std::any

C++17 allows us to put our value in a safe container which can be accessed only when its type is specified. Welcome to std::any.

The new C++17 data types `std::any`, `std::optional`, and `std::variant` are all based on the Boost libraries.

std::any is a type-safe container for single values of any type which is copy-constructible. There are a few ways to create a `std::any` container `any`. You can use the various constructors or the factory function `std::make_any`. By using `any.emplace`, you directly construct one value into `any`. `any.reset` lets you destroy the contained object. If you want to know whether the container `any` has a value, use the method `any.has_value`. You can even get the typeid of the container object via `any.type`. Thanks to the generic function `std::any_cast` you have access to the contained object. If you specify the wrong type, you will get a `std::bad_any_cast` exception.

Here is a code snippet showing the basic usage of `std::any`.

```
#include <any>
#include <iostream>
#include <vector>


using namespace std;
struct MyClass{};

int main(){
  std::vector<std::any> anyVec{true, 2017, std::string("test"), 3.14,  MyClass()};
  std::cout << std::any_cast<bool>(anyVec[0]) << endl;                           // true
  int myInt= std::any_cast<int>(anyVec[1]);
  std::cout << myInt << std::endl << endl;                                       // 2017

  std::cout << anyVec[0].type().name() << endl;                                  // b
  std::cout << anyVec[1].type().name();                                  // i

  return 0;
}
```

std::any

The program snippet defines a `std::vector<std::any>`. To get one of its elements, you have to use `std::any_cast`. As mentioned, if you use the wrong type, you will get a `std::bad_any_cast exception`.

> **i The string representation of the typeid**
> The string representation of the typeid is implementation defined. If `anyVec[1]` is of type `int` the expression `anyVec[1].type().name()` will return `i` with the GCC C++ compiler and `int` with the Microsoft Visual C++ compiler.

`std::any` can have objects of arbitrary types; `std::optional` may or may not have a value.