**Directed Work series N°4: linked lists**

**Exercise 5 (solution)**

**NB:** all the functions that we are going to define require the declaration of two other functions **First** and **Rest** which are defined as follows:

**functionFirst** (L: list): typeq ; // typeq : any type

**Begin**

Return (L ->ele );

**END ;**

**FunctionRest**(L: List): List

**Begin**

Return (L -> next);

**END ;**

1. **Sum** recursivefunction which returns the sum of the elements of a linked list:

**functionSum** (L: List): integer

**Begin**

**If** (L=Null )**then**

return(0);

**Else**

Return (first (L) + Sum (Rest(L) )) ;

**End if**

**END ;**

1. **Max** recursivefunction which returns the maximum of a linked list (we will not deal with the case of the empty list)

**functionMax** (L: List): integer

**Begin**

**If** (rest (L) =Null )**then**

Return (first (L));

**Else**

 **If** First (L) > max (Rest (L)) **then**

 Return (first (L));

 **Else**

 Return(max(Rest (L)));

**End if**

**End if**

**END**

1. **belongs** recursivefunction to check if an element exists in a list or not.

**Function** belongs (x: int, L: List): Boolean;

**Begin**

**If** (L=Null )**then**

Return( false);

**Else**

**if**( First (L) = x) **then**

 Return( true);

**Else**

 Return( belongs (x, Rest (L)));

**End if**

**End if**

**END;**

1. **nb\_occurrences recursive** function allows you to count the number of occurrences of a value x.

**Function** nb\_occurrences (x: integer, L: List): integer;

**Begin**

**If** (L=Null )**then**

return(0);

**Else**

**if**First (L) = x **then**

 Return( 1+ **nb\_occurrences** (x, Rest (L)));

**Else**

 Return(**nb\_occurrences** (x, Rest (L)));

**End if**

**End if**

**END ;**