Programming Report:
Long division

Gerard Renardel de Lavalette (p147604)

January 31, 2017

1 Problem description

General: Write a C program that prints the computation of the integer quotient of two natural numbers with the long division algorithm in the notation that is used in the Netherlands.

Input-output behaviour: The input of the program consists of two natural numbers smaller than 2^{64}, separated by a blank. The first number must be positive. The output is the result of the long division algorithm applied to the two numbers.
Example input:

```
345 12345678
```

Example output:

```
345 / 12345678 \ 35784
      1035
      ----
       1995
       1725
       ----
        2706
        2415
        ----
         2917
         2760
         ----
          1578
          1380
          ----
           198
```

2 Problem analysis

The computation of the quotient of number n and divisor d is easy, using the C function / (integer division). The essence of the problem is the computation of the numbers that appear in the long division, and the appropriate printing of these numbers.

We observe that a long division contains a number of *computation steps*. Each computation step is based on a start value and it yields an end value, which will be the start value for a next step. The initial start value v is the least initial segment of number n such that $d \leq v$, where d is the divisor. Every computation step consists of three parts, printed on three lines:

- a multiple m of d with the property $0 \leq v - m < d$;
- a line,
- the end value of the computation step, obtained by extending $v - m$ with digits from n until it is at least d.

The end value will be the new value of v. Computation steps are performed as long as $d \leq v$.

In pseudocode, the algorithm can be described as follows.

algorithm LongDivision(d, n)
  input natural numbers $d > 0$ and n
  output long division applied to d and n
  $v \leftarrow$ the least initial segment of n that is at least d
  while $d \leq v$ **do**
    $m \leftarrow d \cdot (v \text{ div } d)$ /* so $0 \leq v - m < d$ */
    print m
    print a line
    $v \leftarrow v \bmod d$ /* so $v < d$ */
    as long as $v < d$, extend v with digits from n
    print v

As an example, we display the first computation step in the example output given above:

```
1035
----
 1995
```

Recall that $d = 345$ and $n = 12345678$. The start value for this step is 1234, the first initial segment of 12345678 that is at least 345. Now 1035 equals 3×345, and $1234 - 1035$ equals 199 which is indeed non-negative and smaller than 345. 199 is extended with digit 5 so as to form the end value 1995, which will be the start value for the next step.

3 Program design

We define auxiliary functions `length` to determine the length of the decimal representation of a number, and `printLine` to print a line consisting of characters '−'.

The main function is `makeLongDivision`. It reads `divisor` and `number` and prints the first line of the long division. Then the array `digits` is filled with the digits of `number`, and `value` and `width` are initialized. The main `while` loop computes and prints computation steps, until `value` < `divisor` and no digits of n are available. The main loop contains a `while` loop to compute the new `value` and `width`.

The main program reads the input, checks for positivity of `divisor`, and performs long division.

Design choice. The `while` loop for the initial computation of `value` and `width` (line 57-61) is identical to the `while` loop for the update of these variables (line 71-75). This suggest the introduction of a function to make this identity explicit. However, we decided not to do this. The reason is that there are five parameters involved (three of them as reference parameter), which would lead to a rather complex function definition that would be quite top-heavy in the present context.

Time complexity. The time complexity of the program is in $\mathcal{O}(\log(\texttt{number}))$ (provided that `number` > `divisor`). To see this, we first observe that `length(number)` is in $\mathcal{O}(\log(\texttt{number}))$. In the function `makeLongDivision`, the variable `index` starts with value 0, is raised stepwise to the value `length(number)` and then decreases stepwise to 0. For every value of `index`, a bounded number of computation is performed. So the time complexity of `makeLongDivision` is in $\mathcal{O}(\log(\texttt{number}))$.

4 Evaluation of the program

We first test the program on the input `345 12345678` given in the example above. This yields

```
345 / 12345678 \ 35784
      1035
      ----
       1995
       1725
       ----
        2706
        2415
        ----
         2917
         2760
         ----
          1578
          1380
          ----
           198
```

We test the check on positivity of the divisor by feeding the input `0 12345`:

```
a.out: longDivision.c:92: main: Assertion 'divisor > 0' failed.
Aborted (core dumped)
```

Now we test what happens with the input `34567 1234`, where the number is smaller than the divisor:

```
34567 / 1234 \ 0
```

Finally a test with the input `12345 108018806787000`, which should produce a long division with computation steps that use more than one new digit from `number`, leading to zeroes in the quotient. Moreover, the last line in the last computation step should contain three zeroes.

```
12345 / 108018806787000 \ 8750004600
         98760
        ------
          92588
          86415
          -----
           61730
           61725
           -----
               56787
               49380
               -----
                74070
                74070
                -----
                    000
```

We observe that all tests are successful.

