

# LO27 REPORT

---

## LO27 report

Manipulation of lists of Base-N integers, implementation of a Radix Sort derivate

---

Reference: LO27-F2015  
Version: 1.0  
Updated: 2015/12/21  
Status: Public

Authors: LUCAS LAZARE , MAXIME PINARD

This document describes the LO27 report project.

$\text{\TeX}$  is a trademark of the American Mathematical Society.

$\text{\texttt{tex-upmethodology}}$  is owned by Stéphane Galland, *Arakfiné.org*, France.

This document was realised with  $\text{\LaTeX}$  and  $\text{\texttt{tex-upmethodology}}$ .

Copyright © 2015 Lucas LAZARE & Maxime PINARD.

This document is published by the Université de Technologie de Belfort-Montbéliard. All right reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publishers.

Reference : LO27-F2015

# INTRODUCTION

In this document, we will present you `BaseNIntegerList`, and `BaseNIntegerListOfList` libraries. We'll also make a quick introduction to the `io` library.

*BaseNIntegerList* is a library implementing a doubly link list and functions to manage them.

*BaseNIntegerListOfList* is a library implementing methods for lists of `BaseNIntegerList` and their management.

*io* is a library, implementing input and output functions.

All these libraries were built in order to implement a radix sort solution.

In the first chapter, we will talk about the objectives of this project, and the problems we encountered.

In the second chapter, we will introduce the founded solutions, with their algorithms. Finally, in the third chapter, we will quickly talk about the `io` library



# CONTENTS

<b>1</b>	<b>Objectives and problem statements</b>	<b>7</b>
<b>2</b>	<b>Algorithms</b>	<b>9</b>
2.1	BaseNIntegerList . . . . .	9
2.2	BaseNIntegerListOfList . . . . .	19
<b>3</b>	<b>Input Output Library</b>	<b>21</b>
<b>4</b>	<b>Conclusion</b>	<b>23</b>



## OBJECTIVES AND PROBLEM STATEMENTS

The main objective of this project was to implement a library that could perform a Radix Sort. Radix sort is a algorithm used to sort numbers. It works as the following (with  $m$  the maximum number of digit in the elements of the list,  $b$  the base of the numbers)<sup>1</sup> :

- 1 let  $n$  equals 1, and make a ListOfList of  $B$  lists
- 2 Traverse all the list, watching for the  $n^{th}$  digit of each value, starting from the right.
- 3 Store each element in its corresponding List. If the watched digit was 1, it should go in the list 1.
- 4 Convert the list of list into a single list, enqueueing them.
- 5 if  $m$  is greater or equal to  $n$ , increase  $n$  by 1 and step back to 2.

---

<sup>1</sup>Note that this is a Least Significant Digit (LSD) Radix Sort variant





## ALGORITHMS

## 2.1/ BASENINTEGERLIST

*createIntegerList: Integer  $\rightarrow$  BaseNIntegerList*

Creates a new BaseNIntegerList for storing integers in the specified base.

```

1 function CreateIntegerList (base: integer): BaseNIntegerList
2 Begin
3 |     l: BaseNIntegerList
4 |     head(l) <- undefined
5 |     tail(l) <- undefined
6 |     base(l) <- base
7 |     size(l) <- 0
8 |
9 |     Createintegerlist <- l
10 End

```

#####

*isEmpty: BaseNIntegerList  $\rightarrow$  Boolean*

Returns true if the specified list is empty, false otherwise.

```

1 function IsEmpty (l: BaseNIntegerList): Boolean
2 Begin
3 |     if (size(l) = 0)
4 |     then
5 |         IsEmpty <- true
6 |     else
7 |         IsEmpty <- false
8 |     endif
9 Done

```

#####

*insertHead*:  $\text{BaseNIntegerList} \times \text{char}^* \rightarrow \text{BaseNIntegerList}$

Adds the specified integer ( $\text{char}^*$ , represented in the considered base) at the beginning of the specified list.

