Operations with large numbers

June 12, 2017

Student : Buşe-Dragomir Alexandru

Programming Techniques project, Summer 2017

Group I, Subgroup A
CEN 1.1

Introduction

Large numbers are numbers that are significantly larger than those ordinarily used in everyday life, for instance in simple counting or in monetary transactions. The term typically refers to large positive integers, or more generally, large positive real numbers, but it may also be used in other contexts.

Very large numbers often occur in fields such as mathematics, cosmology, cryptography, and statistical mechanics. Sometimes people refer to numbers as being "astronomically large". However, it is easy to mathematically define numbers that are much larger even than those used in astronomy.

Problem statement

Advanced assignments

Assignment number 5 :

A library for operations with large numbers. The library should provide at least addition, subtraction, multiplication, division and taking the square root.

One Centillion

1,000,000,000,000,000,000,000,000,000,000,000,000,000,0
00,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000
,000,000,000,000,000,000,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,000,000,000,000,000,000,0
00,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000
,000,000,000,000,000,000,000,000,000,000,000,000,000,00
0,000

The largest -illion with a name that is officially considered part of the English language

Pseudocode

Addition function

```
void adunare_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare suma)

        if lungime_numar1 > lungime_numar2
                lungime_mare <- lungime_numar1
                lungime_mica <- lungime_numar2

                for i <- lungime_mica + 1 to lungime_mare do
                        numar2[i] <- 0

        else
                if lungime_numar1 < lungime_numar2
                        lungime_mare <- lungime_numar2
                        lungime_mica <- lungime_numar1

                        for i <- lungime_mica + 1 to lungime_mare do
                                numar1[i] <- 0

                else
                        lungime_mare <- lungime_numar1
                        lungime_mica <- lungime_numar2

        suma[0] <- lungime_mare

        for i <- 1 to lungime_mare do
                suma[i] <- (numar1[i] + numar2[i] + transport) % 10
                transport <- (numar1[i] + numar2[i] + transport) / 10

        if (transport != 0)
                suma[i] <- transport
                suma[0] ++
```

Substraction function

```
void scadere_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare diferenta)

        diferenta[0] <- -1

        if numar1 < numar2
                print ERROR
                return

        if numar1 = numar2
                print 0
                return

        if numar1 > numar2
                lungime_mare <- lungime_numar1
                lungime_mica <- lungime_numar2

                for i <- lungime_mica + 1 to lungime_mare + 1 do
                        numar2[i] <- 0

                for i <- 1 to lungime_mare do

                        if numar1[i] < numar2[i]

                                if index_scadere = 0
                                        diferenta[i] <- (numar1[i] + 10) - numar2[i]
                                        index_scadere <- 1

                                if index_scadere = 1
                                        diferenta[i] <- (numar1[i] - 1 + 10) - numar2[i]
                                        index_scadere <- 1

                        if numar1[i] = numar2[i]

                                if index_scadere = 1
                                        diferenta[i] <- 9
                                        index_scadere <- 1

                                if index_scadere = 0
                                        diferenta[i] <- 0
                                        index_scadere <- 0

                        if numar1[i] > numar2[i]

                                if index_scadere = 1
                                        diferenta[i] <- (numar1[i] - 1) - numar2[i]
                                        index_scadere <- 0

                                if index_scadere = 0
                                        diferenta[i] <- numar1[i] - numar2[i]
                                        index_scadere <- 0

                for i <- lungime_mare to 1 do

                        if diferenta[i] = 0
                                numar_cifre ++

                        else
                             numar_cifre <- lungime_mare - numar_cifre
                             diferenta[0] <- numar_cifre
                             return
```

Multiplication function

```
void inmultire_numar_mare (numar_mare numar1, numar_mare numar2, numar_mare produs)

        for j <- 1 to lungime2 do
                current_value <- numar2[j]

                for i <- 1 to lungime1 do
                        auxiliar[i + count] <- auxiliar[i + count] + numar1[i] * current_value

                count ++


        for i <- 1 to lungime1 + count do
                produs[i] <- (auxiliar[i] + transport) % 10
                transport <- (auxiliar[i] + transport) / 10

        produs[0] <- lungime1 + count

        if produs[produs[0]] = 0
                produs[0] <- produs[0] - 1

        for i <- 1 to produs[0] do

                if produs[i] != 0
                        verificare <- 1


        if verificare = 0
                produs <- 0
```

