Chapter 23. Boost.Any

Strongly typed languages, such as C++, require that each variable have a specific type that defines what kind of information it can store. Other languages, such as JavaScript, allow developers to store any kind of information in a variable. For example, in JavaScript a single variable can contain a string, then a number, and afterwards a boolean value.

Boost.Any provides the class `boost::any` which, like JavaScript variables, can store arbitrary types of information.

Example 23.1. Using `boost::any`

```
#include <boost/any.hpp>

int main()
{
  boost::any a = 1;
  a = 3.14;
  a = true;
}
```

To use `boost::any`, include the header file `boost/any.hpp`. Objects of type `boost::any` can then be created to store arbitrary information. In Example 23.1, **a** stores an `int`, then a `double`, then a `bool`.

Variables of type `boost::any` are not completely unlimited in what they can store; there are some preconditions, albeit minimal ones. Any value stored in a variable of type `boost::any` must be copy-constructible. Thus, it is not possible to store a C array, since C arrays aren't copy-constructible.

To store a string, and not just a pointer to a C string, use `std::string` (see Example 23.2).

Example 23.2. Storing a string in `boost::any`

```
#include <boost/any.hpp>
#include <string>

int main()
{
  boost::any a = std::string{"Boost"};
}
```

To access the value of `boost::any` variables, use the cast operator `boost::any_cast` (see Example 23.3).

Example 23.3. Accessing values with `boost::any_cast`

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

int main()
{
  boost::any a = 1;
  std::cout << boost::any_cast<int>(a) << '\n';
  a = 3.14;
```

```
  std::cout << boost::any_cast<double>(a) << '\n';
  a = true;
  std::cout << std::boolalpha << boost::any_cast<bool>(a) << '\n';
}
```

By passing the appropriate type as a template parameter to `boost::any_cast`, the value of the variable is converted. If an invalid type is specified, an exception of type `boost::bad_any_cast` will be thrown.

Example 23.4. `boost::bad_any_cast` in case of an error

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

int main()
{
  try
  {
    boost::any a = 1;
    std::cout << boost::any_cast<float>(a) << '\n';
  }
  catch (boost::bad_any_cast &e)
  {
    std::cerr << e.what() << '\n';
  }
}
```

Example 23.4 throws an exception because the template parameter of type `float` does not match the type `int` stored in **a**. The program would also throw an exception if `short` or `long` were used as the template parameter.

Because `boost::bad_any_cast` is derived from `std::bad_cast`, `catch` handlers can catch exceptions of this type, too.

To check whether or not a variable of type `boost::any` contains information, use the member function `empty()`. To check the type of the stored information, use the member function `type()`.

Example 23.5. Checking type of currently stored value

```
#include <boost/any.hpp>
#include <typeinfo>
#include <iostream>

int main()
{
  boost::any a = 1;
  if (!a.empty())
  {
    const std::type_info &ti = a.type();
    std::cout << ti.name() << '\n';
  }
}
```

Example 23.5 uses both `empty()` and `type()`. While `empty()` returns a boolean value, the return value of `type()` is of type `std::type_info`, which is defined in the header file `typeinfo`.

Example 23.6 shows how to obtain a pointer to the value stored in a `boost::any` variable using `boost::any_cast`.

Example 23.6. Accessing values through a pointer

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

int main()
{
  boost::any a = 1;
  int *i = boost::any_cast<int>(&a);
  std::cout << *i << '\n';
}
```

You simply pass a pointer to a `boost::any` variable to `boost::any_cast`; the template parameter remains unchanged.