#### Overview

- Creates a new element
- Roots its following element to the actual head of the list
- Reroots the head of the list to it
- Increases the size of the list

```

1 function InsertHead (l: BaseNIntegerList , v: array<characters>):
    BaseNIntegerList
2 Begin
3 |   newel: ListElem*
4 |   newel <- alloc(ListElem)
5 |   value(newel) <- v
6 |   next(newel) <- head(l)
7 |   previous(newel) <- undefined
8 |
9 |   if (not IsEmpty(l))
10 |   then
11 |       previous(head(l)) <- newel
12 |   else
13 |       tail(l) <- newel
14 |   endif
15 |
16 |   head(l) <- newel
17 |   size(l) <- size(l) + 1
18 |
19 |   InsertHead <- l
20 Done

```

#####

*insertTail*:  $BaseNIntegerList \times char^* \rightarrow BaseNIntegerList$

Adds the specified integer (*char\**) at the end of the specified list.

#### Overview

- Creates a new element
- Roots its previous element to the actual tail of the list
- Reroots the tail of the list to it
- Increases the size of the list

```

1  function InsertTail (l: BaseNIntegerList, v: array<characters>):
      BaseNIntegerList
2  Begin
3  |   newel: ListElem*
4  |   newel <- alloc(ListElem)
5  |   value(newel) <- v
6  |   next(newel) <- undefined
7  |   previous(newel) <- tail(l)
8  |
9  |   if (not IsEmpty(l))
10 |   then
11 |       next(tail(l)) <- newel
12 |   else
13 |       head(l) <- newel
14 |   endif
15 |
16 |   tail(l) <- newel
17 |   size(l) <- size(l) + 1
18 |
19 |   InsertTail <- l
20 Done

```

#####

*removeHead*:  $BaseNIntegerList \times char^* \rightarrow BaseNIntegerList$

Removes the first element of the specified list.

**Overview** (assuming the list has more than one element)

- Reroots the head of the list to the second element
- Deletes the new head's previous element
- Reroots the head's previous element to nothing
- Decreases the size of the list

```

1 function RemoveHead(l: BaseNIntegerList): BaseNIntegerList
2 Begin
3   if (not IsEmpty(l))
4   then
5     if (size(l) = 1)
6     then
7       free(value(head(l)))
8       free(head(l))
9       head(l) <- undefined
10      tail(l) <- undefined
11      size(l) <- 0
12     else
13       head(l) <- next(head(l))
14       free(value(previous(head(l))))
15       free(previous(head(l)))
16       previous(head(l)) <- undefined
17       size(l) <- size(l) - 1
18     endif
19   endif
20
21   RemoveHead <- l
22 done

```

#####

*removeTail*:  $\text{BaseNIntegerList} \times \text{char}^* \rightarrow \text{BaseNIntegerList}$

Removes the last element of the specified list.

**Overview** (assuming the list has more than one element)

- Reroots the tail of the list to the second-to-last element
- Deletes the new tail's next element
- Reroots the tail's next element to nothing
- Decreases the size of the list

```

1 function RemoveTail(l: BaseNIntegerList): BaseNIntegerList
2 Begin
3 |   if (not IsEmpty(l))
4 |   then
5 |       if (size(l) = 1)
6 |       then
7 |           free(value(head(l)))
8 |           free(head(l))
9 |           head(l) <- undefined
10 |          tail(l) <- undefined
11 |          size(l) <- 0
12 |       else
13 |           tail(l) <- previous(tail(l))
14 |           free(value(next(tail(l))))
15 |           free(next(tail(l)))
16 |           next(tail(l)) <- undefined
17 |           size(l) <- size(l) - 1
18 |       endif
19 |   endif
20 |
21 |   RemoveTail <- l
22 done

```

#####

*deleteIntegerList*:  $\text{BaseNIntegerList} \rightarrow \emptyset$

Clears and deletes the specified BaseNIntegerList (free previously allocated memory).

```

1 procedure DeleteIntegerList (l: BaseNIntegerList*)
2 Begin
3 |   while (not IsEmpty(*l)) do
4 |       *l <- RemoveHead(*l)
5 |   done
6 Done

```

#####

*sumIntegerList*: *BaseNIntegerList*  $\rightarrow$  *char\**

Sums all the integers defined in the specified list using the Binary addition (base 2) and returns the corresponding results as an integer (*char\**) defined in the base of the list.

