Modern C++: from `typedef` to `using`

Ronan Keryell (`ronan.keryell@amd.com`)
AMD AECG Research Labs, San José, California
2022/06/28

C and old C++ (pre-C++11): `typedef`-name

```
typedef double (*function_pointer)(float, int);

// Shortcut to avoid typing "struct" as "struct point"
typedef struct point { int x; int y; } point;
```

C++11 brought type-alias with `using`

```cpp
#include <iostream>
// We want a pointer to this function
double f(float a, int b) { return a + b; }

// C syntax
typedef double (*function_pointer)(float, int);
// C++11 syntax
using fp2 = double (*)(float, int);
// "using" works even with template
template <typename Return, typename T1, typename T2>
using fp3 = Return (*)(T1, T2);
// Simplify pointer syntax
template <typename T> using ptr = T*;
using fp4 = ptr<double(float, int)>;

int main() {
    // function_pointer p = f;
    // fp2 p = f;
    fp3<double, float, int> p { f };
    // fp4 p { f };
    std::cout << p(0.2, 1) << std::endl;
}
```

https://godbolt.org/z/3d3Y5zb9j

https://stackoverflow.com/questions/10747810/what-is-the-difference-between-typedef-and-using-in-c11

C++ Core Guidelines

`https://github.com/isocpp/CppCoreGuidelines`

> *"Within C++ is a smaller, simpler, safer language struggling to get out." – Bjarne Stroustrup*

The C++ Core Guidelines are a collaborative effort led by Bjarne Stroustrup, much like the C++ language itself. They are the result of many person-years of discussion and design across a number of organizations. Their design encourages general applicability and broad adoption but they can be freely copied and modified to meet your organization's needs.

- **T.43: Prefer using over typedef for defining aliases Reason**
  `https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#t43-prefer-using-over-typedef-for-defining-aliases`
  Improved readability: With `using`, the new name comes first rather than being embedded somewhere in a declaration. Generality: `using` can be used for template aliases, whereas `typedef`s can't easily be templates. Uniformity: `using` is syntactically similar to `auto`.
  - **Example** [...]
  - **Enforcement** Flag uses of `typedef`. This will give a lot of "hits" :-(

Automatic modernizing with tools like `clang-tidy`

- Tool to apply various rules, including some from C++ Core Guidelines
- `https://clang.llvm.org/extra/clang-tidy/checks/modernize/use-using.html`
- Probably integrated in your latest version of IDE or editor
- Part of `clangd` implementing LSP protocol used by various editors & editors

AMD

`typedef` still useful for C++ code compatible with C

https://github.com/Xilinx/XRT/blob/master/src/runtime_src/core/include/xcl_hwctx.h

```
#ifdef __cplusplus
# include <cstdint>
extern "C" {
#else
# if defined(__KERNEL__)
#  include <linux/types.h>
# else
#  include <stdint.h>
# endif
#endif

typedef uint32_t xcl_hwctx_handle;
typedef uint32_t xcl_qos_type;

#ifdef __cplusplus
}
#endif
```

AMD

Conclusion

- Follow the C++ Core Guidelines
  - `using` is more readable
  - `using` is more powerful
- `typedef` still useful when C compatibility is required
- Use tools to do the modernizing