CP Lab 4½: Some problems

This optional labsheet is for those who are already caught up during the catch-up labs.

If you're doing this, you've already confident with everything so far, and so we won't guide you through problems in the style of the normal labs, but simply state some tasks and invite you to do as many as you wish, in any order.

There is no submission or automatic testing – it's purely for fun and practice.

Repetition

Write a program which requires the user to input five numbers between 1 and 50 (inclusive), *with no repetition*. If the user repeats a number, they should be required to enter another number until five distinct numbers have been obtained. You will probably want to use arrays, though it can be done without.

This might take around ten to fifteen minutes.

Deviation

Take one of your 'average of many' programs from lab 3, and extend it to print also the standard deviation of the numbers entered. The standard deviation of numbers $x_1, \ldots, x_n$ is $\sqrt{\sum_{i=1}^{n}(x_i-\mu)^2/n}$ where μ is the mean of the numbers. You will need to use the maths library – remember to `#include <math.h>` and compile with `-lm`.

This should take about five minutes.

Hesitation

Write a program which repeatedly presents 2-digit multiplication problems to the user (e.g. `What is 37*45?`), reads their answer, says if it's correct, and if so, how long it has taken them to do the sum.

You will need:

- to have `#include <sys/time.h>` in your program
- to write the following function in your program, which gives the current time as a floating point number of seconds (since 00:00 on 1 January 1970, as it happens):

```
double mytime(void) {
  struct timeval tv;
  gettimeofday(&tv,NULL);
  return tv.tv_sec + tv.tv_usec/1E6;
}
```

- A pseudo-random number between 0 and 99 can be obtained from the expression `rand()%100` Note that the same random numbers will be obtained each time you run the program. If you want to avoid this, then `#include <time.h>`, and in your initialization code call `srand((int)time(NULL))`

This might take fifteen to twenty minutes.

Tailoring

Take your `rectangle.c` program from lab 4, and extend it so that when it detects a *tall* or *wide* rectangle, it draws a red line to indicate the tallest or widest *almost square* rectangle that fits inside. That is, given a *tall* rectangle of width w and height h, draw a horizontal red line at height $1.25w$ from the bottom, and similarly for wide.

You will need to study the `descartes.h` file to see how to change colours in `descartes`.

This might take about fifteen minutes.

Doomsday

One of the first programs we asked you to write was `whatday.c`. This relied on knowing what weekday 1 January was, and then working out the weekday for the rest of the year. You could, naturally, also work backwards and forwards to deal with earlier and later years. But that would be boring.

The great mathematician John Conway, as well as making many profound contributions to several areas of mathematics, has always had a lot of fun with recreational mathematics. He has devised an algorithm for calculating the day of the week, which is specifically designed to be learned and then used very fast in your head to impress people. He calls it the 'doomsday algorithm'.

Use the Web to find out what the algorithm is, and write a program to implement it, so that your program takes a day, month and year, and prints the weekday.

This might take half an hour or so.

Life

Conway is also famous for the 'Game of Life'. Look it up, and implement it for a 100×100 grid of cells, using `descartes`, so that each cell is a 5×5 pixel square. Let the user click to set the initial live cells, and middle-click (which, recall, returns the point $(-1,-1)$) to set the game running. You will need to use the following:

- the `rectangle_t` type of `descartes`, and the functions `Rectangle()`, `FillRectangle()` and `ClearRectangle()` – see the `descartes.h` file for the details of these functions.
- 2-dimensional arrays – see slide 17 of lecture 9. You will probably want to have arrays of rectangles, and of integers.

It may be useful to remember that you can interrupt a running program with ctrl-C in the terminal window.

Note: Descartes is a very simple library optimized for teaching, not performance. Owing to the way it uses the underlying graphics library, your program will be *very slow* if you update every cell on every pass – write your program so that you only fill/clear cells that have changed colour.

This will probably take at least an hour (though a skilled programmer could do it in 15 minutes:–).

Julian Bradfield, October 2016