Doc. No.: D2190R0 DRAFT 1
Project: Programming Language - C++ (WG21)
Author: Andrew Tomazos <andrewtomazos@gmail.com>
Date: 2020-06-17
Audience: EWGI

Proposal of Designated Constructor Parameters

1. Abstract

We propose designated constructor parameters. They are constructor parameters that correspond directly to data members. As such, a designated constructor parameter implicitly initializes their corresponding data member, and arguments to them can be given with designated initializer syntax. (Today designated initializer syntax can only be used with aggregate types.)

2. Examples

2.1 Example #1

```
class Point {
   // Four dots means all constructor parameters are designated.
   Point(....) {
      std::cout << "point at " << x << ", " << y;
   }

   int x;
   int y;
};

int main() {
    Point pt{.x=3, .y=4}; // outputs: point at 3, 4
}
```

2.2 Example #2

```
class Rect {
public:
    // The type of these constructor parameters
    // are taken from the data member.
    Rect(.height, .width) : area(height*width) {}

    int get_area() { return area; }

private:
    int height, width, area;
};

int main() {
    Rect rect{.height = 3, .width = 4};
    assert(rect.get_area() == 12);
}
```

2.3 Example #3

```
class Employee {
   // The type of the designated constructor parameter differs from
   // the type of its corresponding data member.  Conversion is
   // implicit.
   Employee(std::string_view .name) : id(get_next_id()) {}
   std::string name;
   int id;
};

int main() {
   Employee fred{.name = “fred”};
   assert(is_valid_id(fred.id));
}
```

3. Background

3.1 Background Point #1

It is a common occurrence that a constructor parameter is used to initialize a data member, and for that to be the only use of that constructor parameter:

```
class C {
public:
   C(int x_in) : x(x_in) {}

private:
   int x;
};
```

3.2 Background Point #2

Sometimes no naming convention is used to distinguish the constructor parameter and data member:

```
class C {
public:
   C(int x) : x(x) {}

private:
   int x;
};
```

This can be surprising even to experienced programmers. It works because lookup of the member initializer name only considers data members. Many consider this an anti-pattern as it is easy to accidentally use the constructor parameter when intending the data member:

```
class C {
public:
   C(int x) : x(x) {
      f(&x); // BUG: pointer is to parameter, not data member
   }

private:
   int x;
```

```
};
```

Either way it is extremely rare that one would want to continue using the parameter after the corresponding data member has been initialized.

3.3 Background Point #3

In C++20 we have the designated initializer feature that allows aggregate types to be initialized using a designated syntax:

```
// TODAY
struct Point { int x,y; };
int main() { Point pt{.x=3, .y=4}; }
```

4. Informal Specification

4.1. Syntax

declarator-id:

... $_{opt}$ id-expression

designated-constructor-parameter

designated-constructor-parameter:

. identifier

. ...

4.2. Semantics

1. A designated-constructor-parameter may only appear in the parameter-declaration-clause of a constructor. For the form with an identifier, that identifier must name a non-static data member of the constructor's class. For the ellipsis form, there shall be exactly one designated-constructor-parameter in the parameter-declaration-clause.

2. A designated-constructor-parameter may be declared without a type:

In such cases it is as-if the parameter was declared with the type of its corresponding data member.

Example 4:

```
struct S { S(.x) {} int x; };
```

is equivalent to:

```
struct S { S(int .x) {} int x; };
```

3. A single designated-constructor-parameter of the ellipsis form is equivalent to a sequence of designated-constructor-parameters without a type where the declaration of all possible designated-constructor-parameters appearing in declaration-order of the data members.

Example 5:

```
struct S { S(....) {} int a, b, c; }
```

is equivalent to:

```
struct S { S(.a, .b, .c) {} int a, b, c; }
```

4. The behaviour of a constructor with a designated-constructor-parameter with a type T is equivalent to a parameter declaration of type T where the parameter has some name __N that doesn't otherwise appear in the program and is used to initialize the corresponding data member:

Example 6:

```
struct S { S(T .k) {} U k; }
```

is equivalent to:

```
struct S { S(T __N) : k(__N) {} U k; }
```

except that the parameter object behaves as an unnamed temporary. (That is, because it may not be otherwise referred to in the program, it may be moved from or its existence may be elided altogether.)

5. A designated-initializer-list may initialize a class type with a constructor that has designated-constructor-parameters. In such a case the designated-initializer-list behaves as the equivalent initializer-list without designators, except that the designators must correspond to list elements that initialize a designated-constructor-parameters of the same name:

Example 7:

```
struct S { S(.a, .b, .c) {} /*...*/ }
```

```
int main() {
   S s{1,2,3}; // OK
   S s{.a=1,.b=2,.c=3}; // OK and equivalent
   S s{.c=3, .a=1, .b=2}; // ERROR: designator mismatch
   S s{.a=1, .b = 2, 3}; // OK
}
```