Lab 3

COMP9021, Session 1, 2014

The aim of this lab is to:

- practice the use of arithmetical operators, tests and loops;
- develop problem solving skills by designing solutions to problems similar to others already seen, and to significantly different ones;
- come up with a different design to a given solution.

1 ☞ Finding particular sequences of prime numbers

Write a program that finds all sequences of consecutive prime 5-digit numbers, say of the form (a, b, c, d, e, f), such that $b = a + 2$, $c = b + 4$, $d = c + 6$, $e = d + 8$, and $f = e + 10$. So a, b, c, d and e are all 5-digit prime numbers and no number between a and b, between b and c, between c and d, between d and e, and between e and f is prime.

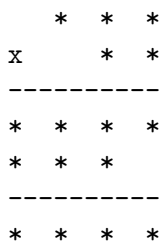
The expected output is:

```
The solutions are:

    13901   13903   13907   13913   13921   13931
    21557   21559   21563   21569   21577   21587
    28277   28279   28283   28289   28297   28307
    55661   55663   55667   55673   55681   55691
    68897   68899   68903   68909   68917   68927
```

2 Decoding a multiplication

Write a program that decodes all multiplications of the form



such that the sum of all digits in all 4 columns is constant.

The expected output is:

```
411 * 13 = 5343, all columns adding up to 10.
425 * 23 = 9775, all columns adding up to 18.
```

3 Decoding a sequence of operations

Write a program that finds all possible ways of inserting `+` and `-` signs in the sequence `123456789` (at most one sign before any digit) such that the resulting arithmetic expression evaluates to `100`.

Here are a few hints.

- `1` can either be preceded by `-`, or optionally be preceded by `+`; so `1` starts a negative or a positive number.
- All other digits can be preceded by `-` and start a new number to be subtracted to the running sum, or be preceded by `+` and start a new number to be added to the running sum, or not be preceded by any sign and be part of a number which it is not the leftmost digit of. That gives 3^8 possibilities for all digits from `2` to `9`. We can generate a number N in the range $\{0, 3^8 - 1\}$, using the function `pow()` from the standard maths library, called as `pow(3, 8)`. This requires the preamble of the program to contain:

  ```
  #include <math.h>
  ```

  Then we can:

  - consider the remainder division of N by 3 to decide which of the three possibilities applies to `2`;
  - consider the remainder division of $\frac{N}{3}$ by 3 to decide which of the three possibilities applies to `3`;
  - consider the remainder division of $\frac{N}{3^2}$ by 3 to decide which of the three possibilities applies to `4`;
  - ...

The expected output is (the ordering could be different):

```
 1 + 23 - 4 + 5 + 6 + 78 - 9 = 100
 123 - 4 - 5 - 6 - 7 + 8 - 9 = 100
 123 + 45 - 67 + 8 - 9 = 100
 123 + 4 - 5 + 67 - 89 = 100
 12 + 3 + 4 + 5 - 6 - 7 + 89 = 100
 123 - 45 - 67 + 89 = 100
 12 - 3 - 4 + 5 - 6 + 7 + 89 = 100
 1 + 2 + 34 - 5 + 67 - 8 + 9 = 100
 1 + 2 + 3 - 4 + 5 + 6 + 78 + 9 = 100
-1 + 2 - 3 + 4 + 5 + 6 + 78 + 9 = 100
 12 + 3 - 4 + 5 + 67 + 8 + 9 = 100
 1 + 23 - 4 + 56 + 7 + 8 + 9 = 100
```

4 Alternating design

Recall the program puzzle_2.c from the third set of notes. Make a copy of it, under the name puzzle_2_variant.c.

Modify puzzle_2_variant.c so that function test() becomes of type int (returning an int) rather than being of type bool (returning true or false), and takes as second argument an int rather than the address of an int. The comment before test() in the listing that follows indicates how this new version of test() is expected to behave. The listing also shows test()'s prototype suitably modified. (First copy and paste from the listing below into puzzle_2_variant.c to appropriately change the prototype of test() and the comment that precedes its definition.)

```
...

int test(int, const int);

...

/* Extracts each digit dig that occurs in i, from right to left,
 * and examines whether dig is null or the dig-th bit of digits is set to 1.
 * If that is the case, returns digits unchanged to indicate
 * that an occurrence of 0 has been found in candidate solution member,
 * or a second occurrence of dig has been found in candidate solution member.
 * Otherwise, sets dig-th bit of digits to 1 for each digit dig that occurs in i
 * and returns the resulting value. */
int test(int i, int digits) {
    ....
}
```