P1.S3

작성일시	@2024년 4월 6일 오후 10:19
강의 번호	C++ 언리얼
유형	강의
복습	☐
날짜	@2024년 5월 26일

불리언

```
#include <iostream>
using namespace std;
// 불리언(boolean) 참/거짓
bool isHighLevel = true;
bool isPlayer = true;
bool isMale = false;

// [Note]
// 사실 bool은 그냥 1바이트 정수에 불과
// 왜 정수 시간에 안 다뤘을까?
// -> 일리있음. 어셈블리에서 bool이라는 것은 없다
// bool만 봐도 참/ 거짓 둘 중 하나라는 힌트를 줍니다.(가독성)
int isFemale = 1;

// bool < 1바이트 정수
// - al

// 실수 (부동소수점)
// float double
// 3.14
// 쩜 앞/뒤를 기준으로 16/16씩 끊으면?
// (0-65535) . (0-65535)

// 부동(아닐 不가 아니고 뜰 浮) 소수점
// .을 유동적으로 움직여서 표현하는 방법
```

```
// 3.1415926535
// 3.1415926535 = 0.31415926435 * 10 = 314/15926535 * 10^-2
// 1) 정규화 = 0.31415926535 * 10
// 2) 31415926535 (유효숫자) 1 (지수)

// float 부호(1) 지수(8) 유효숫자(23) = 32비트 = 4바이트
// double 부호(1) 지수(11) 유효숫자(52) = 64비트 = 9바이트
float attackSpeed = -3.375f; // 4바이트
double attackSpeed2 = 123.4123; // 8바이트

// ex) -3.375라는 값을 저장
// 1) 2진수 변환 = (3) + (0.375) = 0b11 + 0.375
// 0.5 * 0 + 0.25 * 1 + 0.125 * 1 = 0b0.011
// 2) 정규화 0.b1.1011 * 2^1
// 1(부호) 1(지수) 1011(유효숫자)
// 단 지수는 unsigned byte라고 가정하고 숫자 + 127 만들어줌
// 예상 결과 : 0b 1 10000000 1011000'0000'0000'0000'0000

// 프로그래밍할 때 부동소수점은 항상 '근사값'이라는 것을 기억
// 1/3 = 0.3333333333333...
// 특히 수가 커질 수록 오차 범위도 매우 커짐
// 실수 2개를 == 으로 비교하는 것은 지양

int main()
{
    cout << isHighLevel << endl;
    //여성 갯수? 뭔말이지?

}
```

문자와 문자열

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

// 오늘의 주제 : 문자와 문자열
// bool은 그냥 정수지만, 참/거짓을 나타내기 위해 사용한다 했었다
```

```
// 사실 char도 마찬가지. 그냥 정수지만 '문자' 의미를 나타내기 위해 사용

// char : 알파벳 / 숫자 문자를 나타낸다
// wchar_t : 유니코드 문자를 나타낸다

// ASCII (American Standard Code for Information Interchange)

// '문자'의 의미로 작은 따옴표 '' 사용
char ch = 'a';
char ch2 = '1';
char ch3 = 'a' + 1;

// 국제화 시대에는 영어만으로 서비스 할 수 없음
// 전 세계 모든 문자에 대해 유일 코드를 부여한 것이 유니코드
// 참고) 유니코드에서 가장 많은 번호를 차지하는게 한국어/중국어

// 유니코드는 포기 방식이 여러가지가 있는데 대표적으로 UTF8 UTF16
// UTF8
// - 알파벳, 숫자 1바이트(ASCII 동일한 번호)
// - 유럽 지역의 문자는 2바이트
// - 한글, 한자 등 3바이트
// UTF16
// - 알파벳, 숫자, 한글, 한자 등 거의 대부분 문자 2바이트
// - 매~~우 예외적인 고대 문자만 4바이트(사실상 무시해도 됨)

wchar_t wch = 0xc548; // L'안';

// Escape Sequence
// 표기하기 애매한 애들을 표현
// \t = 아스키코드9 = Tab
// \n = 아스키코드10 = LineFeed(한줄 아래로)
// \r = 아스키코드13 = CarriageReturn(커서 <<)
// \' ''' 표현
// \0 아스키코드0

