Assignment 1

COMP9021, Session 1, 2013

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 14, 11:59pm.

Assessment

For each of the first three exercises, up to 2.5 marks will reward the correctness of the solution, and for the fourth exercise, up to 0.5 mark will reward the correctness and the efficiency of the solution (provided these solutions have not been hard coded...). For all exercises, up to 0.5 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 `style_sheet.txt` where **Maximum level of indentation** has to be set to `5` for the first exercise, to `4` for the second one, `4` for the third one, and `6` for the fourth one. More precisely, for each exercise,

- if the script identifies problems different to excessive indentation levels then you will score 0 out of 0.5 for the style;
- otherwise, if the script identifies excessive indentation levels, then you will score 0.25 out of 0.5 for the style;
- otherwise, the script identifies no problem and you will score 0.5 out of 0.5 for the style.

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

For the first three exercises, the automarking script will let each of your programs run for 30 seconds. Still you should not take advantage of this and strive for a solution that gives an immediate output. For the fourth exercise, the automarking script will let your program run for 1 second only, and it will check whether the solution has been output in less than 0.1 second.

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 one, two, seven and nine, distinct letters being assigned distinct digits, and the digit assigned to the first letter of each of the three words being nonzero, such that:

- one + one = two;
- seven is prime;
- nine is a perfect square.

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

```
one = ..., two = ..., seven = ..... and nine = .... is a solution.
```

for each found solution, with of course every occurrence of `.` replaced by an appropriate digit.

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

For this exercise, the maximum number of indentations is set to **5** (use the `style_sheet.txt` file and the `mycstyle` script to check that your program complies with this requirement).

If you use the function `sqrt()` from the standard library to compute the square root of a number, do not forget to insert the line:

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

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 the multiplication

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

where:

- all digits of the multiplicand are distinct and strictly positive, and
- all digits in the multiplier, partial products and final product occur in the multiplicand.

The output of your program, saved as `multiplication.c`, should be of the form

```
   ....
  x ...
   ----
  .....
 .....
 ....
-------
.......
```

for each found solution, with of course every occurrence of `.` being replaced by an appropriate digit.

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

For this exercise, the maximum number of indentations is set to **4** (use the `style_sheet.txt` file and the `mycstyle` script to check that your program complies with this requirement).

Exercise 3: `quotient.c` (3 marks)

Write a program that finds all fractions of the form $\frac{n}{d}$ such that n and d are positive integers with no occurrence of 0, $\frac{n}{d}$ evaluates to 0.5, and every nonzero digit occurs once and only once in n or d.

To print out your solutions properly, your code should include a statement of the following form:

```
printf("%d / %d = 0.5\n", ..., ...);
```

for each found solution and no other `printf()` statement.

Your program should be saved as `quotient.c`.

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

For this exercise, the maximum number of indentations is set to **4** (use the `style_sheet.txt` file and the `mycstyle` script to check that your program complies with this requirement).

Exercise 4: square.c (more difficult, 1 mark)

Write a program that finds all possible ways of replacing stars with digits in

```
          *
        * * *
      * * * * *
    * * * * * * *
* * * * * * * * * *
```

in such a way that:

- all resulting numbers, whether read horizontally or vertically, are perfect squares;
- numbers consisting of 2 digits or more do not start with 0;
- 0 can be one of the 1-digit numbers.

To print out your solutions properly, your code should include a sequence of statements of the following form:

```
printf("%5d\n", ...);
printf("%6d\n", ...);
printf("%7d\n", ...);
printf("%d\n\n", ...);
```

and no other printf() statement.

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

Your program should be saved as square.c.

For this exercise, the maximum number of indentations is set to **6** (use the style_sheet.txt file and the mycstyle script to check that your program complies with this requirement).

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

For this exercise, you will score 0.25 mark by automarking if your program outputs the solution(s) in less than 1 second on my machine, and 0.5 mark if it outputs the solution(s) in less than 0.1 second.

If you use the function sqrt() or the function fmod() from the standard library to compute the square root of a number or to compute the remainder of a floating point division, respectively, do not forget to insert the line:

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