1. Week 2: Before Arrays

1.1 Cash Machine (ATM)

Some cash dispensers only contain £20 notes. When a user types in how much money they'd like to be given, you need to check that the amount requested can be dispensed exactly using only £20 notes. If not, a choice of the two closest (one lower, one higher) amounts is presented.

Exercise 1.1 Write a program that inputs a number from the user and then prompts them for a better choice if it is not correct. For example :

```
How much money would you like ? 175
I can give you 160 or 180, try again.
How much money would you like ? 180
OK, dispensing ...
```

or :

```
How much money would you like ? 25
I can give you 20 or 40, try again.
How much money would you like ? 45
I can give you 40 or 60, try again.
How much money would you like ? 80
OK, dispensing ...
```

In this assessment you may assume the input from the user is "sensible" i.e. is not a negative number etc. ■

1.2 Secret Codes

Write a program that converts a stream of text typed by the user into a 'secret' code. This is achieved by turning every letter 'a' into a 'z', every letter 'b' into a 'y', every letter 'c' into and 'x' and so on.

Exercise 1.2 Write a function whose 'top-line' is :

```
int scode(int a)
```

that takes a character, and returns the secret code for this character. Note that the function **does** need to preserve the case of the letter, and that non-letters are returned unaffected.

When the program is run, the following input:

```
The Quick Brown Fox Jumps Over the Lazy Dog !
```

produces the following output :

```
Gsv Jfrxp Yildm Ulc Qfnkh Levi gsv Ozab Wlt !
```

■

1.3 Leibniz Π

See `http://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80`

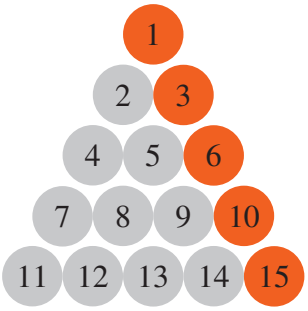
The Mathematical constant π can be approximated using the formula :

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \ldots$$

Notice the pattern here of alternating $+$ and $-$ signs, and the odd divisors. Write a program that computes π looping through smaller and smaller fractions of the series above. How many iterations does it take to get π correctly approximated to 11 digits ?

1.4 Triangle Numbers

A Triangle number is the sum of numbers from 1 to n. The 5^{th} Triangle number is the sum of numbers $1,2,3,4,5$, that is 15. They also relate to the number of circles you could stack up as equilateral triangles



$T_1 = 1$

$T_2 = 3$

$T_3 = 6$

$T_4 = 10$

$T_5 = 15$

Exercise 1.3 Write a program that prints out the sequence of Triangle numbers, using iteration, computing the next number based upon the previous. Check these against `http://oeis.org/A000217`.
■

Exercise 1.4 Write a program that prints out the n^{th} Triangle number based on the Equation :

$$T_n = n * (n+1)/2$$

■

1.5 Hailstone Sequence

Hailstones sequences are ones that seem to always return to 1. The number is halved if even, and if odd then the next becomes `3*n+1`. For instance, when we start with the number 6, we get the sequence : `6, 3, 10, 5, 16, 8, 4, 2, 1` that has nine numbers in it. When we start with the number 11, the sequence is longer, containing 15 numbers : `11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1`.

Exercise 1.5 Write a program that :

- displays which initial number (less than $50,000$) creates the **longest** hailstone sequence.
- displays which initial number (less than $50,000$) leads to the **largest** number appearing in the sequence.

■

2. Week 2: Simple Arrays

2.1 Neill's Microwave

Last week I purchased a new, state-of-the-art microwave oven. To select how long you wish to cook food for, there are three buttons: one marked "10 minutes", one marked "1 minute" and one marked "10 seconds". To cook something for 90 seconds requires you to press the "1 minute" button, and the "10 seconds" button three times. This is four button presses in total. To cook something for 25 seconds requires three button presses; the "10 second" button needs to be pressed three times and we have to accept a minor overcooking of the food.

Exercise 2.1 Using an array to store the cooking times for the buttons, write a program that, given a required cooking time in seconds, allows the minimum number of button presses to be determined.

Example executions of the program will look like :

```
Type the time required
25
Number of button presses = 3
Type the time required
705
Number of button presses = 7
```

■

2.2 Music Playlisters

Most MP3 players have a "random" or "shuffle" feature. The problem with these is that they can sometimes be **too** random; a particular song could be played twice in succession if the new song to play is truly chosen randomly each time without taking into account what has already been played.