// 문자열
// 문자들이 열을 지어서 모여 있는 것(문자 배열?)
// 정수(1~8바이트) 고정 길이로
```

```
// 끝은 NULL \0
char str[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
char str2[] = "Hello World";

int main()
{
    cout << ch << endl;
    cout << ch2 << endl;
    cout << ch3 << endl;

    // cout은 char전용
    wcout.imbue(locale("kor"));
    wcout << wch << endl;



    cout << str << endl;
    cout << str2 << endl;
}
```

산술 연산

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

// 오늘의 주제 : 데이터 연산
// 데이터를 가공하는 방법에 대해서 알아보자

// a[ 1 ]
// a라는 이름의 바구니를 할당하고 안에 1을 넣는다
int a = 1;

// b[ 2 ]
// b라는 이름의 바구니를 할당하고 안에 2을 넣는다
int b = 2;

int main()
{
#pragma region 산술 연산
```

```
    // 산술 연산자

    // a에 b를 대입하고 b를 반환하라
    // -> b라는 바구니 안에 있는 값을, a라는 바구니 안에다 복사한다

    // 대입연산
    a = b;

    // 사칙연산
    // 언제 필요한가?
    // ex) 데미지 계산, 체력을 깎는다, 루프문에서 카운터를 1증가
    a = b + 3;
    a = b - 3;
    a = b * 3;
    a = b / 3;
    a = b % 3;

    a += 3;
    a -= 3;
    a *= 3;
    a /= 3;
    a %= 3;

    // 증감 연산자
    a = a + 1; // add eax, 1 -> inc eax
    a++;
    ++a;
    a--;
    --a;

    b = a++; // b = a -> a를 1증가
    b = ++a; // a를 1증가 -> b = a

    b = (a + 1) * 3;

#pragma endregion
}
```

비교 연산, 논리 연산

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

bool isSame;
bool isDifferent;
bool isGreater;
bool isSmaller;

bool test;

int hp = 100;
bool isInvincible = true;


int main()
{
#pragma region 비교 연산

    // 언제 필요한가?
    // ex) 체력이 0이되면 사망, 체력이 30프로 이하면 궁극기 발동 (100

    // a == b : a와 b의 값이 같은가?
    // 같으면 1, 다르면 0
    isSame = (a == b);

    // a != b : a와 b의 값이 다른가?
    //다르면 1, 같으면 0
    isDifferent = (a != b);

    // a > b : a가 b보다 큰가?
    // a >= b : a가 b보다 크거나 같은가?
    // 1아니면 0
    isGreater = (a > b);

    // a < b : a가 b보다 작은가?
    // a <= b
```

```
    // 1아니면0
    isSmaller = (a < b);

#pragma endregion

#pragma region 논리 연산
    // 언제 필요한가? 조건에 대한 논리적 사고가 필요할 때
    // ex) 로그인할 때 아이디도 같고 AND 비밀번호도 같아야 한다
    // ex) 길드 마스터이거나 or 운영자 계정이면 길드 해산 가능

    // ! not
    // 0이면 1, 그 외 0
    test = !isSame; // 사실상 isDifferent의 의미

    // && and
    // a && b -> 둘다 1이면 1, 그 외 0
    test = (hp <= 0 && isInvincible == false); // 죽음



    // || or
    // a || b -> 둘 중 하나라도 1이면 1(둘다 0이면 0)
    test = (hp > 0 && isInvincible == true); // 살았음