Division function

```
int impartire_la_scalar (numar_mare cat, int scalar)

        verificare <- 0

        if scalar = 0
                return -1

        else
                for i <- lungime_numar to 1 do
                        rest <- 10 * rest + cat[i]
                        cat[i] <- rest / scalar
                        rest <- rest % scalar

                for i <- lungime_numar to 1

                        if cat[i] != 0
                                verificare <- 1

                        else
                                if cat[i] = 0 and verificare = 0
                                        cat[0] <- cat[0] - 1

                if verificare = 0
                        cat <- 0

        return rest
```

Square root function

```
void radical_numar_mare (numar_mare valoare, numar_mare radical)

        for i <- 0 to lungime_numar do
                inceput[i] <- 0
                sfarsit[i] <- 0
                mijloc[i] <- 0
                patrat[i] <- 0
                suma[i] <- 0
                auxiliar[i] <- 0

        if valoare[0] = 1 and valoare[1] = 0
                radical <- 0
                return

        else
                if valoare[0] = 1 and valoare[1] = 1
                        radical <- 1
                        return

                else
                        inceput <- 1
                        sfarsit <- valoare

                        while inceput <= sfarsit
                                mijloc = (inceput + sfarsit) / 2
                                patrat = mijloc * mijloc

                                if patrat = valoare
                                        radical = mijloc
                                        return;

                                else
                                        if patrat < valoare
                                                inceput <- mijloc + 1
                                                radical <- mijloc

                                        else
                                                if patrat > valoare
                                                        sfarsit <- mijloc - 1
```

Application design

In order to define and construct functions for the main operations that arise while handling large numbers, a new data type has been introduced : `numar_mare`

In fact, I used an array with a finite length (1000 memory locations) to store these numbers `----> typedef int numar_mare[1000]`

The header "functions.h" contains all the function prototypes :

- void `citire_numar_mare` (char auxiliar[1000], `numar_mare` valoare)
- void `afisare_numar_mare` (`numar_mare` valoare)
- void `initializare_cu_mic` (`numar_mare valoare_mare`, int `numar_mic`)
- void `initializare_cu_mare` (`numar_mare valoare_initiala`, `numar_mare valoare_noua`)
- int `comparare_numere_mari` (`numar_mare` numar1, `numar_mare` numar2)
- void `adunare_numere_mari` (`numar_mare` numar1, `numar_mare` numar2, `numar_mare` suma)
- void `scadere_numere_mari` (`numar_mare` numar1, `numar_mare` numar2, `numar_mare` diferenta)
- void `inmultire_scalar` (`numar_mare valoare`, int scalar, `numar_mare` produs)
- void textttinmultire_numar_mare (`numar_mare` numar1, `numar_mare` numar2, `numar_mare produs)`
- int `impartire_la_scalar` (`numar_mare` cat, int scalar)
- void `radical_numar_mare` (`numar_mare` valoare, `numar_mare` radical)

The "functions.c" file will contain all the functions used in order to work with large numbers.

The way in which every function is written and should be analysed is very simple. There are just a few steps :

1. The large numbers are saved in an array with a finite dimension. In the first position, array[0], we will save the length of the number and in the positions 1, 2, 3... array[0] every digit of the number. The only thing to keep in mind is that the number will be saved in a reversed order because, in this way, a lot of functions will be easier to implement.

2. Every operation, excluding the square root, is implemented in a natural way. As an example, let's say you want to add two large numbers. My code will do that by simulating the exact thing that each one of us does. Putting them one under another and making the sum of their digits, one by one, from the end towards the most significant one. There is also a transport variable that carries a value equal to the last sum of two digits / 10 and adds it to the current sum. In the case of the square root, we use a binary search algorithm and replace the trivial operations with functions that do the same thing, but on large numbers.

3. We work with positive values so, for example, if we want to compute the difference of two numbers, we must consider the first number greater than or equal to the second (omitting this aspect will be signaled by an ERROR message).

We are talking about natural numbers that have from 1 to 999 digits (we can change their length by making the array that stands behind the data type greater, but I considered that this dimension was enough for my functions).

