Reflections

A function would not be a good replacement for the safe macro because a function can’t postpone the evaluation of its parameters. The variable parameter could be a reference to an object with the interface closeable. Ergo the function in the parameter form has to be evaluated before the object can be closed. But the value from the function should also be returned, and the function has to be evaluated last for that to happen. But most importantly, the function also need the object defined in the parameter variable to work. A macro makes this possible. A function with the same structure would not behave as desired. The form parameter would not be able to be evaluated since it needs the object defined in the parameter variable. The macro in contrast does not evaluate its form parameter until after the object is defined and accessible.

It does not seem likely that a function would be a good replacement for the SQL-like macro, at least not if it's supposed to be called as in the assignment description. A specific evaluation order is required for the desired result, and some functions that's used in the macro body need the return values from other functions as parameters. Evaluation of the parameters need to be postponed for the SQL-like macro to function as desired. The return table can't for example be sorted before there is anything to sort. As mentioned earlier the function could not achieve the same result and have the same structure as the macro, because the evaluation of the parameter value can't be postponed in a function.

We have come to the conclusion that the benefit from using macros in these two assignments is the ability to control execution and therefore the ability to determine when and where code will be evaluated. Moreover, these two macros are a good example of when that benefit is useful.

Henrik Järnbrand	heja4977	Grupp5	PROP HT 2014
Tomas Sandberg	tosa3589	Grupp5	PROP HT 2014