— WRITTEN BY TRIANGLES ON AUGUST 22, 2018 • UPDATED ON SEPTEMBER 23, 2019 • ID 65 —

The differences between "using" and "typedef" in modern C++

Two ways of defining type aliases for a smarter code.

Since the beginning of C you can add *synonyms* to types that otherwise would be too complex or not much meaningful to work with. In a nutshell, you give an existing type, e.g. `int`, a new name, e.g. `Pixel`. Known as **type aliases**, they help you keep your code clean, short and understandable.

Say for example you are working with a graphical library. Compare the following two functions:

```
int getScreenWidth();

// Or ...

Pixel getScreenWidth();
```

The latter is clearly more intuitive: having declared the alias `Pixel` it is obvious what this function is about. Another example:

```
std::map<std::string, std::vector<std::string>> map;

// Or ...

Map map;
```

I think you get the picture.

However keep in mind that a type alias does not create a new type: it only generates a *synonym*, or another way of calling the underlying one. The alias `Pixel` is still an `int` and `Map` is still that frightening `std::map<std::string, std::vector<std::string>>` and they can be used with functions that accept `int`s and `std::map<std::string, std::vector<std::string>>`s as inputs.

Declaring new aliases

There are two ways of declaring new type aliases in modern C++. The first and traditional one is with the `typedef` keyword:

```
typedef [original-type] [your-alias];
```

For example:

```
typedef int Pixel;
typedef std::map<std::string, std::vector<std::string>> Map;
```

The other one, introduced in C++11, is with the `using` keyword:

```
using [your-alias] = [original-type];
```

For example

```
using Pixel = int;
using Map   = std::map<std::string, std::vector<std::string>>;
```

The result is identical: either way you will end up with new names `Pixel` and `Map` that you can use everywhere you need. But...

`using` works best with templates

The alias `Map` created in the two examples above (both with `typedef` and `using`) has its original type set in stone: it will always be a `std::map<std::string, std::vector<std::string>>` and there is no way to change it, for example into a `std::map<int, std::vector<int>>`, unless you don't declare a new alias with that type.

Fortunately the C++11's `using` has the ability to create the so-called **alias template**: an alias that keeps an open door to the underlying type. You can have the usual type aliasing *and* the ability to specify the template parameter(s) in the future.

This is how to declare an alias template:

```
template<[template-parameter-list]> using [your-alias] = [origir
```

For example:

```
template<typename T1, typename T2> using Map = std::map<T1, std
```

Now I can define new `Map` variables of different types:

```
// Actual type: std::map<std::string, std::vector<std::string>>
Map<std::string, std::string> map1;

// Actual type: std::map<int, std::vector<int>>
Map<int, int> map2;

// Actual type: std::map<int, std::vector<float>>
Map<int, float> map3;
```

Such behavior could be replicated with the traditional `typedef`, but it's way trickier and it's not worth it.

A template declaration cannot appear at block scope

You can put type alias declarations — both performed with `typedef` or `using` — everywhere you wish: in namespaces, in classes and inside blocks (i.e. between `{` and `}`).

Alias templates on the other hand follow the same rules of any other templated thing in C++: they cannot appear inside a block. They are actual template declarations, after all!

Sources

cprogramming.com - The Typedef Keyword in C and C++
cppreference.com - Type alias, alias template
cppreference.com - typedef specifier
StackOverflow - What is the difference between ’typedef’ and ’using’ in C++11?

C++11 • C++ • Templates

COMMENTS

fred on June 07, 2019 at 11:14

Thank you it was short but very clear