The identifier used for a namespace name should be unique. It should not be used previously as a global identifier.

namespace Raymond {

// namespace body here...

}Copy

In this example, Raymond is the identifier of the namespace.

If you intend to access a namespace's elements, the namespace's identifier must be known in all translation units.

////////////////////////////Alias for namespace///////////////////////////////////////

namespace INTERNATIONAL\_BUSINESS\_MACHINES {

void f();

}

namespace IBM = INTERNATIONAL\_BUSINESS\_MACHINES;

In this example, the IBM identifier is an alias for INTERNATIONAL\_BUSINESS\_MACHINES.

This is useful for referring to long namespace identifiers.

If a namespace name or alias is declared as the name of any other entity in the same declarative region,

a compile-time error will result. Also, if a namespace name defined at global scope is declared

as the name of any other entity in any global scope of the program, a compile-time error will result.

//////////////////////////Nested NMSPC Alias/////////////////////////////

Namespace definitions hold declarations. Since a namespace definition is a declaration itself, namespace definitions can be nested.

An alias can also be applied to a nested namespace.

namespace INTERNATIONAL\_BUSINESS\_MACHINES {

int j;

namespace NESTED\_IBM\_PRODUCT {

void a() { j++; }

int j;

void b() { j++; }

}

}

namespace NIBM = INTERNATIONAL\_BUSINESS\_MACHINES::NESTED\_IBM\_PRODUCT

In this example, the NIBM identifier is an alias for the namespace NESTED\_IBM\_PRODUCT.

This namespace is nested within the INTERNATIONAL\_BUSINESS\_MACHINES namespace.

//////////////////////////////////NMPSC extend/////////////////////////////////////

Namespaces are extensible. You can add subsequent declarations to a previously defined namespace.

Extensions may appear in files separate from or attached to the original namespace definition.

For example:

namespace X { // namespace definition

int a;

int b;

}

namespace X { // namespace extension

int c;

int d;

}

namespace Y { // equivalent to namespace X

int a;

int b;

int c;

int d;

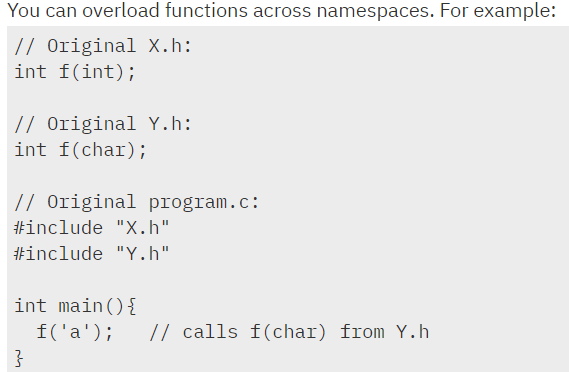
}

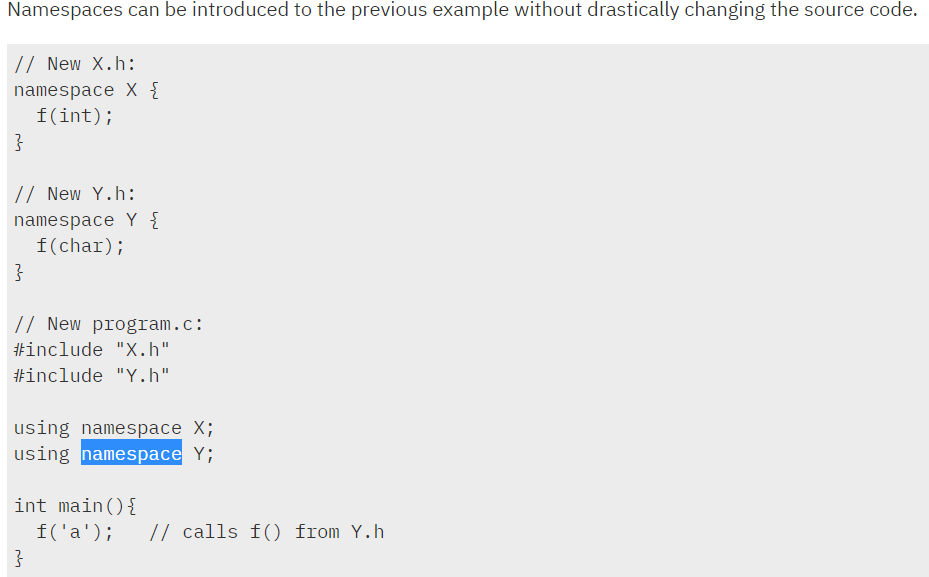
In this example, namespace X is defined with a and b and later extended with c and d.

namespace X now contains all four members. You may also declare all of the required members within one namespace.

This method is represented by namespace Y. This namespace contains a, b, c, and d.

//////////////////////////////////NMPSC extend/////////////////////////////////////





In program.c, the main function calls function f(), which is a member of namespace Y. If you place the using directives in the header files, the source code for program.c remains unchanged.