#pragma endregion
}
```

비트 연산, 비트 플래그

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

unsigned char flag; // 부호를 없애야 >>를 하더라도 부호비트가 딸려오지

int main()
{
#pragma region 비트 연산
```

```
// 언제 필요한가? (사실 많이는 없음)
// 비트 단위의 조작이 필요할 때
// - 대표적으로 BitFlag

// - bitwise not
// 단일 숫자의 모든 비트를 대상으로, 0은 1, 1은 0으로 뒤바꿈

// & bitwise and
// 두 숫자의 모든 비트 쌍을 대상으로, and를 한다

// | bitwise or
// 두 숫자의 모든 비트 쌍을 대상으로, or를 한다

// ^ bitwise xor
// 두 숫자의 모든 비트 쌍을 대상으로, xor를 한다

// << 비트 좌측 이동
// 비트열을 N만큼 왼쪽으로 이동
// 왼쪽의 넘치는 N개의 비트는 버림, 새로 생성되는 N개의 비트는 0
// *2를 할 때 자주 보이는 패턴

// >> 비트 우측 이동
// 비트열을 N만큼 오른쪽 이동
// 오른쪽 넘치는 N개의 비트는 버림,
// 왼쪽 생성되는 N개의 비트는
// - 부호 비트가 존재할 경우 부호 비트를 따라감(부호있는 정수라면 이
// - 아니면 0

// 실습
// 0b0000 [무적][변이][스턴][공중부양]

// 무적 상태로 만든다
flag = (1 << 3);

// 변이 상태를 추가한다 (무적 + 변이)
flag |= (1 << 2);

// 무적인지 확인하고 싶다? (다른 상태는 관심 없음)
```

```
    // bitmask
    bool invincible = ((flag & (1 << 3)) != 0);

    // 무적이거나 스턴 상태인지 확인하고 싶다면?
    bool mask = (1 << 3) | (1 << 1);
    bool stunOrInvincible = ((flag & mask) != 0);

#pragma endregion
}
```

const 와 메모리 영역

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

unsigned char flag; // 부호를 없애야 >>를 하더라도 부호비트가 딸려오지

// 한번 정해지면 절대 바뀌지 않을 값들
// constant의 약자인 const를 붙임 (변수를 상수화 한다고 함)
// const를 붙였으면 초기값을 반드시 지정해야 함

// 그러면 const도 바뀌지 않는 읽기 전용
// .radata?
// 사실 C++ 표준에서 꼭 그렇게 하라는 말이 없음
// 그냥 컴파일러 (VS) 마음이라는 것

const int AIR = 0;
const int STUN = 1;
const int POLYMORPH = 2;
const int INVINCIBLE = 3;

// 전역 변수

// [데이터 영역]
// .data (초기값 있는 경우)
int a = 2;

// .bss (초기값 없는 경우)
```

```
int b;

// .rodata (read only 읽기 전용 데이터)
const char* msg = "Hello World";

int main()
{
    // [스택 영역]
    int c = 3;
#pragma region 비트 연산

    // 언제 필요한가? (사실 많이는 없음)
    // 비트 단위의 조작이 필요할 때
    // - 대표적으로 BitFlag

    // - bitwise not
    // 단일 숫자의 모든 비트를 대상으로, 0은 1, 1은 0으로 뒤바꿈

    // & bitwise and
    // 두 숫자의 모든 비트 쌍을 대상으로, and를 한다

    // | bitwise or
    // 두 숫자의 모든 비트 쌍을 대상으로, or를 한다

    // ^ bitwise xor
    // 두 숫자의 모든 비트 쌍을 대상으로, xor를 한다

    // << 비트 좌측 이동
    // 비트열을 N만큼 왼쪽으로 이동
    // 왼쪽의 넘치는 N개의 비트는 버림, 새로 생성되는 N개의 비트는 0
    // *2를 할 때 자주 보이는 패턴

    // >> 비트 우측 이동
    // 비트열을 N만큼 오른쪽 이동
    // 오른쪽 넘치는 N개의 비트는 버림,
    // 왼쪽 생성되는 N개의 비트는
    // - 부호 비트가 존재할 경우 부호 비트를 따라감(부호있는 정수라면 이
    // - 아니면 0
```

```
    // 실습
    // 0b0000 [무적][변이][스턴][공중부양]

    // 무적 상태로 만든다
    flag = (1 << INVINCIBLE);

    // 변이 상태를 추가한다 (무적 + 변이)
    flag |= (1 << POLYMORPH);

    // 무적인지 확인하고 싶다? (다른 상태는 관심 없음)
    // bitmask
    bool invincible = ((flag & (1 << INVINCIBLE)) != 0);

    // 무적이거나 스턴 상태인지 확인하고 싶다면?
    bool mask = (1 << INVINCIBLE) | (1 << STUN);
    bool stunOrInvincible = ((flag & mask) != 0);

