Chapter 21. Boost.Optional

The library Boost.Optional provides the class `boost::optional`, which can be used for optional return values. These are return values from functions that may not always return a result. Example 21.1 illustrates how optional return values are usually implemented without Boost.Optional.

Example 21.1. Special values to denote optional return values

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

int get_even_random_number()
{
  int i = std::rand();
  return (i % 2 == 0) ? i : -1;
}

int main()
{
  std::srand(static_cast<unsigned int>(std::time(0)));
  int i = get_even_random_number();
  if (i != -1)
    std::cout << std::sqrt(static_cast<float>(i)) << '\n';
}
```

Example 21.1 uses the function `get_even_random_number()`, which should return an even random number. It does this in a rather naive fashion by calling the function `std::rand()` from the standard library. If `std::rand()` generates an even random number, that number is returned by `get_even_random_number()`. If the generated random number is odd, -1 is returned.

In this example, -1 means that no even random number could be generated. Thus, `get_even_random_number()` can't guarantee that an even random number is returned. The return value is optional.

Many functions use special values like -1 to denote that no result can be returned. For example, the member function `find()` of the class `std::string` returns the special value **`std::string::npos`** if a substring can't be found. Functions whose return value is a pointer often return 0 to indicate that no result exists.

Boost.Optional provides `boost::optional`, which makes it possible to clearly mark optional return values.

Example 21.2. Optional return values with `boost::optional`

```
#include <boost/optional.hpp>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using boost::optional;

optional<int> get_even_random_number()
```

```
{
  int i = std::rand();
  return (i % 2 == 0) ? i : optional<int>{};
}

int main()
{
  std::srand(static_cast<unsigned int>(std::time(0)));
  optional<int> i = get_even_random_number();
  if (i)
    std::cout << std::sqrt(static_cast<float>(*i)) << '\n';
}
```

In Example 21.2 the return value of `get_even_random_number()` has a new type, `boost::optional<int>`. `boost::optional` is a template that must be instantiated with the actual type of the return value. `boost/optional.hpp` must be included for `boost::optional`.

If `get_even_random_number()` generates an even random number, the value is returned directly, automatically wrapped in an object of type `boost::optional<int>`, because `boost::optional` provides a non-exclusive constructor. If `get_even_random_number()` does not generate an even random number, an empty object of type `boost::optional<int>` is returned. The return value is created with a call to the default constructor.

`main()` checks whether **`i`** is empty. If it isn't empty, the number stored in **`i`** is accessed with `operator*`. `boost::optional` appears to work like a pointer. However, you should not think of `boost::optional` as a pointer because, for example, values in `boost::optional` are copied by the copy constructor while a pointer does not copy the value it points to.

Example 21.3. Other useful member functions of `boost::optional`

```
#include <boost/optional.hpp>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using boost::optional;

optional<int> get_even_random_number()
{
  int i = std::rand();
  return optional<int>{i % 2 == 0, i};
}

int main()
{
  std::srand(static_cast<unsigned int>(std::time(0)));
  optional<int> i = get_even_random_number();
  if (i.is_initialized())
    std::cout << std::sqrt(static_cast<float>(i.get())) << '\n';
}
```

Example 21.3 introduces other useful member functions of `boost::optional`. This class provides a special constructor that takes a condition as the first parameter. If the condition is true, an object of type `boost::optional` is initialized with the second parameter. If the condition is false, an empty object of type `boost::optional` is created. Example 21.3 uses this constructor in the function `get_even_random_number()`.

With `is_initialized()` you can check whether an object of type `boost::optional` is not empty. Boost.Optional speaks about initialized and uninitialized objects – hence, the name of the member function `is_initialized()`. The member function `get()` is equivalent to `operator*`.

Example 21.4. Various helper functions of Boost.Optional

```
#include <boost/optional.hpp>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace boost;

optional<int> get_even_random_number()
{
  int i = std::rand();
  return make_optional(i % 2 == 0, i);
}

int main()
{
  std::srand(static_cast<unsigned int>(std::time(0)));
  optional<int> i = get_even_random_number();
  double d = get_optional_value_or(i, 0);
  std::cout << std::sqrt(d) << '\n';
}
```

Boost.Optional provides free-standing helper functions such as `boost::make_optional()` and `boost::get_optional_value_or()` (see Example 21.4). `boost::make_optional()` can be called to create an object of type `boost::optional`. If you want a default value to be returned when `boost::optional` is empty, you can call `boost::get_optional_value_or()`.

The function `boost::get_optional_value_or()` is also provided as a member function of `boost::optional`. It is called `get_value_or()`.

Along with `boost/optional/optional_io.hpp`, Boost.Optional provides a header file with overloaded stream operators, which let you write objects of type `boost::optional` to, for example, standard output.