#### Overview

This function traverses the whole list, gradually summing each element.

```

1 function SumIntegerList(l: BaseNIntegerList): array<characters>
2 Begin
3 |   if (not IsEmpty(l))
4 |   then
5 |       element: ListElem*
6 |       element <- head(l)
7 |
8 |       if (size(l) = 1)
9 |       then
10 |           SumIntegerList <- copy(value(element))
11 |       else
12 |           s: array<characters>
13 |           tmp: array<characters>
14 |           s <- value(element)
15 |           element <- next(element)
16 |
17 |           do
18 |               s <- SumBase(s, element->value, l.base)
19 |               free(tmp)
20 |               tmp <- s
21 |               element <- next(element)
22 |           while (element != undefined)
23 |
24 |           Sumintegerlist <- s
25 |       endif
26 |   else
27 |       SumIntegerList <- undefined
28 |   endif
29 done

```

*SumBase(a,b,c)* returns the sum ( $a + b$ ), with *a* and *b* in base *c*

#####

*BaseToInt*:  $\text{char}^* \times \text{integer} \rightarrow \text{integer}$

Converts the value  $\text{char}^*$  (in the given base) into integer.

```

1  function BaseToInt (v: array<characters>, base: positive
    integer): positive integer
2  Begin
3  |   n, temp, size, i: positive integers
4  |   n <- 0
5  |   temp <- 1
6  |   size <- arraySize(v)
7  |
8  |   if (size > 0)
9  |   then
10 |       |   n <- GetValue(v[0])
11 |       |
12 |       |   for i from 1 to (size - 1) do
13 |       |       |   temp <- base * temp
14 |       |       |   n <- n + GetValue(v[i]) * temp
15 |       |       done
16 |       |
17 |   endif
18 |
19 |   BaseToInt <- n
20 Done

```

*GetValue*( $a$ ) returns the value represented by the character  $a$  (ie : ( $\text{GetValue}('F') = 16$ )

#####

*IntToBase*:  $integer \times integer \rightarrow char^*$

Converts the integer value into  $char^*$  in the given base.

```

1 function IntToBase(v: positive integer, base: positive integer):
    array<characters>
2 Begin
3 |   k, i: positive integers
4 |   w, base_digit: array<characters>
5 |
6 |   k <- 1
7 |   i <- base
8 |   base_digit <- {'0', '1', ..., '9', 'A', 'B', ..., 'Z'}
9 |
10 |  while (v >= i) do
11 |    |   i <- i*base
12 |    |   k <- k + 1
13 |  done
14 |
15 |  w <- alloc(k*characters)
16 |  w[0] <- '0'
17 |  k <- 0
18 |
19 |  while (v > 0) do
20 |    |   w[k] <- base_digit[v%base]
21 |    |   k <- k + 1
22 |    |   v <- v/base
23 |  done
24 |  IntToBase <- w
25 Done

```

#####

*ConvertBaseToBinary*:  $char^* \times integer \rightarrow char^*$

Converts the specied integer ( $char^*$ ) represented with the specified base (Integer, second parameter) into a corresponding binary integer (base 2).

```

1 function ConvertBaseToBinary (v: array<characters>, base:
    positive integer): array<characters>
2 Begin
3 |   ConvertBaseToBinary <- IntToBase(BaseToInt(v, base), 2)
4 Done

```

#####



*ConvertBinaryToBase*:  $\text{char}^* \times \text{integer} \rightarrow \text{char}^*$

Converts an integer represented using a binary base (base 2) into a corresponding integer represented with the specified base (Integer, second parameter).

```
1 function ConvertBinaryToBase (v: array<characters>, base:
   positive integer): array<characters>
2 Begin
3 |   ConvertBinaryToBase <- IntToBase(BaseToInt(v, 2), Base)
4 Done
```

#####

*SumBase*:  $\text{char}^* \times \text{char}^* \times \text{integer} \rightarrow \text{char}^*$

