Basic_Concepts_3 - Structure Initialization | Dynamic Memory Initialization

Structure Initialization

```
struct S{
    unsigned x;
    unsigned y;
};

S s1; // 기본 초기화. x, y는 쓰레기값이 들어간다.
S s2 = {}; // 복사 리스트 초기화. x, y는 기본값이 들어간다.
S s3 = {1, 2}; // 복사 리스트 초기화. x=1, y=2
S s4 = {1}; // 복사 리스트 초기화. x=1, y=기본값
//S s5(3, 5); // 컴파일 에러. 초기화가 없다.

S f() {
    S s6 = {1, 2}; // 장황하다.
    return s6;
}
```

```
struct S{
    unsigned x;
    unsigned y;
    void* ptr;
};

S s1{}; // 직접 리스트 초기화.
                // x, y, ptr 모두 기본값으로 초기화된다.
S s2{1, 2}; // 직접 리스트 초기화
                        // x=1, y=2, ptr=기본값
// S s3{1, -2} // 컴파일 에러. Narrowing conversion(타입의 크기가
```

```
S f() {return {3, 2}; } // 장황하지 않다.
```

```
struct S {
    unsigned x = 3; // 대입 초기화
    unsigned y = 2; // 대입 초기화
};
struct S1 {
    unsigned x {3}; // brace initialization
};

S s1; // 기본 생성자 호출. x=3, y=2
S s2{}; // 기본 생성자 호출 (x=3, y=2)
S s3{1, 4}; // 세팅 x=1, y=4
```

C++20부턴 지정 초기화 리스트를 지원한다.

```
struct A{
    int x, y, z;
};

A a1{1, 2, 3}; // 아래와 동일하다.
A a2{.x = 1, .y = 2, .z = 3}; // 지정 초기화 리스트
```

지정 초기화 리스트는 코드의 가독성을 높이는데 유용하게 사용할 수 있다.

```
void f1(bool a, bool b, bool c, bool d, bool e) {}
// 같은 타입의 긴 리스트는 에러를 일으키기 쉽다.

struct B {
    bool a, b, c, d, e;
};
f2({.a = true, .c = true}); // b, d, e = false
```

Structure Binding

```
struct A {
    int x = 1;
```

```
    int y = 2;
} a;

A f() {return A{4, 5}; }

// 상황1. 구조체
auto [x1, y1] = a; // x1=1, y1=2
auto [x2, y2] = f(); // x2=4, y2=5
// 상황2. 배열
int b[2] = {1, 2};
auto [x3, y3] = b; // x3=1, y3=2
// 상황3. 튜플
auto [x4, y4] = std::tuple<float, int>{3.0f, 2};
```

Dynamic Memory Initialization

C++03

```
int* a1 = new int; // 쓰레기값
int* a2 = new int(); // 기본값
int* a3 = new int(4); // 하나의 값을 할당해서 4를 대입했다.
int* a4 = new int[4]; // 4개의 원소를 할당하고 쓰레기값이 들어있다.
int* a5 = new int[4](); // 4개의 원소를 할당하고 기본갑싱 들어있다.
//int* a6 = new int[4](3); // 유효하지 않다.
```

C++11

```
int* b1 = new int[4]{}; // 4개의 원소를 할당하고 기본값이 들어있다.
int* b2 = new int[4]{1, 2}; // 4개의 원소를 할당하고 1, 2, 기본값이
```