        PrintMessage("로비에 입장했습니다");

        // PLAYER!
        StatInfo playerInfo;
        CreatePlayer(&playerInfo);
        PrintStatInfo("Player", playerInfo);

        EnterGame(&playerInfo);
    }
}

void PrintMessage(const char* msg)
{
    cout << "************************" << endl;
    cout << msg << endl;
    cout << "************************" << endl;
}

void CreatePlayer(StatInfo* playerInfo)
{
    bool ready = false;

    while (ready == false)
    {
        PrintMessage("캐릭터 생성창");
        PrintMessage("[1] 기사 [2] 궁수 [3] 법사");
        cout << "> ";

        int input;
```

```
        cin >> input;

        switch (input)
        {
        case PT_Knight:
            playerInfo->hp = 100;
            playerInfo->attack = 10;
            playerInfo->defence = 5;
            ready = true;
            break;
        case PT_Archer:
            playerInfo->hp = 80;
            playerInfo->attack = 15;
            playerInfo->defence = 3;
            ready = true;
            break;
        case PT_Mage:
            playerInfo->hp = 50;
            playerInfo->attack = 25;
            playerInfo->defence = 1;
            ready = true;
            break;
        }
    }
}

void PrintStatInfo(const char* name, const StatInfo& info)
{
    cout << "******************" << endl;
    cout << name << " : HP=" << info.hp << " ATT=" << info.at
    cout << "******************" << endl;
}

void EnterGame(StatInfo* playerInfo)
{
    const int MONSTER_COUNT = 2;

    PrintMessage("게임에 입장했습니다");
```

```
    while (true)
    {
        StatInfo monsterInfo[MONSTER_COUNT];
        CreateMonsters(monsterInfo, MONSTER_COUNT);

        for (int i = 0; i < MONSTER_COUNT; i++)
            PrintStatInfo("Monster", monsterInfo[i]);

        PrintStatInfo("Player", *playerInfo);

        PrintMessage("[1] 전투 [2] 전투 [3] 도망");

        int input;
        cin >> input;

        if (input == 1 || input == 2)
        {
            int index = input - 1;
            bool victory = EnterBattle(playerInfo, &(monsterI
            if (victory == false)
                break;
        }
    }
}

void CreateMonsters(StatInfo monsterInfo[], int count)
{
    for (int i = 0; i < count; i++)
    {
        int randValue = 1 + rand() % 3;

        switch (randValue)
        {
        case MT_Slime:
            monsterInfo[i].hp = 30;
            monsterInfo[i].attack = 5;
            monsterInfo[i].defence = 1;
```

```
            break;
        case MT_Orc:
            monsterInfo[i].hp = 40;
            monsterInfo[i].attack = 8;
            monsterInfo[i].defence = 2;
            break;
        case MT_Skeleton:
            monsterInfo[i].hp = 50;
            monsterInfo[i].attack = 15;
            monsterInfo[i].defence = 3;
            break;
        }
    }
}

bool EnterBattle(StatInfo* playerInfo, StatInfo* monsterInfo)
{
    while (true)
    {
        int damage = playerInfo->attack - monsterInfo->defenc
        if (damage < 0)
            damage = 0;

        monsterInfo->hp -= damage;
        if (monsterInfo->hp < 0)
            monsterInfo->hp = 0;

        PrintStatInfo("Monster", *monsterInfo);

        if (monsterInfo->hp == 0)
        {
            PrintMessage("몬스터를 처치했습니다");
            return true;
        }

        damage = monsterInfo->attack - playerInfo->defence;
        if (damage < 0)
            damage = 0;
```

```
        playerInfo->hp -= damage;
        if (playerInfo->hp < 0)
            playerInfo->hp = 0;

        PrintStatInfo("Player", *playerInfo);

        if (playerInfo->hp == 0)
        {
            PrintMessage("Game Over");
            return false;
        }
    }
}
```

연습 문제 (문자열) - 코테 단골문제들

```
#include <iostream>
using namespace std;

// 오늘의 주제 : 연습 문제

// 문제1) 문자열의 길이를 반환
int StrLen(const char* str) {
    int count = 0;
    while (str[count] != '\0') {
        count++;
    }

    return count;
}

// 문제2) 문자열 복사 함수
char*  StrCpy(char* dest, char* src) {
    /*int count = 0;
    while (src[count] != '\0') {
        dest[count] = src[count];
        count++;
```

