auto type deduction

Michael R. Nowak
Texas A&M University

This lecture follows the explanation given by Meyers, S. (2014). *Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14*. " O'Reilly Media, Inc.".

`auto`

- `auto` type deduction is template type deduction
  - There's slightly more that goes into template type deduction than what I've presented in class
  - Therefore, understand that I am presenting this at a high-level, so there's some hand waving going on

- Template type deduction involves templates and functions and parameters... but you don't see any of those in statements such as:

```
auto i = 11;
auto d = 3.14;
```

`auto`

- Instead, there is relationship defined between template type deduction and `auto` type deduction

- When using `auto` in the declaration of a variable or parameter, `auto` plays the role of `T` in the template

auto

- Recall that a function template, such as the following,

```
template<typename T> void foo(T param)
{
    // …
}
```

  can be instantiated via a function call,

```
foo(arg);
```

  leaving it up to the compiler to deduce the type of T

auto

- Given

  ```
  auto i = 11;
  ```

  compilers act as if there is a template for each declaration with `auto`, along with a call to that template with the value presented to the initializer

  ```
  // conceptual template for deducing i’s type
  template<typename T> void what_is_i(T param);

  // conceptual call where param’s deduced type is i’s type
  what_is_i(i);
  ```

auto

- auto works like template type deduction; however, there is one way in which they differ:

```
auto i1 = 11;       // deduced type is int
auto i2(11);        // deduced type is int
auto i3 = {11};     // deduced type is
                    // std::initializer_list<int>
auto i4{11};        // deduced type is
                    // std::initializer_list<int>
```

- auto treats braced initializer represents std::initializer_list<int> but template type deduction does not

References

Meyers, S. (2014). *Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14*. " O'Reilly Media, Inc.".