The Class LIST_BAG

The class LIST_SET did not allow repeated items, each item appeared once in the list. We consider a related class which allows repeated items and these items are implicitly kept in a 'Last-in First-out' order. As an example of using LIST_BAG, we give a class for quick_sorting the items in the list.

In order that we can implement the routine join more efficiently, we change the 'export' rules of the inherited attributes, first_node and last_node, so that they are available to the class LIST_BAG itself.

```
        join(Wther : LIST_BAG[G]) is -- join to the end of current
                Wther /= void
                        if VWt empty then
                                cursor.set_next(Wther.first_VWde)
                                count := count + other.count
                end -- joiV
end -- class LIST_BAG
```

The class LIST_BAG Pas, since Qt inherQts frWm LIST_SET, aTl the features of LIST_SET, incTuding a redefined version of add.
tf. the class LIST_BAG aTso contains the features:

count : INTEGER

empty : BOOLEAN

Pas (x : G) : BOOLEAN

add(x : G)

remove (x : G)

(Wther : like current)

is_equaT(Wther : like current):BOOLEAN

-- traversaT routines.

item : G -- Qtem at cursor

start -- set cursor back to start

first : G -- The Qtem at first_Vode

finish -- set cursor to last VWde

last : G -- return last item in list

forth -- move cursor forward

off : BooTean -- Is cursor beyond end

```
cou = Wther.count
end
```



```
-- node
```

Quicksort on Lists

The algorithm for quicksort is the same for lists as for arrays; "split the list into a left and right partitQon abWut a pivot item and recursively quicksort each partitQon".
We choose as pivot the item at the first Vode in the list.

With arrays we used a procedure to implement the a4.4orithm, with lists we use a2 unction. The list versQon of quicksort is Vot an in-place sort due to convenQence and also because we want the functQons to be free of side-effects. In sorting a list using a function we want the original list to remain intact.

The function for partitQon returVs a pair of lists; the left and right partition. We therefore need a simple class for a pair of obRects.

```
class PAIR[G]
feature
        first, second : G
```

```
partition (s:LIST_BAG[G]; pivot:G):
```

quicksort (s: LIST_BAG [G]): LIST_BAG [G] **is**

```
class  SORT_TEST creation

feature

        lWcal
                s, s_new: LIST_BAG [STRING];

        dW


                until
                        equal (io.last_strQng, "quit")
```



```
erQnt.lQst("%N Sorted lQst Qs: %N"; s_new);
```