C++ - default/zero initialization

SCSC **장필식**

Reference

http://en.cppreference.com/w/cpp/language/type

https://isocpp.org/wiki/faq

https://en.wikipedia.org/wiki/C%2B%2B_classes

C++ type system

- Fundamental types
- Compound types
  - pointers
  - references
  - arrays
  - POD
  - non-POD
  - Aggregate

Fundamental (Primitive) types

void, int, float, double, bool, ...

default vs zero vs value initialization

```
// default initialization
// value of a is indeterminate
int a;

// zero initialization
// value of a is 0
int a = int();
int a = {};
int a{};

// value initialization
// value of a is 1
int a = int(1);
int a{1};
```

default initialization rules

http://en.cppreference.com/w/cpp/language/default_initialization

Use of an indeterminate value obtained by default-initializing a non-class variable of any type is undefeined behavior

```
int f(bool b)
{
    int x;                // OK: the value of x is indeterminate
    int y = x;            // undefined behavior
    unsigned char c;      // OK: the value of c is indeterminate
    unsigned char d = c; // OK: the value of d is indeterminate
    int e = d;            // undefined behavior
    return b ? d : 0;     // undefined behavior if b is true
}
```

effects of zero initialization

If T is a scalar type, the object's initial value is the integral constant zero explicitly converted to T.

```
int x = int();
int x = {};
int x{};
```

is the same as:

```
int x = 0;
```

compound types

- reference types
- pointer types
- pointer to member types
- array types
- function types
- enumeration types
- class types

reference/pointer/array types

- default initialization
  - reference: not allowed
  - pointer types: indeterminate
  - array types: every element is default initialized
- zero initialization
  - reference: not allowed
  - pointer types: to nullptr (NULL, 0)
  - array types: every element is zero initialized

Initialization for class types

Default constructor

A constructor that has no arguments

```
// create a default constructor on our own
ClassName() {}
// delete the default constructor
ClassName() = delete;
// synthesized default constructor - generated by the compiler
ClassName() = default;
```

synthesized default constructor rules (C++ Primer)

The synthesized constructor initializes each member of the class as:

- If there is an in-class initializer, use it to intialize the member
- Otherwise, default-initialize the member

```
struct StudentInfo {
    // value is default initialized (indeterminate)
    int id;

    // value is initialized with in-class initializer
    double grade = 0.0;

    // value is default initialized with std::vector's default
    std::vector<double> homework;
}
```

synthesized default constructor rules (C++ Primer, 7.1.4)

- The compiler generates a default constructor automatically only if:
  - the class declares no constructors
  - every member which is a class type has a default constructor

Example

```
class NoDefault {
public:
    NoDefault(const std::string&);
    // additional members follow, but no other constructors
};
struct A {  // my_mem is public by default; see § 7.2 (p. 268)
    NoDefault my_mem;
};
A a;        //  error: cannot synthesize a constructor for A
struct B {
    B() {} //  error: no initializer for b_member
    NoDefault b_member;
}
```

default/zero initialization and constructor

default init. and zero init. boths calls the default constructor

```
struct StudentInfo {
    StudentInfo() : name("Default Student"), grade(0.0) {}
    std::string name;
    double grade;
    std::vector<double> homework;
};

int main() {
    StudentInfo student1;
    StudentInfo student2 {};
    std::cout << student1.name << " " <<
        student1.grade << std::endl;
    std::cout << student2.name << " " <<
        student2.grade << std::endl;
}
```

```
Default Student 0
Default Student 0
```

default/zero initialization and constructor

But if the default constructor does default initialization on primitive types... may get garbage value!

```
struct Student {
    StudentInfo() = default;
    std::string name;
    double grade;
    std::vector<double> homework;
};

int main() {
    StudentInfo student1;
    StudentInfo student2 {};

    //student1.grade and student2.grade are both indeterminate
    std::cout << student1.name << " " <<
        student1.grade << std::endl;
    std::cout << student2.name << " " <<
        student2.grade << std::endl;
}
```

Some exceptions

https://stackoverflow.com/questions/2417065/does-the-default-constructor-initialize-built-in-types

```
struct C {
    int x;
}
```

```
C c; // c.x is default initialized (indeterminate value)
```

```
C c = C(); // c.x is zero initialized
```

Note: C() does not call the default constructor!

C() calls a special constructor that zero-initializes its fields

Why? Because C is a POD type

POD (Plain Old Data) type

말그대로 데이터 밖에 없고, 따로 Constructor도 정의 되지 않고, 상속이나 polymorphism등을 사용하지 않는 타입이다.

POD type는 C랑 호환되는 타입으로, C에서도 완전히 똑같은 방식으로 메모리에 배열된다.

음... 복잡하다

사실은 총체적 난국

- initialization에 대한 규칙은 C++ 버젼 (C++98, C++03, C++11, C++14, C++17) 에 따라서 바뀔 수 있다. (cppreference.com을 가보면 문서의 절반 정도가 버젼마다 조금씩 달라지는 내용인 것을 볼 수 있다)
- 심지어 Visual Studio 컴파일러에는 value-initialization이 공식 C++ 스펙과 다르게 작동하는 버그가 있었다! (VS 2015에서 고쳐짐)

교훈

즉 총체적 난국이니... **변수를 만들면 초기화하는걸 잊지 말자**.

끼으읕