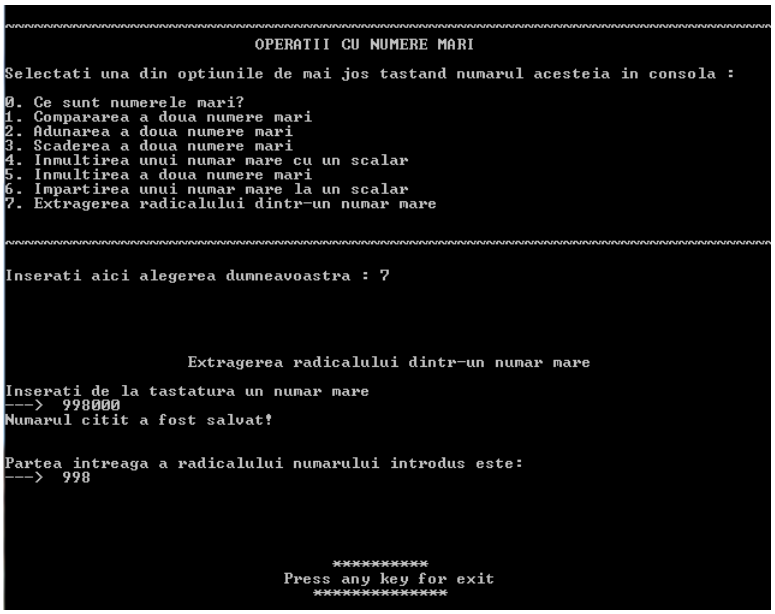
Here are all the functions used:

```
1. citire_numar_mare( )
void citire_numar_mare(char auxiliar[1000], numar_mare valoare)

Parameters
auxiliar -> The number represented as a string
valoare  -> The value of the number represented as a vector

This function will get a string and save it as a large number.



2. afisare_numar_mare( )
void afisare_numar_mare (numar_mare valoare)

Parameters
valoare -> The value of the number that will be printed

This function will get a large number and print its value.



3. initializare_cu_mic ( )
void initializare_cu_mic (numar_mare valoare_mare, int numar_mic)

Parameters
valoare_mare -> The large number that will be replaced
numar_mic    -> The new integer value of the large number

This function will replace the value of the given large number with a scalar value.



4. initializare_cu_mare ( )
void initializare_cu_mare (numar_mare valoare_initiala, numar_mare valoare_noua)

Parameters
valoare_initiala ->     The large number that will be replaced
valoare_noua     ->     The new value of the large number

This function will replace the value of the given large number with the value of a
secondary large number.
```

```
5. comparare_numere_mari ( )
int comparare_numere_mari (numar_mare numar1, numar_mare numar2)

Parameters
numar1 -> The first large number
numar2 -> The second large number

This function will compare the two large numbers.



6. adunare_numere_mari ( )
void adunare_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare suma)

Parameters
numar1 -> The first large number
numar2 -> The second large number
suma   -> The resulting sum of the two large numbers

This function will calculate the sum of two large numbers.



7. scadere_numere_mari ( )
void scadere_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare diferenta)

Parameters
numar1    -> The first large number
numar2    -> The second large number
diferenta -> The resulting difference of the two large numbers

This function will calculate the difference of two large numbers.



8. inmultire_scalar ( )
void inmultire_scalar (numar_mare valoare, int scalar, numar_mare produs)

Parameters
valoare -> The value of the large number
scalar  -> The value of the scalar
produs  -> The resulting product of the numbers

This function will calculate the product between a large number and a scalar value.



9. inmultire_numar_mare ( )
void inmultire_numar_mare (numar_mare numar1, numar_mare numar2, numar_mare produs)

Parameters
numar1 -> The first large number
numar2 -> The second large number
produs -> The resulting product of the two large numbers

This function will calculate the product of two large numbers.



10. impartire_la_scalar ( )
int impartire_la_scalar (numar_mare cat, int scalar)

Parameters
cat    -> The large dividend (we will also save the resulting quotient in it)
scalar -> The scalar value which we will divide by (the divisor)

This function will divide a large number by a scalar value and return the remainder.
The quotient will be saved in the original large number ("cat").



11. radical_numar_mare ( )
void radical_numar_mare (numar_mare valoare, numar_mare radical)

Parameters
valoare -> The large number value
radical -> The floor of the square root of the given number

This function will calculate the floor of the square root of a large number.
```

The "main.c" file will use all of the above functions and incorporate them into a basic application that will provide information about large numbers and a list of possible commands like adding two numbers, extracting the square root, calculating a difference or a product etc.

So, by running the main program, the user will have access to an easy to use and friendly command panel and will be able to get the result of some basic operations that involve large numbers and their manipulation.



