Programming in C

Deadline: *5 February, 23:59*

Assignment 1: Serial numbers

A friend of yours owns a small business and his customers often pay him in cash. He is worried that the banknotes he receives might be counterfeit, so he asks if you can help detect the fakes. After some research you discover that in addition to the visual security features you can also check the serial numbers of the euro banknotes.

The first series of Euro banknotes was introduced in 2002. In 2012 the European Central Bank announced that these will be replaced with the second series, called the Europa series. Since 2013 the new 5 euro and 10 euro banknotes from the Europa series have entered circulation. Both the first and the second series have a unique serial number. The validity of the serial number can be checked in different ways. The format of the serial number on the first series of banknotes is a single letter followed by 11 digits. An example of a valid serial number from the first series of euro banknotes is `X55266826871`. The letter indicates the country that issued the banknote. This checksum_table [2] maps letters to the country that issued the banknote. In this case, the 'X' indicates that our banknote was issued in Germany.

First method

To check the validity of a serial number, we first split the letter from the numerical part of the serial number. This gives us the pair (X, 55266826871). The serial number is valid if the digital root of the numerical part matches the checksum digit associated with the country code 'X'. The digital root is calculated by repeatedly summing the individual digits of the number until the result becomes a single digit. The checksum_table [2] shows that the checksum digit of 'X' is 2. To calculate the digital root of the number 55266826871, we start by calculating the digit sum of 55266826871:

```
5+5+2+6+6+8+2+6+8+7+1 = 56
```

The digit sum of 56 is:

```
5+6 = 11
```

And finally we find the digital root:

```
1 + 1 = 2
```

The digital root of 55266826871 is 2 which matches the checksum digit in the reference table, so our serial number is indeed valid. We urge you to think about how you would implement this. Can you think of an efficient way to calculate a digital sum?

Second method

A second way to check the serial number is to convert the country code character to it's ASCII value. The sum of digits of the number part plus the ASCII value of the country code should then be divisible by 9. The digit sum of 55266826871 is (still):

```
5+5+2+6+6+8+2+6+8+7+1 = 56
```

The ASCII value of 'X' is 88, so:

```
56 + 88 = 144
144 % 9 = 0
```

Therefore, our serial number is also correct according to the second method.

Assignment

Your assignment is to implement these two methods to check the serial numbers of banknotes of the first series.

Your program should read the serial numbers from standard input, one serial number per line. To check your program you can use the banknotes in your wallet or use the `serial.txt` file in the `template_code` directory. The first five serial numbers in `serials.txt` are valid, and the last two are invalid. Your program should print the serial number followed by `OK` if the serial number is correct, or `FAILED` if the serial number fails one or both of the tests. With `serials.txt` as input, the output of your program should be:

```
$ ./checksum < serials.txt
X55266826871: OK
U77882681066: OK
Z74542963239: OK
L40424180045: OK
L25388047085: OK
A25388047085: FAILED
L25388049086: FAILED
```

Implementation tips

C is low level language and reading data is more complicated than in higher level languages such as Python or Java. To help you along with your first assignment we have provided some template code that reads the serial numbers from standard input. The main loop is:

```c
int main(void) {
    long long num; // Need 64 bit integer to store 11 digit serial number.
    char c;

    while (fscanf(stdin, "%c%lld\n", &c, &num) == 2) {
        printf("%c%lld: ", c, num);

        if (check_serial(c, num)) {
            printf("OK");
        } else {
            printf("FAILED");
        }
        printf("\n");
    }
    return EXIT_SUCCESS;
}
```

You can find the rest of the code in `checksum.c`. The first thing to notice is that the number part of the serial number is an 11 digit number that will not fit in the C `int` or `long` type. So the code uses the``long long`` integer type which is guaranteed to be 64 bits (C data types [3]). This might seem confusing at first, but remember that C can be compiled for a myriad of architectures. While your computer might have 32 bit integers, the `int` type only guarantees 16 bits in the C standard. The `long` type, which you might

expect to be 64 bits in size, is only guaranteed to be 32 bits! You might want to think about the following problem: if you want to store a decimal number with eleven digits in binary, how many bits do you need?

The serial numbers of the bank notes have a fixed format, and if you know the format of the input data you should use a function of the `scanf()` family [1]. Since you are reading from a file (remember, standard input in C is nothing more than a glorified file pointer) you use `fscanf()`, which has three arguments: the file descriptor to read from, a format string that specifies the types of the data elements to read, and the locations to store the data that is read. Because the scan function needs to store the data we need to pass the locations of the variables instead of the variables themselves. This is done with the `&` operator and will be explained later on in the lectures. For now you can just use the code and expect the data that is read to appear in the variables `c` and `num`. The return value of `fscanf` is the number of elements read, so the loop ends when there are no more serial number to read. The template code in `checksum.c` lacks the actual checksum tests and will always accept the serial numbers that it reads. You can type `make check` to run the program with `serials.txt` as input.

You will also need a function that takes a country code character and returns the checksum digit for that country. For this can use a `switch` statement or a lookup in an array that contains the data from the checksum_table [2].

You are of course free to ignore this code completely and write your own program from scratch. You could for example consider using `getchar`. Either way we encourage you to read up on the details of the library functions you use in the manual pages. For more information about the manual system type: `man man`

Always check your input. Your program should identify invalid input and report this to the user, it should not crash or display undefined behaviour. So think about what you consider invalid input and implement the correct behaviour for it.

Bonus assignments

Europa Series

Extend your implementation to handle serial numbers from both the first series and the Europa series of banknotes. The serial numbers of the Europa series have a different format: Two letters followed by 10 digits. The first letter is still the country code, and the second letter can be seen as an extended digit. When you calculate the digit sum of the serial number you can just use the ASCII value of the second letter. The program should check if the format of the serial number is valid, identify the serie the serial belongs to and then perform the checks to see if the serial itself is valid. The file `tests_bonus_assignment.txt`:

```
NA4532489209
TA2317091431
NA4532489210
X55266826871
4532489209
NAA4532489209
1234 is not a serial number
U77882681066

EA4116628099
```

Should give the following output:

```
NA4532489209: [europa series] OK
TA2317091431: [europa series] OK
NA4532489210: [europa series] FAILED
X55266826871: [first series] OK
4532489209: INVALID FORMAT
```

```
NAA4532489209: INVALID FORMAT
1234 is not a serial number: INVALID FORMAT
U77882681066: [first series] OK
EA4116628099: [europa series] OK
```

Generating serial numbers

Checking serial numbers is useful, but it is of course more fun to find an efficient way to generate valid serial numbers. As a bonus exercise write a program that will efficiently generate valid serial numbers for a given country code.

1	the **f** in scan**f**() stands for format.
2(1, 2, 3)	http://en.wikipedia.org/wiki/Euro_banknotes#Serial_number
3	http://en.wikipedia.org/wiki/C_data_types