Class-C Specification

Preface

Since Class-C is a very thin layer over C, Class-C specification is based on the ISO C specification. The latest ISO C specification (C11 : ISO/IEC 9899:2011) can be reached at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1570.pdf or milgra.com/classc/n1570.pdf, this specification is an extension of that document.

The language and the specification was created by Milan Toth.
milgra@milgra.com
http://milgra.com

6.4 Lexical elements

6.4.1. Keywords

1. All keywords from the same section of the ISO C specification and

Syntax

keyword:

self

6.5. Expressions

6.5.2. Postfix operators

1. All operators from the same section of the ISO C specification and

Syntax

postfix-expression:

postfix-expression **:** *identifier*

6.5.2.2. Function calls

1. [extending the same section of the ISO C specification] There are two kinds of function call : ordinary function call and member function call.

6.5.2.3.1. Class member access

1. A postfix expression followed by a dot or a double-colon and then followed by an *identifier*, is a member access expression. The postfix expression before the dot or double-colon is evaluated; the result of that evaluation, together with the *identifier*, determines the result of the entire postfix expression.
2. The type of the first expression (the object expression) shall be "class object" (of a complete type).
3. There are four types of member function calls : class-explicit call, instance-explicit call, local-explicit call and instance-implicit call.
4. In a class-explicit call the class and the method is declared explicitly. The first parameter is an object or NULL. If the first parameter is not NULL, it has to be dynamically casted to the call's class.
   [*Example :*
   ```
   CLObject:init( self );
   CLObject:init( instance );
   CLObject:doSomething( NULL , 5 , 6 );
   ```
   --- end example]

5. In an instance-explicit call the class is declared explicitly from the type of the object, and it is mapped to a class-explicit call, but first parameter is not dynamically casted.
   [*Example :*
   `object:doSomething( ... );`
   --- end example]
6. local-explicit calls are member function calls on the same class from a class method definition, they are also mapped to class-explicit calls, but first parameter is not dynamically casted.
   [*Example :*
   `doSomething( ... );`
   --- end example]
7. instance-implicit calls are member function calls using the original instance's function mapping table, and using the original instance as first parameter.
   [*Example :*
   `object.doSomething( ... );`
   --- end example]

6.5.4. Cast operators

Syntax

1. *cast-expression:*
   unary-expression
   (type name) cast-expression

7. If type-name is a class, then cast-expression has to be dynamically casted to the type-name.

6.8. Statements and blocks

6.8.5. Iteration statements

All statements from the same section of the ISO C specification and

Syntax

iteration-statement:

for (*declaration expression*$_{opt}$ **:** *expression*$_{opt}$) *statement*

6.8.5.3. The for statement

3. The statement

for (*declaration expression-1* **:** *expression-2*) *statement*

behaves as follows : if *expression-2* is an iterable container, then the statements iterates through all elements of *expression-2* and in each step *expression-1* represents an element of *expression-2*.

6.12. Classes

1. A class is a type. Its name becomes a *class-name* within its scope.

   class-name:
   identifier

   An object of a Class-Consists of a sequence of members.

   class-specifier:
   class-head { member-specification$_{opt}$ *}*

   class-head:
   class-head-name base-clause$_{opt}$

2. A class name is inserted into the scope of the class itself.
3. If a block starting bracket is inserted after a class-name, it becomes a class definition. A class is considered defined after the closing brace of its *class-specifier*.

6.12.1. Class names

1. A class definition introduces a new type.
2. A class declaration introduces the class name into the scope where it is declared.

6.12.2. Class members

Syntax

member-specification:

member-declaration member-specification$_{opt}$

member-declaration:
member-declarator
function-definition

1. The member-specification in a class definition declares the full set of members of the class, no members can be added elsewhere. Members of a class are data members and member functions.
2. A class is considered a completely-defined object type at the closing } of the *class-specifier*.
3. A member of a class type can be allocated using an allocator member function.

6.12.3. Member functions

1. Functions defined in the definition of a class are called member functions of that class.

6.12.4. The self pointer

1. In the body of a member function, the keyword self is an lvalue expression whose value is the address of the object for which the function is called. The type of self in a member function of a class X is X*.

6.12.5. Derived classes

1. A list of base classes can be specified in a class definition using the notation :

Syntax

base-clause:
: *base-specifier-list*

base-specifier-list:
class-name
base-speficier-list

6.12.6. Multiple base classes

1. A Class-Can be derived from any number of base classes.

[*Example :*

```
classD:classC:classB:classA { }
```

--- *end example*]

2. Class member variable duplication causes compile error.
3. Class methods override each other in a left-to-right order.

6.13 Objects

1. Objects are allocated instances of classes.
2. Objects are standard structures with pre-defined members.

6.14. Allocator member function

1. The allocator is a special member function
2. The syntax uses the class name, the explicit accessor and the alloc function name.

[*Example :*

```
CLObject:alloc( );
```

--- *end example*]

3. An allocator is used to allocate objects of its class type.
4. An allocator returns a pointer containing the address of the allocated object.
5. The allocator is a built-in function, it cannot be overridden.

6.15. The object deallocator function

1. The deallocator is a special global function
2. The deallocator is used to destroy objects.
3. The syntax uses the object as the only parameter

[*Example :*

```
free_object( object );
```

--- *end example*]