The command panel generated by the main function

Source code - *functions.h*

```
typedef int numar_mare[1000];
void citire_numar_mare (char auxiliar[1000], numar_mare valoare);
void afisare_numar_mare (numar_mare valoare);
void initializare_cu_mic (numar_mare valoare_mare, int numar_mic);
void initializare_cu_mare (numar_mare valoare_initiala, numar_mare valoare_noua);
int comparare_numere_mari (numar_mare numar1, numar_mare numar2);
void adunare_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare suma);
void scadere_numere_mari (numar_mare numar1, numar_mare numar2, numar_mare diferenta);
void inmultire_scalar (numar_mare valoare, int scalar, numar_mare produs);
void inmultire_numar_mare (numar_mare numar1, numar_mare numar2, numar_mare produs);
int impartire_la_scalar (numar_mare cat, int scalar);
void radical_numar_mare (numar_mare valoare, numar_mare radical);
```

Source code - *main.c*

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <string.h>
# include "functions.h"

typedef int numar_mare[1000];

numar_mare X;
numar_mare Y;
numar_mare suma;
numar_mare diferenta;
numar_mare produs;
numar_mare cat;
numar_mare radical;

char numar_citit[1000];


int main ()
{

        int i;
        int scalar_dividere;
        int numar_mic;
        int scalar_produs;
        int rest = 0;
        int alegere_consola;
        int verificare = 0;

        char caracter;


        printf ("\n\n\n\n\n\n\n");
        printf ("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printf ("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printf ("                                  OPERATII CU NUMERE MARI\n\n");
        printf ("Selectati una din optiunile de mai jos tastand numarul);
        printf (" acesteia in consola : \n\n");
        printf ("0. Ce sunt numerele mari? \n");
        printf ("1. Compararea a doua numere mari \n");
        printf ("2. Adunarea a doua numere mari \n");
        printf ("3. Scaderea a doua numere mari \n");
        printf ("4. Inmultirea unui numar mare cu un scalar \n");
        printf ("5. Inmultirea a doua numere mari \n");
        printf ("6. Impartirea unui numar mare la un scalar \n");
        printf ("7. Extragerea radicalului dintr-un numar mare \n\n\n");
        printf ("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printf ("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
        printf ("Inserati aici alegerea dumneavoastra : ");
        scanf ("%d", &alegere_consola);
        printf ("\n\n");


        if (alegere_consola == 0)
        {
                printf ("\n\n\n");
                printf ("Numerele mari\n\n");
                printf ("Numere intregi pozitive de");
                printf (" dimensiuni foarte mari.\n");
                printf ("Lungimea lor va fi limitata la 999");
                printf (" de cifre in acest algoritm.\n\n");
                printf ("Observatie : Se va considera valid");
                printf (" ca numar mare orice numar cu dimensiunea\n");
                printf ("intre 1 si 999 de cifre.\n\n");
                printf ("!ATENTIE!\nIn cazul unor operatii
                precum inmultirea, trebuie luata in considerare");
                printf ("\ndimensiunea finala a rezultatului.\n");
                printf ("Se recomanda introducerea unor numere de");
                printf ("1-500 cifre pentru a fi evitate\n");
                printf ("eventualele erori de calcul.\n");
                printf ("Daca se doreste lucrul cu numere mai mari,");
                printf (" acest lucru se poate face prin\n");
                printf ("marirea lungimii vectorului din spatele");
                printf (" tipului de date numere_mari.\n\n\n\n");
                verificare = 1;
        }
```

```
if (alegere_consola == 1)
{
        printf ("\n\n\n");
        printf ("Compararea numerelor mari\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, X);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("Inserati de la tastatura inca un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, Y);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

                if (comparare_numere_mari(X, Y) == 0)
        {
                printf ("Cele doua numere sunt egale!\n\n");
        }

        else
        {
                if (comparare_numere_mari(X, Y) == 1)
                {
                        printf ("Primul numar este mai mare decat al doilea!\n\n");
                }

                else
                {
                        if (comparare_numere_mari (X, Y) == 2)
                        {
                                printf ("Al doilea numar este mai mare decat primul!\n\n");
                        }
                }
        }

        verificare = 1;
}


if (alegere_consola == 2)
{
        printf ("\n\n\n");
        printf ("Adunarea numerelor mari\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, X);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("Inserati de la tastatura inca un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, Y);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("Suma celor doua numere este :\n");
        adunare_numere_mari (X, Y, suma);
        printf ("--->  ");
        afisare_numar_mare (suma);
        printf ("\n\n\n");

        verificare = 1;
}


if (alegere_consola == 3)
{
        printf ("\n\n\n");
        printf ("Scaderea numerelor mari\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, X);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");
```