In order to check for memory leaks, we apply `valgrind`. With the input `345 1234567812345678` we then get the following information from `valgrind`:

```
==581== HEAP SUMMARY:
==581==     in use at exit: 0 bytes in 0 blocks
```

```
==581==   total heap usage: 1 allocs, 1 frees, 64 bytes allocated
==581==
==581== All heap blocks were freed -- no leaks are possible
==581==
==581== For counts of detected and suppressed errors, rerun with: -v
==581== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

So no memory leaks are found. We conclude that all tests are successful.

5 Process description

A long division consists of a first line, followed by a series of computation steps in which two numbers are subtracted. It worked well to use this overall structure of a long division as a blueprint for the main function `makeLongDivision`: see the `while` loop on line 64 to 78.

Designing the `while` loops for the initial computation and the subsequent recomputation of `value` and `width` was somewhat more involved. It lead to the slightly surprising guard `value < divisor && i >= 0`, and then to the insight that both loops are identical.

By developing this program I learned about the type `unsigned long` and also some useful properties and options of the function `printf`.

6 Conclusions

The program solves the problem of printing long division in a straightforward way, following the structure of a long division as a series of subtractions. It is also efficient: its time complexity is bounded by $\log(\texttt{number})$, which is also a tight bound for the number of computation steps in the long division.

7 Appendix: program text

Listing 1: `longDivision.c`

```
1  /* Gerard Renardel, 31 January 2017
2   * a C program for long division
3   */
4
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include <assert.h>
8
9  /* length(n) is the length of the decimal representation of the number n
10  */
11 int length (unsigned long n) {
12    int i = 0;
13    while ( n > 0 ) {
14       i = i + 1;
15       n = n/10;
16    }
17    return i;
18 }
19
20 /* printLine(l,w) prints a line consisting of l characters "-", preceded by an
       offset of w-l blanks
21  */
22 void printLine (int length, int width) {
23    int offset = width - length;
```

```
24    while ( offset > 0 ) {
25       printf("␣");
26       offset = offset - 1;
27    }
28    while ( length > 0 ) {
29       printf("-");
30       length = length - 1 ;
31    }
32    printf("\n");
33 }
34 
35 /* longDivision(d,n) prints the long division of n by d
36  */
37 void makeLongDivision (unsigned long divisor, unsigned long number) {
38    unsigned long value;
39    int width, oldWidth, index, lth;
40    int *digits;
41 
42    /* print the first output line: */
43    printf("%lu␣/␣%lu␣\␣%lu\n", divisor, number, number / divisor);
44 
45    /* allocate the required memory for digits, and fill it with the digits of
          number */
46    lth = length(number);
47    digits = malloc(lth * sizeof(int));
48    assert(digits != NULL);
49    index = 0;
50    while ( index < lth ) {
51       digits[index] = number % 10;
52       number = number / 10;
53       index = index + 1;
54    }
55 
56    /* initialize value as the least initial segment of number that is not
          smaller than divisor */
57    value = 0;
58    width = length(divisor) + 3;
59    while ( value < divisor && index > 0 ) {
60       index = index - 1;
61       value = value * 10 + digits[index];
62       width = width + 1;
63    }
64 
65    /* print computation steps until value < divisor */
66    while ( divisor <= value ) {
67       /* compute and print the appropriate multiple of divisor */
68       printf("%*lu\n", width, divisor * (value / divisor));
69       printLine(length(value), width);
70       /* update value and width */
71       value = value % divisor;
72       oldWidth = width;
73       while ( value < divisor && index > 0 ) {
74          index = index - 1;
75          value = value * 10 + digits[index];
76          width = width + 1;
77       }
78       /* print the new value, possibly preceded by zeroes */
79       printf("%*.*lu\n", width, width - oldWidth + 1, value);
```

```
80     }
81     free(digits);
82  }
83
84  /* the main program:
85   * scan the input,
86   * check for positivity of the divisor,
87   * perform long division.
88   */
89  int main(int argc, char *argv[]){
90     unsigned long divisor, number;
91     scanf("%lu␣%lu", &divisor, &number);
92     assert(divisor > 0);
93     makeLongDivision(divisor, number);
94     return 0;
95  }
```