**Type aliases (typedef / using).**

A type alias is a different name by which a type can be identified. In C++, any valid type can be aliased so that it can be referred to with a different identifier.

In C++, there are two syntaxes for creating such type aliases: The first, inherited from the C language, uses the *typedef* keyword:

*typedef existing\_type new\_type\_name*;

where *existing\_type* is any type, either fundamental or compound, and *new\_type\_name* is an identifier with the new name given to the type.

For example:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | typedef char C;  typedef unsigned int WORD;  typedef char \* pChar;  typedef char field [50]; |  |

This defines four type aliases: *C*, *WORD*, *pChar*, and *field* as *char*, *unsigned int*, *char\** and *char[50]*, respectively. Once these aliases are defined, they can be used in any declaration just like any other valid type:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | C mychar, anotherchar, \*ptc1;  WORD myword;  pChar ptc2;  field name; |  |

More recently, a second syntax to define type aliases was introduced in the C++ language:

|  |  |  |
| --- | --- | --- |
|  | using new\_type\_name = existing\_type ; |  |

For example, the same type aliases as above could be defined as:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | using C = char;  using WORD = unsigned int;  using pChar = char \*;  using field = char [50]; |  |

Both aliases defined with *typedef* and aliases defined with *using* are semantically equivalent. The only difference being that *typedef* has certain limitations in the realm of templates that *using* has not. Therefore, *using* is more generic, although *typedef* has a longer history and is probably more common in existing code.

Note that neither *typedef* nor *using* create new distinct data types. They only create synonyms of existing types. That means that the type of *myword* above, declared with type *WORD*, can as well be considered of type *unsigned int*; it does not really matter, since both are actually referring to the same type.

Type aliases can be used to reduce the length of long or confusing type names, but they are most useful as tools to abstract programs from the underlying types they use. For example, by using an alias of *int* to refer to a particular kind of parameter instead of using *int* directly, it allows for the type to be easily replaced by *long* (or some other type) in a later version, without having to change every instance where it is used.