Assignment 1

COMP9021, Session 1, 2014

Aims: The purpose of the assignment is to

- let you design solutions to simple problems;
- let you implement these solutions in the form of short C programs;
- let you study existing code and take advantage of it to design solutions to your problems and write your own code;
- practice the use of arithmetic computations, tests, repetitions and arrays.

Submission

Your programs will be stored in a number of files, with one file per exercise, of the appropriate name. When you have developed and tested your programs, look at the provided checklist and verify that you tick all boxes (or at least be aware that you should tick them all...). Then upload your files using WebCMS. Assignments can be submitted more than once: the last version is marked. Your assignment is due by April 6, 11:59pm.

Assessment

For the first two exercises, up to 2 marks and for the last exercise, up to 3 marks will reward the correctness of the solution (provided it has not been hard coded...).

For each exercise, up to 1 mark will reward good formatting of the source code and reasonable complexity of the underlying logic as measured by the level of indentation of statements. For that purpose, the `mycstyle` script will be used, together with `cryptarithm_style_sheet.txt`, `multiplication_style_sheet.txt` and `division_style_sheet.txt` where *Maximum level of indentation* is set to `4` in the first two and to `6` in the last one. For each exercise,

- if the script identifies problems different to excessive indentation levels then you will score 0 out of 1 for the style;
- otherwise, if the script identifies excessive indentation levels, then you will score 0.5 out of 1 for the style;
- otherwise, if the script identifies no problem then you will score 1 out of 1 for the style.

In these style sheets, you can change any of the variable constraints except for the maximal level of indentation (you will be penalised if you do). You will have to submit these style sheets.

For each exercise, if your program attempts too little and contains no substantial code, then the `mycstyle` script won't be used and you will score 0.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

Exercise 1: `cryptarithm.c` (3 marks).

A cryptarithm is a puzzle where digits have to be assigned to the letters of some words, different letters being assigned different digits, and no word starting with `0`, such that a number of conditions are satisfied, that are naturally related to the meanings of the words when the words do have a meaning. The program `sample_cryptarithm.c` that comes with this assignment provides you with an example of a cryptarithm, where the task is to assign digits to the letters in three, four and eight, in such a way that:

- three is prime;
- four is a perfect square;
- eight is a perfect cube

It has a unique solution, namely, `42611` for three, `7056` for four, and `13824` for eight (so t is assigned the digit `4`, h is assigned the digit `2`, etc.).

Write a program that assigns a digit to each letter that occurs in the words `cross`, `roads` and `danger`, distinct letters being assigned distinct digits, and the digit assigned to the first letter of each of the three words being nonzero, such that the following equation holds.

$$\texttt{cross} + \texttt{roads} = \texttt{danger}$$

The output of your program, saved as `cryptarithm.c`, should be a line of the form

```
cross = ....., roads = ..... and danger = ...... is a solution.
```

Your program should not make any assumption on the actual solution, except obvious ones on the ranges of some values.

The automarking script will let your program run for 30 seconds. Still you should not take advantage of this and strive for a solution that gives an immediate output.

For this exercise, the maximum number of indentations is set to **4**.

Use `cryptarithm_style_sheet.txt` and the `mycstyle` script to check that your program complies with this requirement.

Exercise 2: `multiplication.c` (3 marks).

A related kind of puzzle consists in assigning digits to stars so that some operation such as a multiplication or a division whose outline is given is satisfied, possibly together with other constraints. The program `sample_multiplication.c` that comes with this assignment provides you with an example of this kind of puzzle, where the aim is to solve the multiplication

```
            * * *
          x * * *
            -------
          * * * *
        * * * *
      * * * *
      -------------
      * * * * * *
```

in such a way that:

- each star stands for a digit, with the leftmost star on each line standing for a nonzero digit;
- all partial products are made up of the same digits;
- the first and second partial products are different;
- the orderings of digits in the second and third partial products are inverse of each other;
- the third partial product is the sum of the first two partial products.

Write a program that solves a multiplication

```
     * * *
   x   * *
     -----
   * * * *
 * * * *
 ---------
 * * * * *
```

in such a way that every star is replaced by a prime digit.

The output of your program, saved as `multiplication.c`, should be of the form above, with of course the stars being replaced by appropriate digits.

Your program should not make any assumption on the actual solution, except obvious ones on the ranges of some values.

For this exercise, the maximum number of indentations is set to **4**.

Use `multiplication_style_sheet.txt` and the `mycstyle` script to check that your program complies with this requirement.

Exercise 3: `division.c` (4 marks).

Write a program that solves the division

```
* * * * * * * * | * * *
                |----------
* * * *         | * 7 * * *
-------         |
    * * *       |
    * * *       |
    -----       |
    * * * *     |
      * * *     |
      -----     |
        * * * * |
        * * * * |
        ------- |
              0 |
```

Of course, no number starts with `0`. Moreover, some stars might be replaced by `7`.

The output of your program, saved as `division.c`, should be of the form above, with of course stars being replaced by appropriate digits, with no blank line, and with no space to the left of any digit or `|` on the first line of output.

You can assume that there is a single solution, hence let the program exit after a solution has been found.

Your program should not make any assumption on the actual solution, except obvious ones on the ranges of some values.

Again, the automarking script will let your program run for 30 seconds, but you should not take advantage of this and strive for a solution that gives an immediate output.

For this exercise, the maximum number of indentations is set to **6**.

Use `division_style_sheet.txt` and the `mycstyle` script to check that your program complies with this requirement.