C++11

introduction to **`auto`**

auto keyword

- **`auto`** in C++11 is used used for deducing types for the initializer.
- **`int x = 123;`**
- **`auto x = 123;`** // auto is a placeholder for int
- When the variable has an initializer, then the type does not need to be explicitly stated. The compiler will do the type deduction.
- **`auto`** always requires an initializer. This is because the type deduced is of the initializer, *not* the variable.
- **`auto x; // compile time error`**

usefulness of **auto**

- **`auto`** becomes useful when writing the data type becomes harder and requires more typing.
- compiler does the magic to figure out the type for **`itr`**

```
std::map<std::string, uint32_t>::iterator itr
= m.begin();
```

```
auto itr = m.begin();
```

usefulness of **auto**

- **auto** tends to be more resilient to change
- if **vector** was to be changed to **list**, then the loop does not have to be rewritten.

```
for (vector<T>::iterator p = arg.begin(); p!=arg.end(); ++p)
     *p = 7;

for (auto p = arg.begin(); p!=arg.end(); ++p)
     *p = 7;
```

`auto` and range based **`for`** loop

- **`auto`** works nicely with the range based **`for`** loop.
- notice how the const and reference are specified.

```
void f(vector<int>& v) {
     for (const auto& x : v)
     {
      // ...
     }
} // x is a const int&
```

avoid type mismatches

- sometimes we take shortcuts like this:

```
uint32_t vsize = v.size();
```

- the above is incorrect because the correct data type is **`std::vector<int>::size_type`** which can actually be 32 or 64 bits depending on the OS underneath.

```
auto vsize = v.size();
```

- compiler will deduce the type correctly as **`std::vector<int>::size_type`**

readability

- there is a concern that auto does not allow the type to be known by just reading the code.
- this concern can be mitigated by writing well chosen variable names.

references

- Effective Modern C++ [Scott Myers]
- The C++ Programming Language (Fourth Edition) [Bjarne Stroustrup]