Documentation Group Project 2 (Assignment07)

Mario Neuner, Leander Decristoforo

May 5, 2025

Contents

1 Problem01: Addition long numbers

1.1 Idea

In this problem we had to add a 30 numbers with each consisting of 40 digit numbers encoded in long chain of integers. Because there is no such common data type to store such big numbers we decided to use arrays instead and use them to implement the addition logic. So, the idea was to create two arrays, one for the result and another for the 1200 digit string and then use loops to calculate and store each digit in the result array.

1.2 Implementation

Firstly, we allocated space for the result integer array and initialized it with 0s. Afterwards we created a string for storing the long number containing the 40 digit numbers. We continued with creating the addition logic and started a double loop, the first one for the 40 digits and the second one for the 30 numbers. At the start of every iteration we calculated the correct index for the 1200 number and checked for negative indices, just to prevent errors. Then we set a new variable called digit to the digit of the number with the corresponding index and a position variable for the correct index in the result array. For a proper addition we first calulated the sum of the result element and the digit variable and only set the result element to the sum modulo 10, we assigned the rest (i.e. sum/10) to the next element in the array and continued the loop thereafter. The last few lines of code check for the first digit in the result array to be non 0 and then prints out the number from that start.

1.3 Output

`The program compiles with "gcc -Wall -Wextra -Werror -Wpedantic -std=c18 addition.c -o addition".` When running `./addition` you get the output: "The result (sum) is: 179160132458177213830275506295538542694832"

Listing 1: Addition

```
1    #include <stdlib.h>
2    #include <stdio.h>
3    // Authors: Mario Neuner, Leander Decristoforo
4
5    int main(void) {
6        // Allocate space for arrays
7        int *result = malloc(50 * sizeof(int));
8        if (result == NULL) {
```

```
9           fprintf(stderr, "ERROR: Not enough space for
                array allocation!\n");
10          return EXIT_FAILURE;
11      }
12
13      // Initialize result array to zeros
14      for (int i = 0; i < 50; i++) {
15          result[i] = 0;
16      }
17
18      char nums[1200] = "938918109104763030378 29...";
19
20      // Addition
21      for (int i = 0; i < 30; i++) {
22          for (int j = 0; j < 40; j++) {
23              // Calculate the correct index: Start
                    from end of string and move backwards
24              int index = 1199 - i*40 - j;
25              if (index >= 0) {
26                  int digit = nums[index] - '0';
27                  // Add to corresponding position in
                        result
28                  int pos = 49 - j;
29                  int sum = result[pos] + digit;
30                  result[pos] = sum % 10;
31
32                  // Add the rest to higher digits
33                  if (sum >= 10) {
34                      if (pos > 0) {
35                          result[pos-1] += sum / 10;
36                      }
37                  }
38              }
39          }
40      }
41
42      // Print sum
43      fprintf(stdout, "The result (sum) is:\n");
44
45      // Find first non-zero digit
46      int first_non_zero = 0;
47      while (first_non_zero < 49 && result[
            first_non_zero] == 0) {
```

```
48              first_non_zero++;
49          }
50
51          // Print from first non zero digit to least
52          for (int i = first_non_zero; i < 50; i++) {
53              fprintf(stdout, "%d", result[i]);
54          }
55          fprintf(stdout, "\n");
56
57          free(result);
58          return EXIT_SUCCESS;
59      }
```

2 Problem02: Pokerchips and Probability

2.1 Implementation and idea

For this problem we started with defining the required variables: The amount of total chips given, the amount of chips that are pulled out of the bag and the amount of chips with a value of ten euros. Following these definitions, we computed the expected value with a standard statistics formula: The probability to pull out a ten-euro-chip is multiplied by the amount of chips that are pulled out of the bag simultaneously. In conclusion, the result of this computation is printed via a standard print statement.

2.2 Output

`The program compiles with "gcc -Wall -Wextra -Werror -Wpedantic -std=c18 Pokerchips.c -o poker"`. When running `./poker` you get the output:
"The expected value for this draw is 1.800"

Listing 2: Pokerchips

```
1   #include <stdio.h>
2   // Authors: Mario Neuner, Leander Decristoforo
```

```
3
4       int main(void)
5       {
6           // definition of required variables
7           float chips_in_total = 50;
8           float chips_drawn = 9;
9           float zehn_euro_chips = 10;
10
11          // computation of expected value via standard
                formula
12          float erwartungswert = (chips_drawn *
                zehn_euro_chips / chips_in_total);
13
14          // print statement for result
15          printf("The␣expected␣value␣for␣this␣draw␣is␣%.3f
                \n", erwartungswert);
16
17          return 0;
18      }
```

3 Problem03: gapped square

3.1 Idea

The goal of this problem was to find another number which when squared follows a particular pattern, namely "1_2_3_4_5_4_3_2_1_0" where all "_" is an arbitrary digit. Therefore our idea was to use again a looping techinque and several conditionals for checking the each number until one is found that differs from the provided 1010101010.

3.2 Implementation

For this problem we started with implementing a function for computing the powers of 10 with a given integer (here just positive) for the sake of

convinience later in the code. Then the main functions starts with some variables for keeping track of the current number (num) as well as the result (result) of its square and boolean (found) for terminating the upcoming while-loop. We started with a sufficiently big number and terminated latest at the number with a bigger square than the desired pattern. In the iteration we started with incrementing the number by a factor of 10 since the last digit must be a 0 and set our boolean to true. Afterwards we checked for skipping the provided solution, telling the reader we did so. The last and crucial part of the loop first looks for the correct last 5 provided digits to be correct via a for loop, modulo 10 and the division by the desired power of ten with our neat function. Before the first 5 conditions are checked in a similar fashion, we sorted out the wrong numbers and continued. To be honest, the time efficiency of this alghorithm is certainly not the fastes but thus a bit easier to read and code.

3.3 Output

`The program compiles with "gcc -Wall -Wextra -Werror -Wpedantic -std=c18 gapped_square.c -o gs"`. When running `./gs` you get the output:
"Found 1010101010
The number is 1194710910 with the square 1427334158473028100"

Listing 3: Gapped square

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <stdbool.h>
4   // Authors: Mario Neuner, Leander Decristoforo
5
6   long long power_10(int power) {
7       long long result = 1;
8       for (int i = 0; i < power; i++) {
9           result *= 10;
10      }
11      return result;
12  }
13
14  int main (void) {
15      long long num = 1010100000, result = 0;
16      bool found = false;
```

```
17          // Loop through numbers
18          while (found == false && num < 1390000000) {
                    // ~sqrt(19300000000000000000)
19              // Increment variable
20              num += 10;
21
22              result = num * num;
23              found = true;
24
25              if (num == 1010101010) {
26                  fprintf(stdout,"Found␣1010101010\n");
27                  found = false;
28                  continue;
29              }
30
31              // Check second 5 conditions
32              for (int i = 4; i >= 0; i--) {
33                  if ((result / power_10(2 * i)) % 10 != i
                        ) {
34                      found = false;
35                      break;
36                  }
37              }
38
39              // If conditions so far false, continue
40              if (found == false) {
41                  continue;
42              }
43
44              // Check first 5 conditions
45              for (int i = 1; i <= 5; i ++) {
46                  if ((result / power_10(20 - 2*i)) % 10
                        != i) {
47                      found = false;
48                      break;
49                  }
50              }
51          }
52
53          fprintf(stdout, "The␣number␣is␣%lld␣with␣the␣
                square␣%lld\n", num, result);
54          return EXIT_SUCCESS;
55      }
```