From SFINAE to Concepts, From C++98 to C++20 ... by (brief) Example

Mario Konrad

May 2019

Challenge

```
template <class T>
void func(T t)
{
    // ...
}
```

Does it work?

- with `int`?
- with `float`?
- with `struct foo {}`?
- differently for `int` and `float`?

$\Rightarrow$ it's a template, but...

Timeline

- C++98 : SFINAE (Substitution Failure Is Not An Error)
- C++11 : `<type_traits>`, default parameters for function templates
- C++17 : `std::void_t`
- C++20 : Concepts

Template Deduction

```
template <class T> T func(T a) { /* ... */ }
```

Deduction for type `int`:

```
int func(int a) { /* ... */ }
```

Substitution

If not possible for type `foo`, no deduction

Overload Set

```
struct foo {};
struct base {};
struct derived : base {};

void func(int)    { std::cout << "int"    << '\n'; }
void func(double) { std::cout << "double" << '\n'; }
void func(base)   { std::cout << "base"   << '\n'; }
void func(...)    { std::cout << "other"  << '\n'; }

func(10);
func(10.0);
func(foo{});
func(base{});
func(derived{});
```

Demo

Case 1

```
template <class T> void func(T);
```

- Different implementations for integers and floating point numbers

Case 2

```
template <class T> void func(T);
```

- Type `T` must provide member function: `int add(int, int)`

Exercise

Implement a search function template, similar to something like:

```
template < /* ... */ >
Iterator search(Iterator first,
                Iterator last,
                Element element)
```

which searches for the specified element within the boundaries given by the iterators and returns the iterator pointing to the element or `last` if not found.

Possible plot twists:

- provide specializations for *forward iterators* and *random access iterators*, assume range to be sorted
- element type has special properties vs. not

Resources

https://github.com/mariokonrad/cppug-sfinae-concepts