Neill's Microwave
Music Playlisters
Rule 110
Palindromes
Int to String
Roman Numerals
Soundex Coding

3. 1D Arrays and Strings

For some of these exercises you'll need to understand command line input to `main()` from the shell, often simply referred to as `argc/argv` in C. Please see:

`http://www.thegeekstuff.com/2013/01/c-argc-argv/`

for more information about this.

3.1 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 3.1 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
```

3.2 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 3.2 Write a program that gets a number from the user (to represent the number of songs required) and outputs a randomised list. ■

Exercise 3.3 Rewrite Exercise 3.2 so that the program passes an array of integers (e.g. `[1,2,3,4,5,6,7,8,9,10]`) to a function and re-orders them **in-place** (no other arrays are used) and with an algorithm having complexity $O(n)$. ■

3.3 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 3.4 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 3.5 Rewrite the program above to allow other rules to be displayed - for instance $124, 30$ and 90.

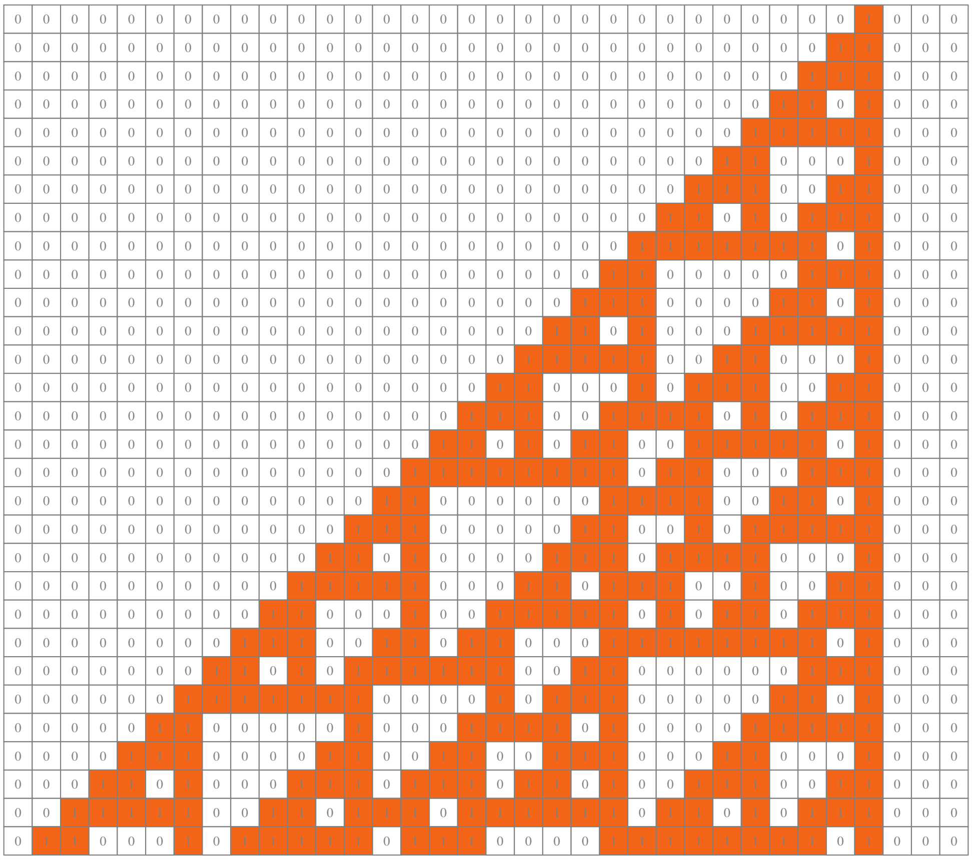

Figure 3.1: 1D cellular automaton using Rule 110. Top line shows initial state, each subsequent line is produced from the line above it. Each cell has a rule to switch it ‘on’ or ’off’ based on the state of the three cells above it in the diagram.

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

3.4 Palindromes

From `wikipedia.org` :

> A palindrome is a word, phrase, number or other sequence of units that has the property of reading the same in either direction (the adjustment of punctuation and spaces between words is generally permitted).
>
> The most familiar palindromes, in English at least, are character-by-character: the written characters read the same backwards as forwards. Palindromes may consist of a single word (such as "civic" or "level"), a phrase or sentence ("Neil, a trap! Sid is part alien!", "Was it a rat I saw?") or a longer passage of text ("Sit on a potato pan, Otis."), even a fragmented sentence ("A man, a plan, a canal: Panama!", "No Roman a moron"). Spaces, punctuation and case are usually ignored, even in terms of abbreviation ("Mr. Owl ate my metal worm").

Exercise 3.6 Write a program that prompts a user for a phrase and tells them whether it is a palindrome or not. **Do not** use any of the built-in string-handling functions (`string.h`), such as `strlen()` and `strcmp()`. However, you **may** use the character functions (`ctype.h`), such as `islower()` and `isalpha()`.

Check you program with the following palindromes :