To solve this, many of them randomly order the entire playlist so that each song appears in a random place, but once only. The output might look something this:

```
How many songs required ? 5
4 3 5 1 2
```

or :

```
How many songs required ? 10
1 9 10 2 4 7 3 6 5 8
```

Exercise 2.2 Write a program that gets a number from the user (to represent the number of songs required) and outputs a randomised list. ■

2.3 Rule 110

Rather interesting patterns can be created using *Cellular Automata*. Here we will use a simple example, one known as *Rule 110* : The idea is that in a 1D array, cells can be either on ■ or off □ (perhaps represented by the integer values 1 and 0). A new 1D array is created in which we decide upon the state of each cell in the array based on the cell above and its two immediate neighbours.

If the three cells above are all 'on', then the cell is set to 'off' ($111 \rightarrow 0$). If the three cells above are 'on', 'on', 'off' then the new cell is set to 'on' ($110 \rightarrow 1$). The rules, in full, are:
$111 \rightarrow 0$
$110 \rightarrow 1$
$101 \rightarrow 1$
$100 \rightarrow 0$
$011 \rightarrow 1$
$010 \rightarrow 1$
$001 \rightarrow 1$
$000 \rightarrow 0$

You take a 1D array, filled with zeroes or ones, and based on these, you create a new 1D array of zeroes and ones. Any particular cell uses the three cells 'above' it to make the decision about its value. If the first line has all zeroes and a single one in the middle, then the automata evolves as:

Exercise 2.3 Write a program that outputs something similar to the above using plain text, giving the user the option to start with a randomised first line, or a line with a single 'on' in the central location. ■

Exercise 2.4 Rewrite the program above to allow other rules to be displayed - for instance 124, 30 and 90.

http://en.wikipedia.org/wiki/Rule_110

■

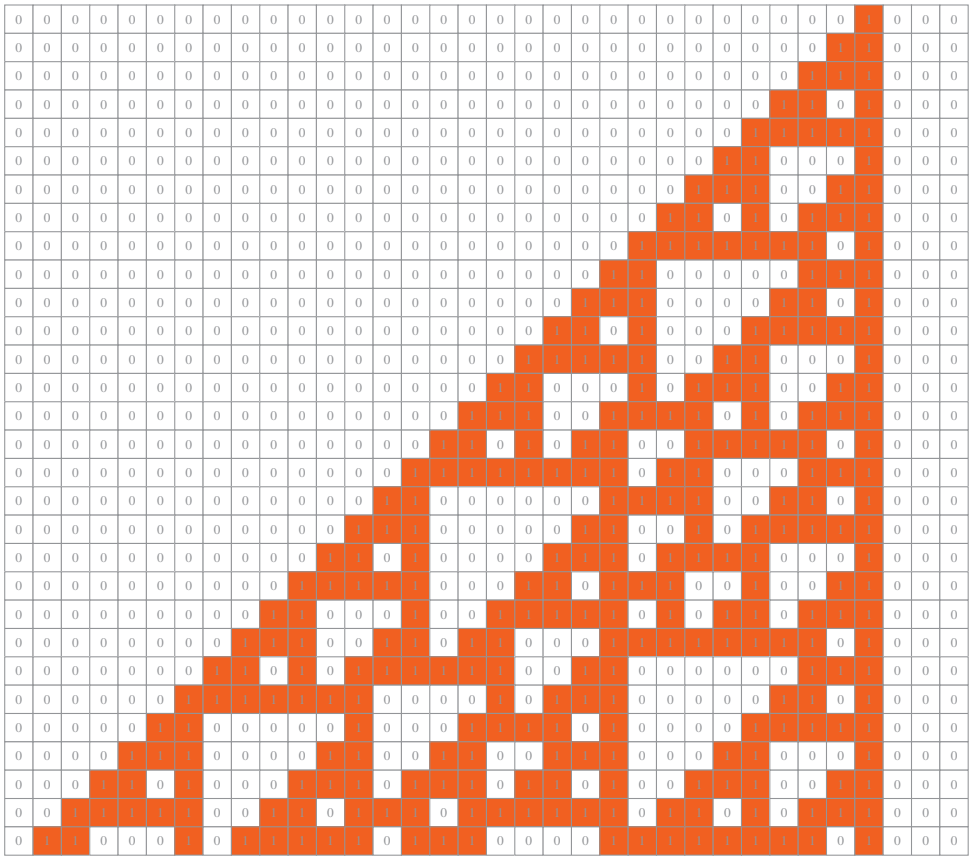


Figure 2.1: 1D cellular automaton using Rule 110. Top line shows initial state, each subsequent line is produced from the line above it. Each cell has a rule to switch it ‘on’ or ’off’ based on the state of the three cells above it in the diagram.