any::thing you wanted to know about C++17 std::any

Amir Kirsh
Academic College of Tel-Aviv-Yaffo
Tel-Aviv University
Independent SW Consultant
(kirshamir at gmail com)

javascript *(pardon my French :-)*

```
var a = 3;    // Number

a = "hello"; // String

a = {x: "foo", y: 3.5}; // Object

a = function(msg){alert(msg);}; // function
```

Can we do it in C++?

Well, C++ is a Strictly Typed language…
So, ***<u>NO</u>***, right?

Can we do it with auto??

No!!

```
auto a = 3;   // int

a = "hello"; // compilation error
             // cannot bind const char* to int
```

boost::any

Invented by Kevlin Henney

(initially wanted to call the class “Henney” after his name but decided to go with “any”)

- **2000: presenting the idea http://www.two-sdg.demon.co.uk/curbralan/papers/ValuedConversions.pdf**

- **2001: added to Boost https://scicomp.ethz.ch/public/manual/Boost/1.55.0/any.pdf**

std::any

Added in C++17, based on boost::any and *almost* the same

Example:

```
std::any a = 3; // holding int

a = "hello";    // holding const char*

a = []{std::cout << "I'm a lambda"}; // now holding a lambda!
```

std::any - wait, unfortunately it’s not so easy...

```
std::any a = 3; // holding int

std::cout << a; // compilation error, std::cout cannot print std::any

std::cout << (int)a; // compilation error, almost..., but not yet

std::cout << std::any_cast<int>(a); // ok - prints 3
```

Note: you should know it’s an int!

can we overload on std::any type?

Unfortunately the type is “erased” and can be achieved in run-time only as `type_info` through the `type()` method of `std::any`

- **std::any is not templated**
- **the actual inner type is “well hidden” (= “erased”)**
- **you cannot overload on different types**
- **`type_info` retrieved through the `type()` method can be used in runtime code - e.g. in if statements, but not in compile time**

std::any - type()

```
void foo(std::any a) {
    if(!a.has_value()) {
       std::cout << "empty" << std::endl;
    }
    else if(a.type() == typeid(int)) {
       std::cout << "int: " << std::any_cast<int>(a) << std::endl;
    }
    else if(a.type() == typeid(const char*)) {
       std::cout << "const char*: "
                 << std::any_cast<const char*>(a) << std::endl;
    }
    else {
       std::cout << "unsupported type" << std::endl;
    }
}
```

let’s try std::any_cast with our *try_any_cast*

```
template<class T>
void try_any_cast(std::any a) {
  try {
    std::any_cast<T>(a);
    std::cout << "succeeded in casting std::any of "
              << type_to_console(a.type())
              << "to: " << type_to_console(typeid(T));
  }
  catch (const std::bad_any_cast& e) {
    std::cout << e.what() << ": tried to cast to "
              << type_to_console(typeid(T))
              << "BUT actual type is: " << type_to_console(a.type());
  }
}
```

http://coliru.stacked-crooked.com/a/c9c4cc487fe07e51

std::any - type() - int

`7 => std::any(7)`

```
int main() {
     try_any_cast<int>(7);
}
```

prints:
```
succeeded in casting std::any of int
to: int
```

std::any - type() - lambda

```
int main() {
     auto lambda = []{};
     try_any_cast<std::function<void(void)>>(lambda);
}
```

prints:

```
bad any_cast: tried to cast to a function<void(void)>
BUT actual type is: main::{lambda()#1}
```

This is not the fault of std::any_cast …
each lambda creates its own type
which is not an std::function
https://stackoverflow.com/a/20825525/2085626

std::any - type() - lambda again

```
int main() {
     std::function<void(void)> lambda = []{};
     try_any_cast<std::function<void(void)>>(lambda);
}
```

prints:

```
succeeded in casting std::any of std::function<void ()>
to: std::function<void ()>
```

std::any_cast - polymorphism?

```
struct A{
    virtual ~A(){}
};

struct B: public A{};

int main() {
  try_any_cast<A>(B());
}
```

prints:

```
bad any_cast: tried to cast to A
BUT actual type is: B
```

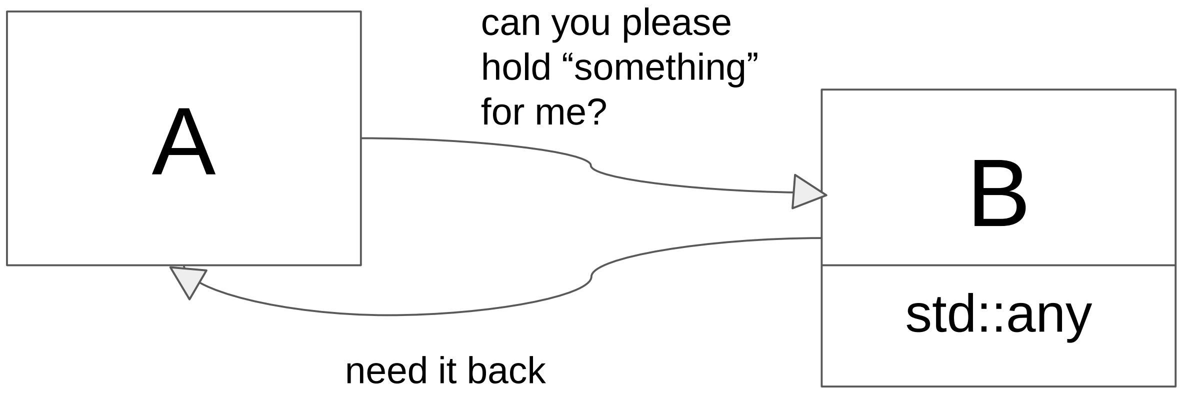
std::any_cast - int => long?

