Default constructor

From Wikipedia, the free encyclopedia

In computer [programming languages](http://en.wikipedia.org/wiki/Programming_languages) the term **default constructor** can refer to a [constructor](http://en.wikipedia.org/wiki/Constructor_(computer_science)) that is automatically generated by the compiler in the absence of any programmer-defined constructors (e.g. in Java), and is usually a [nullary constructor](http://en.wikipedia.org/wiki/Nullary_constructor" \o "Nullary constructor). In other languages (e.g. in C++) it is a constructor that can be called without having to provide any arguments, irrespective of whether the constructor is auto-generated or user-defined. Note that a constructor with formal [parameters](http://en.wikipedia.org/wiki/Parameter_(computer_science)) can still be called without arguments if[default](http://en.wikipedia.org/wiki/Default_(computer_science)) arguments were provided in the constructor's definition.

C++[[edit](http://en.wikipedia.org/w/index.php?title=Default_constructor&action=edit&section=1" \o "Edit section: C++)]

In [C++](http://en.wikipedia.org/wiki/C%2B%2B), the standard describes the default constructor for a class as a [constructor](http://en.wikipedia.org/wiki/Constructor_(computer_science)) that can be called with no arguments (this includes a constructor whose parameters all have default arguments).[[1]](http://en.wikipedia.org/wiki/Default_constructor#cite_note-1) For example:

class MyClass

{

public:

MyClass(); // constructor declared

private:

int x;

};

MyClass :: MyClass() : x(100) // constructor defined

{

}

int main()

{

MyClass m; // at runtime, object m is created, and the default constructor is called

}

When allocating memory dynamically, the constructor may be called by adding parenthesis after the class name. In a sense, this is an explicit call to the constructor:

int main()

{

MyClass \* pointer = new MyClass(); // at runtime, an object is created, and the

// default constructor is called

}

If the constructor does have one or more parameters, but they all have default values, then it is still a default constructor. Remember that each class can have at most one default constructor, either one without parameters, or one whose all parameters have default values, such as in this case:

class MyClass

{

public:

MyClass (int i = 0, std::string s = ""); // constructor declared

private:

int x;

int y;

std::string z;

};

MyClass :: MyClass(int i, std::string s) // constructor defined

{

x = 100;

y = i;

z = s;

}

In C++, default constructors are significant because they are automatically invoked in certain circumstances; and therefore, in these circumstances, it is an error for a class to not have a default constructor:

* When an object value is declared with no argument list (e.g.: MyClass x;) or allocated dynamically with no argument list (e.g.: new MyClass; or new MyClass();), the default constructor of MyClass is used to initialize the object.
* When an array of objects is declared, e.g. MyClass x[10];; or allocated dynamically, e.g. new MyClass [10]. The default constructor of MyClass is used to initialize all the elements.
* When a derived class constructor does not explicitly call the base class constructor in its initializer list, the default constructor for the base class is called.
* When a class constructor does not explicitly call the constructor of one of its object-valued fields in its initializer list, the default constructor for the field's class is called.
* In the standard library, certain containers "fill in" values using the default constructor when the value is not given explicitly. E.g. vector<MyClass>(10); initializes the vector with ten elements, which are filled with a default-constructed MyClass> object.

If a class has no explicitly defined constructors, the compiler will implicitly declare and define a default constructor for it. This implicitly defined default constructor is equivalent to an explicitly defined one with an empty body. For example:[[2]](http://en.wikipedia.org/wiki/Default_constructor" \l "cite_note-2)

class MyClass

{

int x; // no constructor, so the compiler produces an (implicit) default constructor

};

int main()

{

MyClass m; // no error at runtime: the (implicit) default constructor is called

}

If constructors are explicitly defined for a class, but they are all non-default, the compiler will not implicitly define a default constructor, leading to a situation where the class does not have a default constructor. This is the reason for a typical error, demonstrated by the following example.

class MyClass

{

public:

MyClass (int y); // declaration a non-default constructor

private:

int x;

};

MyClass :: MyClass (int y)

{

x = y;

}

int main()

{

MyClass m(100); // the non-default constructor is called

MyClass \* p; // for pointer declarations, the compiler does not need to know about constructors

p = new MyClass(); // error at compilation: no default constructor

return 0;

}

Since neither the programmer nor the compiler has defined a default constructor, the creation of the objected pointed to by p leads to an error.[[3]](http://en.wikipedia.org/wiki/Default_constructor#cite_note-3)

On the other hand in C++11 a default constructor can be explicitly created:

class MyClass

{

public:

MyClass () = default; // force generation of a default constructor

};

Or explicitly inhibited:

class MyClass

{

public:

MyClass () = delete; // prevent generation of default constructor

};