Sum two integer ( $\text{char}^*$ ) represented with the specified base (Integer, third parameter) and return the sum in the same base.

```

1  function SumBase(a: array<characters>, b: array<characters>,
   base: positive integer): array<characters>
2  Begin
3  |   i, j, k, a_len, b_len, remainder: integers
4  |   s, base_digits : array<characters>
5  |
6  |   i <- 0
7  |   j <- 0
8  |   k <- 0
9  |   a_len <- arraySize(a)
10 |   b_len <- arraySize(b)
11 |   base_digits <- { '0', '1', \ldots, '9', 'A', 'B', \ldots,
   'Z' }
12 |
13 |   while ((i < a_len) and (j < b_len)) do
14 |   |   remainder <- remainder + GetValue(a[i]) +
   GetValue(b[j])
15 |   |   s[k] <- base_digits[remainder\%base]
16 |   |   remainder <- remainder / base
17 |   |   k <- k + 1
18 |   |   i <- i + 1
19 |   |   j <- j + 1
20 |   done
21 |
22 |   while (i < a_len) do
23 |   |   remainder <- remainder + Getvalue(a[i])
24 |   |   s[k] <- base_digits[remainder\%base]
25 |   |   remainder <- remainder / base
26 |   |   k <- k+1
27 |   |   i <- i+1
28 |   done
29 |
30 |   while (j < b_len) do
31 |   |   remainder <- remainder + Getvalue(a[i])
32 |   |   s[k] <- base_digits[remainder\%base]
33 |   |   remainder <- remainder / base
34 |   |   k <- k+1
35 |   |   j <- j+1
36 |   done
37 |
38 |   if (remainder != 0)
39 |   then
40 |   |   s[k] <- base_digits[remainder]
41 |   endif
42 |
43 |   SumBase <- s
44 Done

```

*SumBase*:  $\text{char}^* \times \text{char}^* \times \text{integer} \rightarrow \text{char}^*$

Sum two integer (char\*) represented in binary base and return the sum in binary base.

**Similar to *SumBase***

## 2.2/ BASENINTEGERLISTOFLIST

*createBucketList*:  $\text{Integer} \rightarrow \text{BaseNIntegerListOfList}$

Creates a BaseNIntegerListOfList for storing list of integers in base N (N being the specified integer, first parameter).

```

1 function CreateBucketList (N: positive integer):
   BaseNIntegerListOfList
2 Begin
3 |   l: BaseNIntegerListOfList
4 |
5 |   base(l) <- N
6 |   list(l) <- array<BaseNIntegerList>[N] // From 0 to N-1
7 |
8 |   for i from 0 to (N - 1) do
9 |     |   list(l)[i] <- CreateIntegerList(N)
10 |   done
11 |
12 |   CreateBucketList <- l
13 Done

```

#####

*buildBucketList: Integer  $\rightarrow$  BaseNIntegerListOfList*

Builds a new BaseNIntegerListOfList according to the specified BaseNIntegerList and considering the specified digit position (rightmost).

```

1 function BuildBucketList (list: BaseNIntegerList, digit_pos:
   Integer): BaseNIntegerListOfList
2 Begin
3 |   list_of_list: BaseNIntegerListOfList
4 |   number: array<characters>
5 |
6 |   list_of_list <- CreateBucketList(base(list))
7 |   list_element <- head(list)
8 |
9 |   while (list_element != undefined) do
10 |       number <- copy(value(list_element))
11 |
12 |       if (arraySize(number) > digit_pos)
13 |       then
14 |           list_of_list <- AddIntegerIntoBucket
   (list_of_list, number, GetValue(number[digit_pos]))
15 |       else
16 |           list_of_list <- AddIntegerIntoBucket
   (list_of_list, number, 0)
17 |       endif
18 |
19 |       list_of_list <- next(list_of_list)
20 |   done
21 |
22 |   Buildbucketlist <- list_of_list
23 |
24 Done

```

#####

3

## INPUT OUTPUT LIBRARY



# 4

## CONCLUSION