C++17: template<auto>

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

auto

C++11

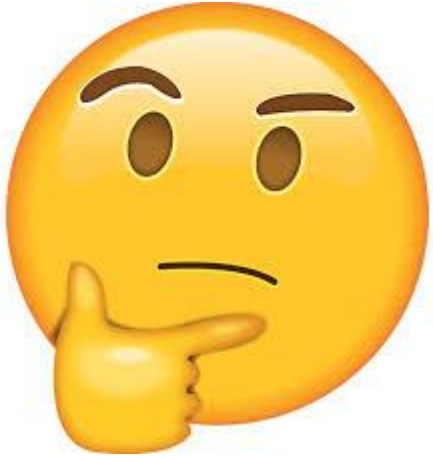
- variables, lambda parameters

C++14

- functions return type

C++17

- where can we add *auto*?



type and non-type template parameters

```
template<typename T, size_t SIZE>
class MyArray {
        T arr[SIZE];
public:
        // ...
};
```

Side Note:

Non-type template parameters can be:

- integral
- enum
- const pointers and references known at compile time (e.g. pointer to a function or to a variable with a static storage duration and linkage)

Variadic Templates - with <u>type</u> parameters

```
template<typename T>
T sum(const T& t) {
        return t;
}

template<typename T, typename... Ts>
std::common_type_t<T, Ts...> sum(const T& t, const Ts&... ts) {
        return t + sum(ts...);
}
```

we can mix types!

Call: `auto result = sum(1, 4.5, 2);`

Variadic Templates - with non-type parameters

```
template<int NUM, int... NUMs> struct Sum {
    int operator()()const {  return NUM + Sum<NUMs...>()(); }
};

// specialized version for single variable
template<int NUM> struct Sum<NUM> {
    int operator()()const {  return NUM; }
};
```

can we mix types?
e.g. Sum<1, ‘A’, 2>

Call: `auto result = Sum<1, 4, 2>()();`

Variadic Templates - mixing non-type parameters

Trying to mix types - make the type itself be part of the template:

`template<typename T,  T NUM>` -- **That's OK!!**

BUT:

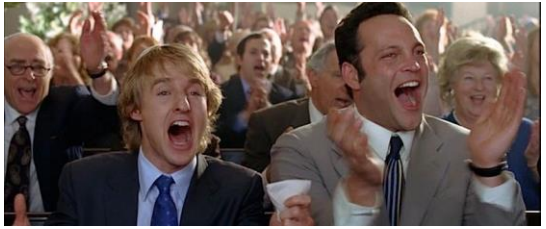
```
template<typename T, typename... Ts,  T NUM, Ts... NUMs>
```

Problem: **illegal** - template parameter pack must be the last template parameter

Ugly Solutions (pre C++17): Wrapping the Type and the Value into a new class

Variadic Templates - mixing non-type parameters

C++17:



```
template<auto NUM> struct Sum<NUM> {
    auto operator()()const { return NUM; }
};

template<auto NUM, auto... NUMs> struct Sum {
    auto operator()()const {    return NUM + Sum<NUMs...>()(); }
};
```

Call: `auto result = Sum<1, 'A', 2>()(); // => 68 -- hip hip hurray!`

But how did we use tuple before C++17?

Isn’t tuple mixing values of different types??

But how did we use tuple before C++17?

Tuple is based on <u>type</u> template parameters!

```
tuple<int, float> a = std::make_tuple(3, 4.5);
// OR better:
auto b = std::make_tuple(3, 4.5, "hello");
```

the actual values are NOT template parameters

-- side note, <u>another feature of C++17</u>:

```
std::tuple cpp17magic("C++", 17, "magic");
// ^ class template argument deduction
```

tuple is quite a complex creature by itself which may worth a talk…
usually we just *pass* variadic packs
but tuple actually *holds* it!

Wait… that’s not all

another usage for template<auto>

```
template<size_t SIZE>
class Foo {};
```

We want a **generic** function:

```
extractSize(const T<K>&) { return K; }
```

Usage example:

```
extractSize(Foo<6>());  // should be 6
```

extracting size before C++17

```
template<size_t SIZE>
class Foo {};

template < template<size_t> class T,  size_t  K >
auto extractSize(const T<K>&) { return K; }

int main() {
    std::cout << extractSize(Foo<6>());
}
```

using template template parameter
but can’t make it generic for ‘any type’...

extracting size before C++17 - this doesn’t work...

```
template<size_t SIZE>
class Foo {};

template <typename InnerType, template<InnerType> class T, InnerType K>
auto extractSize(const T<K>&) {
  return K;
}
```

Compilation Error (C++14): candidate template ignored: couldn't infer template argument 'InnerType'

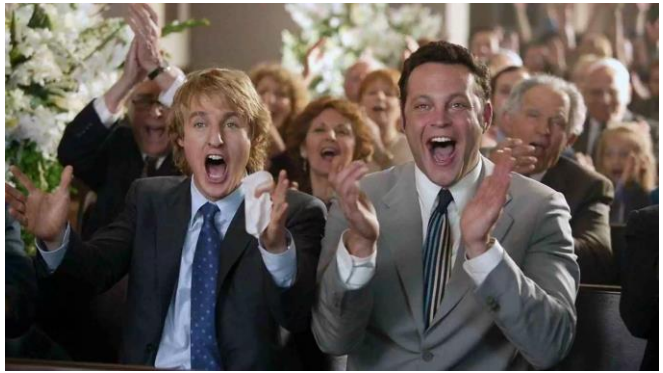
By the way, it does work with C++17, but with C++17 there is a nicer syntax =>

extracting size with C++17: template<auto>

```
template<size_t SIZE>
class Foo {};

template <template<auto> class T,  auto K>
auto extractSize(const T<K>&) { return K; }

int main() {
   std::cout << extractSize(Foo<6>()); // 6, of course
}
```



Some links

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0127r1.html

https://stackoverflow.com/questions/48608830/template-template-parameter-of-unknown-type

https://stackoverflow.com/questions/38026884/advantages-of-auto-in-template-parameters-in-c17

Thank you!

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

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