**Task 1:**

Call-by-value and call-by-reference:

In call-by-reference, we pass the location where the objects are stored in the store as parameter. So, inside the procedure, we can directly access to this location and do some manipulations. However, in call-by-value, the value that is stored is passed as parameter. Although we can make calculations with this value, as we do not have a reference to where it is stored in store, we cannot manipulate the actual value in the store. Therefore, they can give different results sometimes.

The advantage of call-by-value is that it preserves the data by not changing the content of the variable. Therefore, whenever we call a function with call-by-value, we can be sure that the contents of the arguments will not change. Some disadvantages can be as follows:

* A variable can not be changed inside a procedure when we need to do it.
* When we pass the copy of the value of a variable, two copies of the variable exist in the program: one in the store and one inside the procedure, which makes it memory inefficient.

The advantage of call-by-reference is that values of the arguments can be changed inside the procedure, which can be useful. Also, it does not create a new copy for the variables passed into the functions. Therefore, it is more memory efficient. The disadvantages can be as follows:

* The value of the arguments can be changed inside the function, which can make the program harder to understand and follow.

Call-by-need and call-by-name:

In call-by-need, after evaluating a thunk associated with a variable for the first time, we update its value in the store by replacing thunk with the actual value of the expression. On the other hand, in call-by-name, even if a thunk is evaluated, its value is not updated in the store. When the program encounters the same variable again, thunk is evaluated again. In some cases, where the expression stored in the thunk depends on another variable, call-by-need and call-by-name can give different results. For example, if the expression depends on variable y, whose value is 5 at the beginning, in call-by-need, when we first see the expression, we evaluate it according to y = 5 and then update its value in the store. Let’s say we change the value of y after this to 6. When we encounter the expression a second time in the program, we return the value that we calculated when y = 5 as we do not evaluate the thunk again. If we would use call-by-name, as the thunk would be evaluated a second time, the result would be calculated according to y = 6. Therefore, the results will differ from each other.

The advantage of call-by-need is that as the value of the variable is updated after the thunk is evaluated, we do not evaluate the thunk expressions again and again. Therefore, it is faster than call-by-name.

The disadvantage of it is that it is more complicated with new side effects. Someone, who has no knowledge about the implementation can have hard time understanding the results of the program.

The advantage of call-by-name is that it can capture state changes. For example, in the case above, the resulting value of the expression has changed when the y value changed.

The disadvantage of call-by-name is that it recomputes the thunk associated with a variable each time that this variable is used. Therefore, it is less efficient then call-by-need.

**Task 2:**

The reason why these code pieces are needed is that in call-by-need, an expression corresponding to a variable or an argument are not evaluated until it is used at least once. For storing those expressions without evaluating, there is a need for a new data type, which is thunks. When the expression is evaluated for the first time, the value of the expression is updated in the store. So, the thunk is replaced with the actual value of the expression. When a variable is used in a piece of code, var-exp helps us to find its value in the environment. Therefore, we need to make some change in the evaluation of var-exp to handle with the evaluation of thunks and updating the values.

The first piece of code: This code should be implemented inside value-of procedure. When a new var-exp is evaluated with value-of procedure, this part of the code works. It first finds the reference associated with the variable in the environment, then it takes the value stored in the location pointed by the reference. If the value is an expressed value, it returns it because this means that this variable expression was already evaluated. If it is not an expressed value, this means that it is a thunk. Therefore, it is the first time that this variable expression is evaluated. The value of thunk is evaluated using the value-of thunk function and is stored in a local variable called val1. Then, after updating the value in the store corresponding this thunk with val1, val1 is returned.

The second piece of code: When value-of-thunk procedure is called with a thunk data typed object, it extracts the expression that was recorded in this thunk and the environment when this expression was created. Then, it evaluates the expression in the saved environment.

The changes given in the pdf should be done in the interp.scm file.