#pragma endregion
}
```

유의사항

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

// 오늘의 주지 : 유의사항

// 1) 변수의 유효범위

// 전역 변수
//int hp = 10;

// 스택
// {} 중괄호의 범위가 생존 범위
// 같은 이름 두번 사용할 때 문제

// 2) 연산 우선순위
```

```
//bool isEvenOrDivBy3 = ((hp % 2) == 0) || ((hp % 3) == 0);

// 3) 타입 변환

// 4) 사칙 연산 관련

int main()
{
    int hp = 123;

    // 바구니 교체

    short hp2 = hp; // 윗쪽 비트 데이터가 짤린 상태로 저장
    float hp3 = hp; // 실수로 변환할 때 정밀도 차이가 있기 때문에 데이
    unsigned int hp4 = hp; // 비트 단위로 보면 똑같은데, 분석하는 방

    // 곱셈
    // - 오버플로우
    // 나눗셈
    // - 0 나누기 조심
    // - 실수 관련

    int maxHp = 1000;

    float ratio = hp / (float)maxHp; // 0.123
}
```

가위바위보

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

int main() {
    const int SCISSORS = 1;
    const int ROCK = 2;
    const int PAPER = 3;

    int win = 0;
```

```
    int count = 0;

    while (true) {
        cout << "가위(1) 바위(2) 보(3) 골라주세요!" << endl;
        cout << "> ";

        if (count == 0) {
            cout << "현재 승률 : 없음" << endl;
        }
        else {
            int stats = 100 * win / count;
            cout << "현재 승률(" << win << "/" << count << ") :
        }

        int computerValue = 1 + (rand() % 3);

        int input;
        cin >> input;

        if (input == SCISSORS) {
            switch (computerValue) {
            case SCISSORS:
                cout << "가위(님) vs 가위(컴퓨터) 비겼습니다!" << e
                break;
            case ROCK:
                cout << "가위(님) vs 바위(컴퓨터) 졌습니다!" << end
                count++;
                break;
            case PAPER:
                cout << "가위(님) vs 보(컴퓨터) 이겼습니다!" << end
                win++;
                count++;
                break;
            }
        }
        else if (input == ROCK) {
            switch (computerValue) {
            case SCISSORS:
```

```
                cout << "바위(님) vs 가위(컴퓨터) 이겼습니다!" << e
                win++;
                count++;
                break;
            case ROCK:
                cout << "바위(님) vs 바위(컴퓨터) 비겼습니다!" << e
                break;
            case PAPER:
                cout << "바위(님) vs 보(컴퓨터) 졌습니다!" << endl
                count++;
                break;
            }
        }
        else if (input == PAPER) {
            switch (computerValue) {
            case SCISSORS:
                cout << "보(님) vs 가위(컴퓨터) 졌습니다!" << endl
                count++;
                break;
            case ROCK:
                cout << "보(님) vs 바위(컴퓨터) 이겼습니다!" << end
                win++;
                count++;
                break;
            case PAPER:
                cout << "보(님) vs 보(컴퓨터) 비겼습니다!" << endl
                break;
            }
        }
        else {
            cout << "종료합니다" << endl;
            break;
        }
        cout << endl;
    }
}
```

열거형

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

// 상수인건 알겠는데 너무 따로 노는느낌?
const int SCISSORS = 1;
const int ROCK = 2;
const int PAPER = 3;

// 숫자를 지정안하면 첫 값은 0부터 시작
// 그 다음 값들은 이전 값 + 1
enum ENUM_SRP {
    ENUM_SCISSORS = 1,
    ENUM_ROCK,
    ENUM_PAPER
};

// #이 붙은거 -> 전처리 지시문
//  #include <iostream> 이라는 파일을 찾아서 해당 내용을 그냥 복붙
// 1) 전처리 2) 컴파일 3) 링크
// 단점) 디버깅할때 보이질 않음. 직접 파일을 찾아가서 확인해야하는 경우도있
// 따라서 디파인은 최대한 지양하자는 국룰이있음
#define DEFINE_SCISSORS 1+2
#define DEFINE_TEST cout << "Hello World" << endl;

int main() {
    DEFINE_TEST;

    // 1 + 2 * 2
    // 이런 문제를 해결하기위해서는 define에서 괄호로 감싸주기
    int a = DEFINE_SCISSORS * 2;
    cout << a << endl;
}
```