Chapter 40. Boost.Function

Boost.Function provides a class called `boost::function` to encapsulate function pointers. It is defined in `boost/function.hpp`.

If you work in a development environment supporting C++11, you have access to the class `std::function` from the header file `functional`. In this case you can ignore Boost.Function because `boost::function` and `std::function` are equivalent.

Example 40.1. Using `boost::function`

```
#include <boost/function.hpp>
#include <iostream>
#include <cstdlib>
#include <cstring>

int main()
{
  boost::function<int(const char*)> f = std::atoi;
  std::cout << f("42") << '\n';
  f = std::strlen;
  std::cout << f("42") << '\n';
}
```

`boost::function` makes it possible to define a pointer to a function with a specific signature. Example 40.1 defines a pointer **f** that can point to functions that expect a parameter of type `const char*` and return a value of type `int`. Once defined, functions with matching signatures can be assigned to the pointer. Example 40.1 first assigns the function `std::atoi()` to **f** before `std::strlen()` is assigned to **f**.

Please note that types do not need to match exactly. Even though `std::strlen()` uses `std::size_t` as its return type, it can still be assigned to **f**.

Because **f** is a function pointer, the assigned function can be called using `operator()`. Depending on what function is currently assigned, either `std::atoi()` or `std::strlen()` is called.

If **f** is called without having a function assigned, an exception of type `boost::bad_function_call` is thrown (see Example 40.2).

Example 40.2. `boost::bad_function_call` thrown if `boost::function` is empty

```
#include <boost/function.hpp>
#include <iostream>

int main()
{
  try
  {
    boost::function<int(const char*)> f;
    f("");
  }
  catch (boost::bad_function_call &ex)
  {
    std::cerr << ex.what() << '\n';
```

```
  }
}
```

Note that assigning `nullptr` to a function pointer of type `boost::function` releases any currently assigned function. Calling it after it has been released will result in a `boost::bad_function_call` exception being thrown. To check whether or not a function pointer is currently assigned to a function, you can use the member functions `empty()` or `operator bool`.

It is also possible to assign class member functions to objects of type `boost::function` (see Example 40.3).

Example 40.3. Binding a class member function to `boost::function`

```
#include <boost/function.hpp>
#include <functional>
#include <iostream>

struct world
{
  void hello(std::ostream &os)
  {
    os << "Hello, world!\n";
  }
};

int main()
{
  boost::function<void(world*, std::ostream&)> f = &world::hello;
  world w;
  f(&w, std::ref(std::cout));
}
```

When calling such a function, the first parameter passed indicates the particular object for which the function is called. Therefore, the first parameter after the open parenthesis inside the template definition must be a pointer to that particular class. The remaining parameters denote the signature of the corresponding member function.