Documentation

Operations project Tiut Cristian, Group 917

# Problem statement

This project repesents an application that provides sevices for computations in different bases. The options the user may choose from are:

1. Addition of 2 numbers in a base
2. Subtraction of 2 numbers in a base
3. Multiplication of a number with a digit in a base

# Subalgorithm’s diagram

The appplication uses multiple subalgorithms for abstraction and to make the algorithms easier to understand. In this sense, we use algorithms for:

* UI
  + printing a number represented as a vector of integers
  + reading a number and trasmiting it through the output parameter as a vector of integers
  + printing the menu
  + getting the user input (choosing which operation to perform)
* manipulating numbers
  + comparing 2 numbers represented as vectors of integers
  + checking if a number belongs to the limits imposed by a given base
* the functionalities from the problem statement
  + algorithm for addition
  + algorithm for subtraction
  + algorithm for mulitplication

These algorithms are called by the logic in the diagram on the next page.

# Used data types

* vector<int> - used to represent the operands and the result of:
  + the addition
  + the subtraction
  + the multiplication (in this case it was used to represent the first operand and the result, the digit being an int)
* bool – used to represent the sign in case of the subtraction and different values returned by functions checking different properties
* int – for all the other data, such as bases, indexes, option chose by user etc.

# Important algorithms description

## The addition of 2 numbers in a base

### Input, output:

* input
  + base (int)
  + first term (const vector<int> &) – const refference used to not copy the whole vector each time the function is called
  + second term (const vector<int> &) – const refference used to not copy the whole vector each time the function is called
* output
  + the result (vector<int> &) – output parameter, passed by refference

### Pseudocode:

add (base, a, b, res):

clear(res)

if base 1:

print “Operation could not be performed. Base must be strictly

greater than 1.\n”

return

endIf

if not check\_valid\_vector(a, base) or not check\_valid\_vector(b, base):

print “Operation could not be performed. Operands' digits exceed base

limits.\n”

return

endIf

i 0

carry 0

while i < size(a) and i < size(b):

curr\_sum ai + bi + carry

push to res (curr\_sum mod base)

carry curr\_sum div base

i i+1

endWhile

while i < size(a):

curr\_sum ai + carry

push to res (curr\_sum mod base)

carry curr\_sum div base

i i+1

endWhile

while i < size(b):

curr\_sum bi + carry

push to res (curr\_sum mod base)

carry curr\_sum div base

i i+1

endWhile

if carry 0

push to res carry

endIf

endAdd

## The subtraction of 2 numbers in a base

### Input, output:

* input
  + base (int)
  + first term (const vector<int> &) – const refference used to not copy the whole vector each time the function is called
  + second term (const vector<int> &) – const refference used to not copy the whole vector each time the function is called
* output
  + the result (vector<int> &) – output parameter, passed by refference
  + the sign (bool) – 0 if the sign is and 1 if the sign is

### Pseudocode

sub (base, a, b, res, sgn)

{

clear(res)

if base 1:

print “Operation could not be performed. Base must be strictly

greater than 1.\n”

return

endIf

if not check\_valid\_vector(a, base) or not check\_valid\_vector(b, base):

print “Operation could not be performed. Operands' digits exceed base

limits.\n”

return

endIf

sgn 0

if cmp(a, b) 0:

resize res to size(a)

lenb size(b)

borrow 0;

for i 0, lenb-1:

resi ai - bi - borrow

borrow 0

if res[i] < 0:

resi resi + base

borrow 1

endIf

endFor

lena size(a)

reslenb alenb - borrow

for j lenb + 1, lena – 1:

resj aj

endFor

else

resize res to size(b)

sub (base, b, a, res, sgn)

sgn 1

endIf

reslen size(res)

if resreslen-1 = 0

resize res to (reslen - 1)

endIf

endSub

## The multiplication of a number by a digit in a base

### Input, output, preconditions, post-conditions

* input
  + base (int)
  + first operand (const vector<int> &) – const refference used to not copy the whole vector each time the function is called
  + digit (int)
* output
  + the result (vector<int> &) – output parameter, passed by refference

### 

### Pseudocode

mul (base, a, digit, res):

clear(res)

if base 1:

print “Operation could not be performed. Base must be strictly

greater than 1.\n”

return

endIf

if not check\_valid\_vector(a, base) or digit base:

print “Operation could not be performed. Operands' digits exceed base

limits.\n”

return

endIf

carry 0

for d in a:

tmp d \* digit + carry

push to res (tmp mod base)

carry tmp div base

endFor

if carry 0:

push to res (carry)

endIf

endMul

# Test data for the application

1. Testing addition:
2. Test1: (result = 103)

1

7

25

45

1. Test2: (result = 70)

1

10

25

45

1. Test3: (result = 6A)

1

16

25

45

1. Testing subtraction:
2. Test1: (result = -5)

2

10

5

10

1. Test2: (result = 5)

2

10

10

5

1. Test3: (result = 35)

2

6

44

5

1. Testing multiplication:
2. Test1: (result = 1536)

3

7

234

5

1. Test2: (result = 1170)

3

10

234

5

1. Test3: (result = B04)

3

16

234

5