```
"kayak"
"A man, a plan, a canal: Panama!"
"Madam, in Eden I'm Adam,"
"Level, madam, level!"
```

■

3.5 Int to String

Exercise 3.7 Write a function that converts an integer to a string, so that the following code snippet works correctly:

```
int i;
char s[256];
scanf("%d", &i);
int2string(i,s);
printf("%s\n", s);
```

The integer may be signed (i.e. be positive or negative) and you may assume it is in base-10.

Avoid using any of the built-in string-handling functions to do this (e.g. `itoa()`!) including those in `string.h`. ■

3.6 Roman Numerals

Adapted from:

www `http://mathworld.wolfram.com/RomanNumerals.html`

> "Roman numerals are a system of numerical notations used by the Romans. They are an additive (and subtractive) system in which letters are used to denote certain "base" numbers, and arbitrary numbers are then denoted using combinations of symbols. Unfortunately, little is known about the origin of the Roman numeral system.
>
> The following table gives the Latin letters used in Roman numerals and the corresponding numerical values they represent :

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

> For example, the number 1732 would be denoted MDCCXXXII in Roman numerals. However, Roman numerals are not a purely additive number system. In particular,

instead of using four symbols to represent a 4, 40, 9, 90, etc. (i.e., IIII, XXXX, VIIII, LXXXX, etc.), such numbers are instead denoted by preceding the symbol for 5, 50, 10, 100, etc., with a symbol indicating subtraction. For example, 4 is denoted IV, 9 as IX, 40 as XL, etc."

It turns out that every number between 1 and 3999 can be represented as a Roman numeral made up of the following one- and two-letter combinations:

I	1	IV	4
V	5	IX	9
X	10	XL	40
L	50	XC	90
C	100	CD	400
D	500	CM	900
M	1000		

Exercise 3.8 Write a program that reads a roman numeral (in the range `1 - 3999`) and outputs the corresponding valid arabic integer. Amongst others, check that `MCMXCIX` returns `1999`, `MCMLXVII` returns `1967` and that `MCDXCI` returns `1491`.

You should use the following template :

```
#include <stdio.h>

int romanToArabic( char *roman );

int main(int argc, char **argv)
{
    if( argc==2 ){
        printf("The roman numeral %s is equal to %d\n", \
        argv[1], romanToArabic(argv[1]));
    }else{
        printf("ERROR: Incorrect usage, try e.g. %s XXI\n", argv[0]);
    }
    return 0;
}
```

You need to add the function `romanToArabic()`. ■

3.7 Soundex Coding

First applied to the 1880 census, Soundex is a phonetic index, not a strictly alphabetical one. Its key feature is that it codes surnames (last names) based on the way a name sounds rather than on how it is spelled. For example, surnames that sound the same but are spelled differently, like Smith and Smyth, have the same code and are indexed together. The intent was to help researchers find a surname quickly even though it may have received different spellings. If a name like Cook, though, is spelled Koch or Faust is Phaust, a search for a different set of Soundex codes and cards based on the variation of the surname's first letter is necessary.

To use Soundex, researchers must first code the surname of the person or family in which they are interested. Every Soundex code consists of a letter and three numbers, such as B536, representing names such as Bender. The letter is always the first letter of the surname, whether it is a vowel or a consonant.

The detailed description of the algorithm may be found at :

www http://www.highprogrammer.com/alan/numbers/soundex.html

The first letter is simply the first letter in the word. The remaining numbers range from 1 to 6, indicating different categories of sounds created by consanants following the first letter. If the word is too short to generate 3 numbers, 0 is added as needed. If the generated code is longer than 3 numbers, the extra are thrown away.

Code	*Letters Description*
1	*B, F, P, V Labial*
2	*C, G, J, K, Q, S, X, Z Gutterals and sibilants*
3	*D, T Dental*
4	*L Long liquid*
5	*M, N Nasal*
6	*R Short liquid*
SKIP	*A, E, H, I, O, U, W, Y Vowels (and H, W, and Y) are skipped*

There are several special cases when calculating a soundex code:

- *Letters with the same soundex number that are immediately next to each other are discarded. So Pfizer becomes Pizer, Sack becomes Sac, Czar becomes Car, Collins becomes Colins, and Mroczak becomes Mrocak.*
- *If two letters with the same soundex number seperated by "H" or "W", only use the first letter. So Ashcroft is treated as Ashroft.*

Sample Soundex codes:

Word	*Soundex*
Washington	*W252*
Wu	*W000*
DeSmet	*D253*
Gutierrez	*G362*
Pfister	*P236*
Jackson	*J250*
Tymczak	*T522*
Ashcraft	*A261*

Exercise 3.9 Write a program that takes the name entered as `argv[1]` and prints the corresponding soundex code for it. ■