Basic_Concepts_3 - Variable Initialization | Uniform Initialization | Fixed-size Array Initialization

Variable Initialization

```
int a1; // 기본 초기화 (쓰레기 값)

int a2(2); // 직접 초기화
int a3(0); // 직접 초기화 (쓰레기값 정리)
// int a4(); // 이렇게 하면 함수이다.

int a5 = 2; // 복사 초기화
int a6 = 2u; // 복사 초기화 (암시적 형변환)
int a7 = int(2); // 복사 초기화
int a8 = int(); // 복사 초기화 (쓰레기값 정리)

int a9 = {2}; // 복사 리스트 초기화
```

Uniform Initialization

C++11의 brace-initialization 혹은 braced-init-list 라고도 불리는 **Uniform Initialization** 문법은 다른 엔티티의 초기화를 허가한다.

uniform initialization은 안전하게 산술 타입으로 변환할 수 있어서, 값을 유실할 수 있는 가능성이 있는 암시적 형변환을 막는다.

```
int b1{2}; // 직접 리스트 초기화
int b2{}; // 직접 리스트 초기화 (쓰레기값 정리)

int b3 = int{}; // 복사 초기화 (쓰레기값 정리)
int b4 = int{4}; // 복사 초기화

int b5 = {}; // 복사 리스트 초기화 (쓰레기값 정리)
```

```
int b6 = -1; // ok
int b7{-1}; // ok
int b8 = -1; // ok
unsigned b9{-1}; // 컴파일 에러

float f1{10e30}; // ok
float f2 = 10e40; // ok, "inf" 값.
//float f3{10e40}; // 컴파일 에러
```

Fixed-Size Array Initialization

1차원

```
int a[3] = {1, 2, 3}; // 명시적 크기
int b[] = {1, 2, 3}; // 암시적 크기
char c[] = "abcd"; // 암시적 크기
int d[3] = {1, 2}; // d[2] = 0 -> 기본 값

int e[4] = {0}; // 모든 값이 0으로 초기화된다.
int f[3] = {}; // 모든 값이 0으로 초기화된다. (C++11)
int g[3] = {}; // 모든 값이 0으로 초기화된다. (C++11)
```

2차원

```
int a[][2] = {{1, 2}, {3, 4}, {5, 6}}; // ok
int b[][2] = {1, 2, 3, 4}; // ok
// a와 b의 타입은 int[]의 배열 타입이다.
//int c[][] = ...; // 컴파일 에러
//int d[2][] = ...; // 컴파일 에러
```