**<const int형 포인터>**

포인터 변수의 주소를 변경 -> 가능

const int \*ptr1 = &val1;

/\* const int \*ptr;

ptr = &val1;

//가능. \*/

ptr1 = &val2; //가능

de-refer값 변경 -> 불가능

const int \*ptr1 = &val1;

\*ptr1 = 6; //불가능

**<포인터 자체가 상수인 경우>**

포인터 변수의 주소를 변경 -> 불가능

int \*const ptr1 = &val1; //※Compile Time에 Initialize필요

ptr1 = &val2; // 🡨 불가능

de-refer값 변경 -> 가능

int \*const ptr1 = &val1;

\*ptr1 = 6; // 🡨 가능

**<const int 형 상수 포인터>**

포인터 변수의 주소를 변경 -> 불가능

de-ref값 변경 -> 불가능

const int \*const ptr = &val; //※CompileTime 에 Initialize 필요.

ptr = &val2; // 🡨불가능

\*ptr = 10; // 🡨불가능

**<참조변수 : reference variable>**

특징 1. val == ref

특징 2. &val == &ref

특징 3. 참조변수는 컴파일타임에 초기화 되어야 한다. (int &ref; <-- 불가능)

특징 4. 초기화 할 때 Rvalue는 메모리 주소가 있어야한다. 즉, Literal Initialization 불가능하다. (int &ref = 10; <- 불가능)(예외적으로 const int &ref는 가능.)

의의 1. 다중 구조체에서 깊게 들어가야하는 멤버를 참조할 때 유용함.

struct Sth

{

int v1;

float v2;

};

struct Sth\_other

{

Sth st;

};

//main

ot.st.v1 = 1.0; //번거롭다.

int &v1 = ot.st.v1;

v1 = 1; //간편하다.

<참조와 const>

const int &ref = 3+4;

//int &ref의 경우 Rvalue를 Literal Initialization이 불가능 하지만, const의 경우 가능.

const int ref = 3+4;

언뜻보면, const int ref와 const int& ref는 차이점이 없어 보인다.

void doSth(const int ref){…} //변수가 복사된다.

void doSth(const int& ref){…} //변수복사가 발생하지 않아 메모리 면에서 효율적이다.