CSCI3180 Asgn 3

Choi Man Kin

1155077469

There are three subs are based on dynamic scoping, *Server->task\_info()*, *Server->task\_attr()*, *Server->gpu\_info()*. Simply saying, if we want to use them, we need to have a “our” variable outside all subs, and a “local” variable inside the caller subs. Each time we call a dynamic scoping sub, Perl will first find whether the caller sub has the required local variable. If it does have, it will use it. If it does not have, it will find other sub following the calling stack.

To simplify my program and follow the specification, I declared a “local” variable before the calling of every dynamic scoping sub to minimize the scope involving dynamic scoping.

*Added these two “our” variables before all sub declaration*

**our** $task = **undef**;  
**our** $gpu = **undef**;

*Example 1*

**local** $task = $gpu->task();  
**print**($self->task\_info());

*Example 2*

**local** $task = $checking\_task; # $checking\_task is the “my” variable inside the sub  
**print**($self->task\_info());

There are two advantages. The first advantage is convenience. The caller variables will be exposed to the callee. The second advantage is shorter and simpler sub. If we need to achieve the same functionality with lexical scoping, the sub must be longer to receive the passed parameters.

The disadvantage is poor readability. Inside the callee, we don’t know whether the caller really has the required variable. If it does have, it is good. If it doesn’t have, it might finally find the global “our” variables. However, it is hard to determine whether it is the desired situation or just typo. It may cause more bugs. If we use lexical scoping, it is easier for the compiler to check the variable in its symbol table, and report to the programmer.

Another disadvantage is the impossibility to implement statically type checking. When a language support dynamic scoping, the compiler doesn’t know the calling sequence. It requires the programmer to manually check the errors, it may waste more time to debug.