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

Introduction

Contents

1

Objectives and problem statements

2

Algorithms

2.1/ BaseNIntegerList

createIntegerList: *Integer* → *BaseNIntegerList*
Creates a new BaseNIntegerList for storing integers in the specified base.

```
function CreateIntegerList (base: integer): BaseNIntegerList
Begin
|       l: BaseNIntegerList
|       head(l) <- undefined
|       tail(l) <- undefined
|       base(l) <- base
|       size(l) <- 0
|
|       Createintegerlist <- l
End
```

###############

isEmpty: *BaseNIntegerList* → *Boolean*
Returns true if the specified list is empty, false otherwise.

```
function IsEmpty (l: BaseNIntegerList): Boolean
Begin
|     if (size(l) = 0)
|     then
|     |     IsEmpty <- true
|     else
|     |     IsEmpty <- false
|     endif
Done
```

###############

insertHead: *BaseNIntegerList* × *char** → *BaseNIntegerList*
Adds the specified integer (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* × (char*) → *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* × (char*) → *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

```
function RemoveHead(l: BaseNIntegerList): BaseNIntegerList
Begin
|     if (not IsEmpty(l))
|     then
|     |     if (size(l) = 1)
|     |     then
|     |     |     free(value(head(l)))
|     |     |     free(head(l))
|     |     |     head(l) <- undefined
|     |     |     tail(l) <- undefined
|     |     |     size(l) <- 0
|     |     else
|     |     |     head(l) <- next(head(l))
|     |     |     free(value(previous(head(l))))
|     |     |     free(previous(head(l)))
|     |     |     previous(head(l)) <- undefined
|     |     |     size(l) <- size(l) - 1
|     |     endif
|     endif
|
|     RemoveHead <- l
done
```

###############

removeTail: *BaseNIntegerList* × (char*) → *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

```
function RemoveTail(l: BaseNIntegerList): BaseNIntegerList
Begin
|     if (not IsEmpty(l))
|     then
|     |     if (size(l) = 1)
|     |     then
|     |     |     free(value(head(l)))
|     |     |     free(head(l))
|     |     |     head(l) <- undefined
|     |     |     tail(l) <- undefined
|     |     |     size(l) <- 0
|     |     else
|     |     |     tail(l) <- previous(tail(l))
|     |     |     free(value(next(tail(l))))
|     |     |     free(next(tail(l)))
|     |     |     next(tail(l)) <- undefined
|     |     |     size(l) <- size(l) - 1
|     |     endif
|     endif
|
|     RemoveTail <- l
done
```

###############

deleteIntegerList: *BaseNIntegerList* → ∅
Clears and deletes the specified BaseNIntegerList (free previously allocated memory).

```
procedure DeleteIntegerList (l: BaseNIntegerList*)
Begin
|     while (not IsEmpty(*l)) do
|     |     *l <- RemoveHead(*l)
|     done
Done
```

###############

sumIntegerList: *BaseNIntegerList* → *char**
Sums all the intgers 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.

```
function SumIntegerList(l: BaseNIntegerList): array<characters>
Begin
|    if (not IsEmpty(l))
|    then
|    |    element: ListElem*
|    |    element <- head(l)
|    |
|    |    if (size(l) = 1)
|    |    then
|    |    |    SumIntegerList <- copy(value(element))
|    |    else
|    |    |    s: array<characters>
|    |    |    tmp: array<characters>
|    |    |    s <- value(element)
|    |    |    element <- next(element)
|    |    |
|    |    |    do
|    |    |    |    s <- SumBase(s, element->value, l.base)
|    |    |    |    free(tmp)
|    |    |    |    tmp <- s
|    |    |    |    element <- next(element)
|    |    |    while (element != undefined)
|    |    |
|    |    |    Sumintegerlist <- s
|    |    endif
|    else
|    |    SumIntegerList <- undefined
|    endif
done
```

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

BaseToInt: *char** × *integer* → *integer*
Converts the value char* (in the given base) into integer.

```
function BaseToInt (v: array<characters>, base: positive
   integer): positive integer
Begin
|     n, temp, size, i: positive integers
|     n <- 0
|     temp <- 1
|     size <- arraySize(v)
|
|     if (size > 0)
|     then
|     |     n <- GetValue(v[0])
|     |
|     |     for i from 1 to (size - 1) do
|     |     |     temp <- base * temp
|     |     |     n <- n + GetValue(v[0]) * temp
|     |     done
|     |
|     endif
|
|     BaseToInt <- n
Done
```

GetValue(a) returns the value represented by the character *a* (ie : *(GetValue('F')* = 16)
###############

IntToBase: *integer* × *integer* → *char**
Converts the integer value into char* in the given base.

```
function IntToBase(v: positive integer, base: positive integer):
    array<characters>
Begin
|     k, i: positive integers
|     w, base_digit: array<characters>
|
|     k <- 1
|     i <- base
|     base_digit <- {'0','1', ..., '9', 'A', 'B', ..., 'Z'}
|
|     while (v >= i) do
|     |     i <- i*base
|     |     k <- k + 1
|     done
|
|     w <- alloc(k*characters)
|     w[0] <- '0'
|     k <- 0
|
|     while (v > 0) do
|     |     w[k] <- base_digit[v%base]
|     |     k <- k + 1
|     |     v <- v/base
|     done
|     IntToBase <- w
Done
```

###############

ConvertBaseToBinary: *char** × *integer* → *char**
Converts the specied integer (char*) represented with the specified base (Integer, second parameter) into a corresponding binary integer (base 2).

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

###############

ConvertBinaryToBase: *char** × *integer* → *char**
Converts an integer represented using a binary base (base 2) into a corresponding integer represented with the specified base (Integer, second parameter).

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

###############

SumBase: *char** × *char** × *integer* → *char**
Sum two integer (char*) represented with the specified base (Integer, third parameter) and return the sum in the same base.

```
function SumBase(a: array<characters>, b: array<characters>,
   base: positive integer): array<characters>
Begin
|    i, j, k, a_len, b_len, remainder: integers
|    s, base_digits : array<characters>
|
|    i <- 0
|    j <- 0
|    k <- 0
|    a_len <- arraySize(a)
|    b_len <- arraySize(b)
|    base_digits <- { '0', '1', \ldots, '9', 'A', 'B', \ldots,
   'Z' }
|
|    while ((i < a_len) and (j < b_len)) do
|    |    remainder <- remainder + GetValue(a[i]) +
   GetValue(b[j])
|    |    s[k] <- base_digits[remainder\%base]
|    |    remainder <- remainder/base
|    |    k <- k + 1
|    |    i <- i + 1
|    |    j <- j + 1
|    done
|
|    while (i < a_len) do
|    |    remainder <- remainder + Getvalue(a[i])
|    |    s[k] <- base_digits[remainder\%base]
|    |    remainder <- remainder/base
|    |    k <- k+1
|    |    i <- i+1
|    done
|
|    while (j < b_len) do
|    |    remainder <- remainder + Getvalue(a[i])
|    |    s[k] <- base_digits[remainder\%base]
|    |    remainder <- remainder/base
|    |    k <- k+1
|    |    j <- j+1
|    done
|
|    if (remainder != 0)
|    then
|    |    s[k] <- base_digits[remainder]
|    endif
|
|    SumBase <- s
Done
```

SumBase: *char** × *char** × *integer* → *char**
Sum two integer (char*) represented in binary base and return the sum in binary base.
Similar to *SumBase*

2.2/ BaseNIntegerListOfList

3

Input Output Library

4

Conclusion