```
        printf ("Inserati de la tastatura inca un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, Y);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        scadere_numere_mari (X, Y, diferenta);
        printf ("\n");

        if (diferenta[0] != -1)
        {
                printf ("Diferenta celor doua numere este :\n");
                printf ("--->  ");
                afisare_numar_mare (diferenta);
                printf ("\n\n\n");
        }

        verificare = 1;
}


if (alegere_consola == 4)
{
        printf ("\n\n\n");
        printf (" Inmultirea unui numar mare cu un scalar\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, X);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("Inserati acum un scalar\n");
        printf ("--->  ");
        scanf ("%d", &scalar_produs);
        printf ("\nProdusul celor doua numere este : \n");
        printf ("--->  ");
        inmultire_scalar (X, scalar_produs, produs);
        afisare_numar_mare (produs);
        printf ("\n\n\n");

        verificare = 1;
}


if (alegere_consola == 5)
{
        printf ("\n\n\n");
        printf (" Inmultirea a doua numere mari\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, X);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("Inserati de la tastatura inca un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
        citire_numar_mare (numar_citit, Y);
        printf ("Numarul citit a fost salvat!");
        printf ("\n\n\n");

        printf ("\nProdusul celor doua numere este :\n");
        printf ("--->  ");
        inmultire_numar_mare (X, Y, produs);
        afisare_numar_mare (produs);
        printf ("\n\n\n");


        verificare = 1;
}


if (alegere_consola == 6)
{
        printf ("\n\n\n");
        printf ("Impartirea unui numar mare la un scalar\n\n");
        printf ("Inserati de la tastatura un numar mare\n");
        printf ("--->  ");
        scanf ("%s", numar_citit);
```

```
                citire_numar_mare (numar_citit, X);
                printf ("Numarul citit a fost salvat!");
                printf ("\n\n\n");

                printf ("Inserati de la tastatura un scalar\n");
                printf ("--->  ");
                scanf ("%d", &scalar_dividere);
                rest = impartire_la_scalar (X, scalar_dividere);

                if (rest == -1)
                {
                        printf ("\n\n Eroare!\n");
                        printf ("  !S-a efectuat impartirea la 0!");
                        printf ("\n\n\n");
                }

                else
                {
                        printf ("\nCatul impartirii este  ");
                        printf ("--->  ");
                        afisare_numar_mare (X);
                        printf ("\nRestul impartirii este  ---> %d", rest);
                        printf ("\n\n\n");
                }

                verificare = 1;
        }


        if (alegere_consola == 7)
        {
                printf ("\n\n\n");
                printf ("  Extragerea radicalului dintr-un numar mare\n\n");
                printf ("Inserati de la tastatura un numar mare\n");
                printf ("--->  ");
                scanf ("%s", numar_citit);
                citire_numar_mare (numar_citit, X);
                printf ("Numarul citit a fost salvat!");
                printf ("\n\n\n");

                printf ("Partea intreaga a radicalului numarului introdus este:\n");
                printf ("--->  ");
                radical_numar_mare (X, radical);
                afisare_numar_mare (radical);
                printf ("\n\n\n");

                verificare = 1;
        }


        if (verificare == 0)
        {
                printf ("\n");
                printf ("---> Numarul introdus nu se regaseste in lista de mai sus.\n");
                printf ("---> Inserati una din valorile 0,1,2,3,4,5,6,7.");
                printf ("\n\n\n");
        }

        else
        {
                printf ("\n\n\n                                          **********\n");
                printf ("                                    Press any key for exit\n");
                printf ("                                       **************");
                printf ("\n\n\n\n");
        }

        return 0;
}
```

Because the space does not allow it, in order to check the full source code (functions.c), visit my GitHub account by checking the hyperlink : Source code for functions

If the hyperlink does not work, you can follow the link :

https://github.com/Xerophyt/Functions-for-operations-with-large-numbers/blob/master/JustFunctions.c/functions.c

Experiments and results

In order to test the program, I used a random number generator that takes as an input a length and returns a random value having the corresponding number of digits.

This is the source code :

