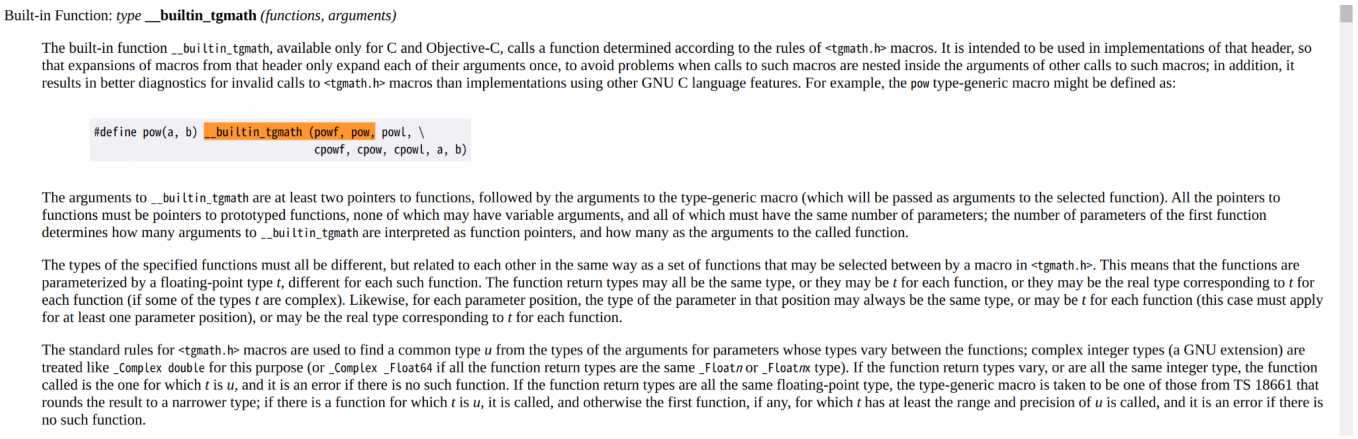
首先，我到GitHub上面找到了tgmath.h的原始碼
https://github.com/gcc-mirror/gcc/blob/master/gcc/ginclude/tgmath.h

我發現了一個很特別的地方，就是當我使用tgmath.h的 pow 時實際上是跑一個macro。
接著我到gcc gnu 官方document 看了該macro的定義
https://gcc.gnu.org/onlinedocs/gcc-8.3.0/gcc/Other-Builtins.html



Built-in Function: *type* **__builtin_tgmath** *(functions, arguments)*

The built-in function `__builtin_tgmath`, available only for C and Objective-C, calls a function determined according to the rules of `<tgmath.h>` macros. It is intended to be used in implementations of that header, so that expansions of macros from that header only expand each of their arguments once, to avoid problems when calls to such macros are nested inside the arguments of other calls to such macros; in addition, it results in better diagnostics for invalid calls to `<tgmath.h>` macros than implementations using other GNU C language features. For example, the `pow` type-generic macro might be defined as:

```
#define pow(a, b) __builtin_tgmath (powf, pow, powl, \
                                    cpowf, cpow, cpowl, a, b)
```

The arguments to `__builtin_tgmath` are at least two pointers to functions, followed by the arguments to the type-generic macro (which will be passed as arguments to the selected function). All the pointers to functions must be pointers to prototyped functions, none of which may have variable arguments, and all of which must have the same number of parameters; the number of parameters of the first function determines how many arguments to `__builtin_tgmath` are interpreted as function pointers, and how many as the arguments to the called function.

The types of the specified functions must all be different, but related to each other in the same way as a set of functions that may be selected between by a macro in `<tgmath.h>`. This means that the functions are parameterized by a floating-point type *t*, different for each such function. The function return types may all be the same type, or they may be *t* for each function, or they may be the real type corresponding to *t* for each function (if some of the types *t* are complex). Likewise, for each parameter position, the type of the parameter in that position may always be the same type, or may be *t* for each function (this case must apply for at least one parameter position), or may be the real type corresponding to *t* for each function.

The standard rules for `<tgmath.h>` macros are used to find a common type *u* from the types of the arguments for parameters whose types vary between the functions; complex integer types (a GNU extension) are treated like `_Complex double` for this purpose (or `_Complex _Float64` if all the function return types are the same `_Float`*n* or `_Float`*n*`x` type). If the function return types vary, or are all the same integer type, the function called is the one for which *t* is *u*, and it is an error if there is no such function. If the function return types are all the same floating-point type, the type-generic macro is taken to be one of those from TS 18661 that rounds the result to a narrower type; if there is a function for which *t* is *u*, it is called, and otherwise the first function, if any, for which *t* has at least the range and precision of *u* is called, and it is an error if there is no such function.

這個時候我就發現了tgmath.h 和 math.h 的pow 的差異

一個是macro 同時去根據規則選擇要用 powf pow powl cpowf cpow cpowl 其中一個的回傳 value
一個則是就一般的pow 一個double 進回傳 double 的 function

透過這個例子，我發現了tgmath.h厲害的地方，它會去識別parameter的型別並選擇究竟要回傳什麼樣的型別，不像math.h 要根據自己想要的型別去決定要 pow 還是 powl 還是 powf。