```
    }
    dest[count] = src[count];*/

    char* ret = dest;

    while (*src) {
        *dest++ = *src++;
    }
    *dest = '\0';

    return ret;
}

// 문자3) 문자열 덧붙이는 함수
char* StrCat(char* dest, char* src) {
    /*int len = StrLen(dest);

    int i = 0;
    while (src[i] != '\0') {
        dest[len + i] = src[i];
        i++;
    }
    dest[len + i] = '\0';

    return dest;*/

    char* ret = dest;

    while (*dest) {
        dest++;
    }

    while (*src) {
        *dest++ = *src++;
    }

    *dest = '\0';
```

```
    return ret;
}

// 문제4) 두 문자열을 비교하는 함수
int StrCmp(char* a, char* b) {
    int i = 0;

    while (a[i] != '\0' || b[i] != '\0') {
        if (a[i] > b[i])
            return 1;
        if (a[i] < b[i])
            return -1;
        i++;
    }

    return 0;
}

// 문제5) 문자열을 뒤집는 함수
void ReverseStr(char* str) {
    int len = StrLen(str);

    for (int i = 0; i < len / 2; i++) {
        int temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

#pragma warning(disable: 4996)

int main() {
    const int BUF_SIZE = 100;
    char a[BUF_SIZE] = "Hello";
    char b[BUF_SIZE];
    char c[BUF_SIZE] = "World";
    char d[BUF_SIZE] = "Hello";
    char e[BUF_SIZE] = "Hello";
```

```
    int len = StrLen(a);
    cout << len << endl;

    StrCpy(b, a);

    cout << b << endl;

    StrCat(a, c);

    cout << a << endl;

    int result = StrCmp(d, e);

    cout << result << endl;

    ReverseStr(a);

    cout << a << endl;

    return 0;
}
```

연습문제(달팽이)

```
#include <iostream>
using namespace std;
#include <iomanip>

// 오늘의 주제 : 연습 문제

const int MAX = 100;
int board[MAX][MAX] = {};
int N;

void PrintBoard() {
    for (int y = 0; y < N; y++) {
        for (int x = 0; x < N; x++) {
```

```
            cout << setfill('0') << setw(2) << board[y][x] <<
        }
        cout << endl;
    }
}

enum DIR {
    RIGHT = 0,
    DOWN = 1,
    LEFT = 2,
    UP = 3,
};

bool CanGo(int y, int x) {
    if (y < 0 || y >= N)
        return false;
    if (x < 0 || x >= N)
        return false;
    if (board[y][x] != 0)
        return false;
    return true;
}

void SetBoard() {
    int dir = RIGHT;
    int num = 1;
    int y = 0;
    int x = 0;

    int dy[] = { 0, 1, 0, -1 };
    int dx[] = { 1, 0, -1, 0 };

    while (true) {
        board[y][x] = num;

        if (num == N * N)
            break;
```

```
        int nextY = y + dy[dir];
        int nextX = x + dx[dir];

        /*switch (dir){
        case RIGHT:
            nextY = y;
            nextX = x + 1;
            break;
        case DOWN:
            nextY = y + 1;
            nextX = x;
            break;
        case LEFT:
            nextY = y;
            nextX = x - 1;
            break;
        case UP:
            nextY = y - 1;
            nextX = x;
            break;
        }*/

        if (CanGo(nextY, nextX)) {
            y = nextY;
            x = nextX;
            num++;
        }
        else {
            dir = (dir + 1) % 4;

            /*switch (dir) {
            case RIGHT:
                dir = DOWN;
                break;
            case DOWN:
                dir = LEFT;
                break;
            case LEFT:
```

```
                dir = UP;
                break;
            case UP:
                dir = RIGHT;
                break;
            }*/
        }
    }
}

int main() {
    cin >> N;

    SetBoard();

    PrintBoard();

    return 0;
}
```

파일 분할 관리

CPP_Study_1.cpp

```
#include <iostream>
using namespace std;
#include "Test1.h"
#include "Test2.h"
// 오늘의 주제 : 파일 분할 관리


int main() {
    Test_2();

    return 0;
}
```

Test1.cpp

```
#include <iostream>
using namespace std;
#include "Test1.h"

void Test_1() {
    Test_2();
}

void Test_2() {
    cout << "Hello World" << endl;
}

void Test_3() {

}
```

Test1.h

```
//#pragma once

#ifndef _TEST1_H__
#define _TEST1_H__
// 올드한 방식, 하지만 컴파일러에서 문제가 생기는 경우가 간혹생기기에 사용하

// 헤더파일에 #include <iostream> 등등 막 써도 문제는 없겠지만 이 헤더
// 따라서 정말 꼭 필요한것만 include해놓는다

struct StatInfo {
    int hp;
    int attack;
    int defence;
};

void Test_1();

void Test_2();

void Test_3();
```

```
#endif
```

Test2.h

```
#pragma once

#include "Test1.h"
```