A Primer On Initialization

steven arellano

Agenda

Initialization at a High Level

Syntax of Initialization

Types of Initialization

Initialization in C++

```
vector<int> vec1(5, 8); // {8, 8, 8, 8, 8}

vector<int> vec2{5, 8}; // {5, 8}


int a = 3.14159; // 3

int b(3.14159); // 3

int c{3.14159}; // ERROR

int d = {3.14159}; // ERROR
```

Agenda

Initialization at a High Level

Syntax of Initialization

Types of Initialization

Syntax of Initialization

declarations

```
<decl-specifier-seq> <declarator> [initializer] [requires-clause] ;
```

```
int num = 10;

int num{42};

int arr[] = {1, 2, 3};
```

…also in initialization in functions & new expression.

Agenda

Initialization at a High Level

Syntax of Initialization

Types of Initialization

Default Initialization

T object;

Fundamental Types ⇒ Undefined Behavior

Code

```
int a; // UB
```

Classes ⇒ Default constructor if available; else members are undefined.

```
struct Point { int a; };

Point p; // p.a = UB
```

Aggregates ⇒ For each element, follows its types default init method.

Copy Initialization

T object = other;

Code

```
int x = 5;

class MyClass {

public:

      int a;

      MyClass(int val): a(val) {}

      MyClass(const MyClass& other): a(other.a * 2){}

}

MyClass obj1 = 10; // obj1.a = 10

MyClass obj2 = obj1; // obj2.a = 20
```

Non-class type ⇒ Standard conversion is performed

Same Class or Prvalue ⇒ Initialize without creating unnecessary temps.

Same or Derived Class ⇒ The most appropriate constructor, excluding explicits.

Different Class ⇒ Pick the user defined conversion if available.

Aggregate Initialization

T object = {arg1, arg2, …}

[since C++11]

T object{arg1, arg2, …}

[since C++20]

T object = {.des1=arg1, .des2{arg2}}

T object{.des1=arg1, .des2{arg2}}

Aggregates ⇒ Assigns each value of the initializer list in the order the members/elements.

Code

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

Point p = {1, 2}; // {1,2,0}

Point p = {.y=2, .z=3} // {0,2,3}
```

Zero Initialization

static T object;

T object = {};

Code

```
int x = {}; // 0

class Constr {

public:

      int h; float s;

}

Constr obj2 = {} // obj2.h = 0, obj2.s = 0
```

Fundamental Types ⇒ Set values to 0 for the given type (e.g. 0.0, nullptr, etc.).

Classes w/ data members & no user-defined constructors ⇒ Zero-init all members.

Aggregates ⇒ Zero init all members

Direct Initialization

[since c++98]

T object(arg);

T object(arg1, arg2, …);

[since C++11]

T object{arg};

Code

```
int x(5);

vector<int> vec(3, 10); // {10, 10, 10}
```

Fundamental types ⇒ Directly initialize the value using the arg.

Classes ⇒ best-matching constructor (non-explicit or explicit) is selected.

Arrays or aggregates ⇒ each member initialized directly.

Value Initialization

[since C++03]

T();

[since C++11]

T object{};

Code

```
int i{}; // 0
int i(); // FUNCTION DECL
```

Fundamental types ⇒ Objects are zero-initialized

Classes ⇒ Default constructor if available, else all members are zero-init

Arrays or aggregates ⇒ each member is zero-init.

List Initialization (Uniform Initialization)

```
[since C++11]

T object{arg1, arg2};

T object = {arg1, arg2};


[since C++20]

T object{.des1=arg1, .des2{arg2},...}

T object = {.des1=arg1, .des2{arg2},...}
```

Aggregates ⇒ <covered in aggregate initialization>

Classes ⇒ Calls initializer list constructor if available.

Arrays or aggregates ⇒ each member is zero-init.

Summary

Default Initialization

Copy Initialization

~~Aggregate Initialization~~

Zero Initialization

Direct Initialization

Value Initialization

List Initialization

Aggregate Initialization

…copy elision, static initialization, initialization in params