```
# include <stdlib.h>
# include <stdio.h>

void number_generator (size_t length)
{
        int i;
        int first_digit;

        srand(time(NULL));

        if (length > 1)
        {
                first_digit = rand() % 9 + 1;

                printf ("%d", first_digit);

                for(i = 2; i <= length; i ++)
                {
                        printf ("%d", rand() % 10);
                }
        }

        else
        {
                if (length == 1)
                {
                        printf ("%d", rand() % 10);
                }
        }


}

int main ()
{
        int length;

        printf ("Inserati numarul de cifre ->  ");

        scanf ("%d", &length);

        number_generator (length);

        return 0;
}
```

Here I will insert 10 input sets and the results generated by using the main functions (addition, subtraction, multiplication, dividing and extracting the square root).

For the last operation, we will provide the square root of the first number (we read two numbers in order to use the other functions). In fact, it computes the floor of the square root of the given number because the values that we work with are natural numbers.

The inputs will be all generated using the above number generator function. I will start with small numbers and I will increase their length gradually in order to demonstrate that by saving the numbers in this way, we can operate with any finite number, being it one digit long or 100 digits long.

The algorithms being homogeneous, (the same rules of calculation being applied on smaller numbers, as well as on very large ones) I will try to take decent values because testing the correctness of multiplying two numbers that have more than 100 digits each can take a very long time (100 x 100 = 10.000 intermediary products by hand calculation).

Test 1
Number 1 : 6
Number 2 : 4
Sum : 10
Difference : 2
Product : 24
Quotient : 1
Remainder : 2
Floor of square root : 2

Test 2
Number 1 : 27
Number 2 : 93
Sum : 120
Difference : ERROR (first number must be at least equal to the second one)
Product : 2511
Quotient : 0
Remainder : 27
Floor of square root : 5

Test 3
Number 1 : 446
Number 2 : 59
Sum : 505
Difference : 387
Product : 26314
Quotient : 7
Remainder : 33
Floor of square root : 21

Test 4
Number 1 : 211
Number 2 : 1381
Sum : 1592
Difference : ERROR (first number must be at least equal to the second one)
Product : 291391
Quotient : 0
Remainder : 211
Floor of square root : 14

Test 5
Number 1 : 2908
Number 2 : 1973
Sum : 4881
Difference : 935
Product : 5737484
Quotient : 1
Remainder : 935
Floor of square root : 53

Test 6
Number 1 : 91609
Number 2 : 900
Sum : 92509
Difference : 90709
Product : 82448100
Quotient : 101
Remainder : 709
Floor of square root : 302

Test 7
Number 1 : 523002
Number 2 : 0
Sum : 523002
Difference : 523002
Product : 0
Quotient : Error message (dividing by 0)
Remainder : Error message (dividing by 0)
Floor of square root : 723

Test 8
Number 1 : 1608255
Number 2 : 4957
Sum : 1613212
Difference : 1603298
Product : 7972120035
Quotient : 324
Remainder : 2187
Floor of square root : 1268

Test 9
Number 1 : 19018809
Number 2 : 511176
Sum : 19529985
Difference : 18507633
Product : 9721958709384
Quotient : 37
Remainder : 105297
Floor of square root : 4361

Test 10
Number 1 : 9967604506
Number 2 : 103475090
Sum : 10071079596
Difference : 9864129416
Product : 1031398773342755540
Quotient : 96
Remainder : 33995866
Floor of square root : 99837

Conclusions

For me, the implementation of this project has been an opportunity to test my level of preparation and the degree in which I can apply the C language in solving more complex problems.

At the same time, it was a good way to improve my methods of processing and editing documents in LaTeX. I also understood the importance of using Doxygen in order to get a well structured project documentation.

It was an exercise of correct code indentation and function management using makefiles, combined with the use of a gcc compiler for a command prompt application.

I enjoyed developing the source code and analyzing the situations that arose in the process because these have shown me once more the undeniable link that connects mathematics and programming.

I will finish by saying that the process of assembling all the project parts into one application and testing it has represented a very good practice for my long term growth associated to the domain of programming.

References

1. https://en.wikipedia.org/wiki/Large_numbers
2. http://www.infoarena.ro/lucrul-cu-nr-mari
3. https://stackoverflow.com
4. https://www.pbinfo.ro/?pagina=articole&subpagina=afisare&id=5
5. https://www.sharelatex.com
6. http://www.stack.nl/~dimitri/doxygen/
7. https://github.com/Xerophyt
8. https://blogs.sap.com
9. https://softwareengineering.stackexchange.com
10. https://en.wikipedia.org/wiki/Arithmetic