`7 => std::any(7)`

```
int main() {
   try_any_cast<long>(7);
}
```

prints:
bad any_cast: tried to cast to long
BUT actual type is: int

std::any - usage

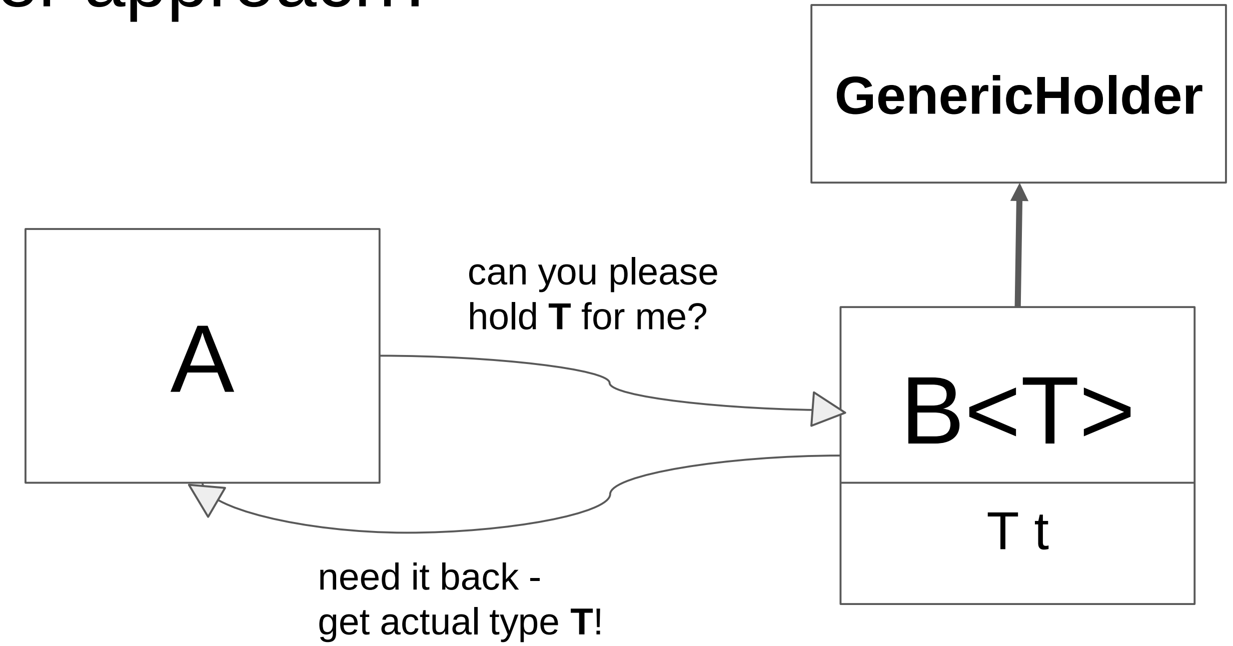


std::any - usage

https://stackoverflow.com/questions/52715219/when-should-i-use-stdany

> “ Use `std::any` where in the past you would have used `void*`. Which is to say, ideally, almost nowhere. ” – Richard Hodges

A better approach?



std::variant

Should know the possible options in compile time

Can “visit” the possible options

Should be more useful than std::any

Great C++ Core Meetup talk by Dvir Yitzchaki:
https://corecppil.github.io/Meetups/2018-05-28_Practical-C++Asio-Variant

std::any implementation

Erasure

```cpp
// a simplified proposed version of how std::any is implemented
class any {
    struct base_holder {
        virtual ~base_holder() {}
        virtual const std::type_info& type() const = 0;
                // ...
    };

    template < typename T >
    struct holder : base_holder {
        T value;
        holder(T t) : value(t) {}
        virtual const std::type_info& type() const override {
            return typeid(T);
        }
    };

// => continues next page
// class any continued
    base_holder * ptr = nullptr;
public:
    any()  {}
    ~any() { delete ptr; }

    template < typename T >
    any(T t) : ptr(new holder<T>(t)) {}

    any& operator=(any a) {
        std::swap(ptr, a.ptr);
        return *this;
    }

    const std::type_info& type() const { return ptr->type(); }

        // ...
};
```

The “erasure” happens here!
base_holder points to **holder<T>**
BUT - letting the type T “disappear”
Thus: *any* is not templated!

```
template<class T>
T any_cast(any& a) {
    if(a.type() != typeid(T)) {
        throw bad_any_cast();
    }
    return static_cast<any::holder<T>*>(a.ptr)->value;
}
```

Very poor runtime information available

http://coliru.stacked-crooked.com/a/e637a3571d74087a

Some links

https://en.cppreference.com/w/cpp/utility/any

https://scicomp.ethz.ch/public/manual/Boost/1.55.0/any.pdf

https://stackoverflow.com/questions/52715219/when-should-i-use-stdany

https://stackoverflow.com/questions/49428018/
why-doesnt-stdany-cast-support-implicit-conversion

https://blogs.msdn.microsoft.com/vcblog/2018/10/04/stdany-how-when-and-why

https://lists.boost.org/Archives/boost/2014/03/212154.php

Thank you!

```
void conclude(auto greetings) {
        while(still_time() && have_questions()) {
                ask();
        }
        greetings();
}

conclude([]{ std::cout << "Thank you!"; });
```