Computation I 5EIA0
Exercise 3: Synesthesia & Double Dutch
(v1.7, September 18, 2020)
Deadline Wednesday 23 September 23:55

Synesthesia is "the production of a sense impression relating to one sense or part of the body by stimulation of another sense or part of the body." In particular, for about 1% of the population this translates into the phenomenon where all letters and digits have a colour, even when they are printed black and white. (Other manifestations include hearing colours, seeing sounds, etc.)

Figure 1: The effect of synesthesia, see `https://www.youtube.com/watch?v=KZ1ljTj6zhE`

In this exercise you will first write a program that uses the colour LEDs on the PYNQ board to display different colours when printing text.

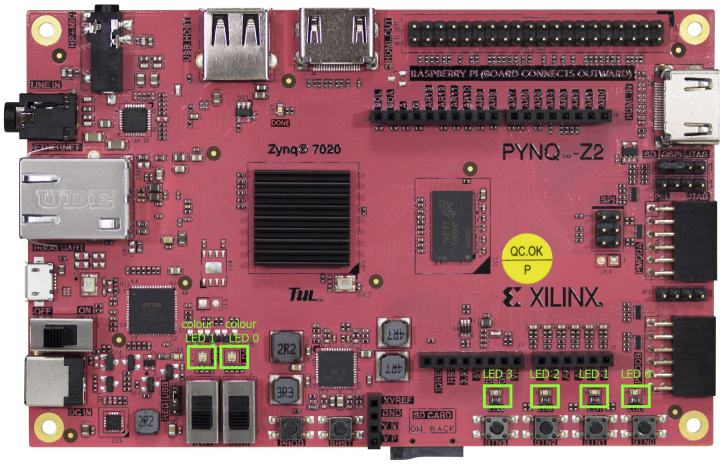

Figure 2: The colour LEDs on the PYNQ board.

The printed text will be a variant of the game known as "Double Dutch"
(see `https://www.youtube.com/watch?v=a8tPGzn10eY`). In this game, players replace specific characters with other characters or they insert additional characters after a specific character. For example the sentence "Mary had a little lamb" translated to "Mumarugyub hutchadud a lulisquatutlule lulamumbub" in Double Dutch. As you can see on the Internet, there exist many variants of this game and they often have different rules on how characters need to be replaced and reordered. In this exercise, you will develop step-by-step one such a variant of this game.

Developing all this in one go is not easy, and we will therefore split the problem into smaller steps. First you will light up the colour LEDs on the PYNQ board with different patterns.
After that you will develop the Double Dutch game in a few steps, and finally the LEDs and game are combined.

If you have problems with the PYNQ board then you can start with task 6 and return to tasks 1 to 5 (the LEDs) later.

The RGB (red, green, blue) colour model is commonly used, including on the PYNQ for both the colour LEDs and the display (which we will see in a later exercise). As Figure 3 shows, colours can be made by mixing three basic components: red, green, and blue. Each colour has an intensity from 0 to 255 (inclusive). Black is red = green = blue = 0 (i.e. no light), and white is red = green = blue = 255 (i.e. all colours). See Wikipedia `https://en.wikipedia.org/wiki/RGB_color_model` for more details.

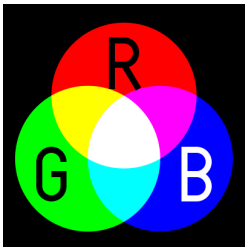

Figure 3: The RGB colour space.

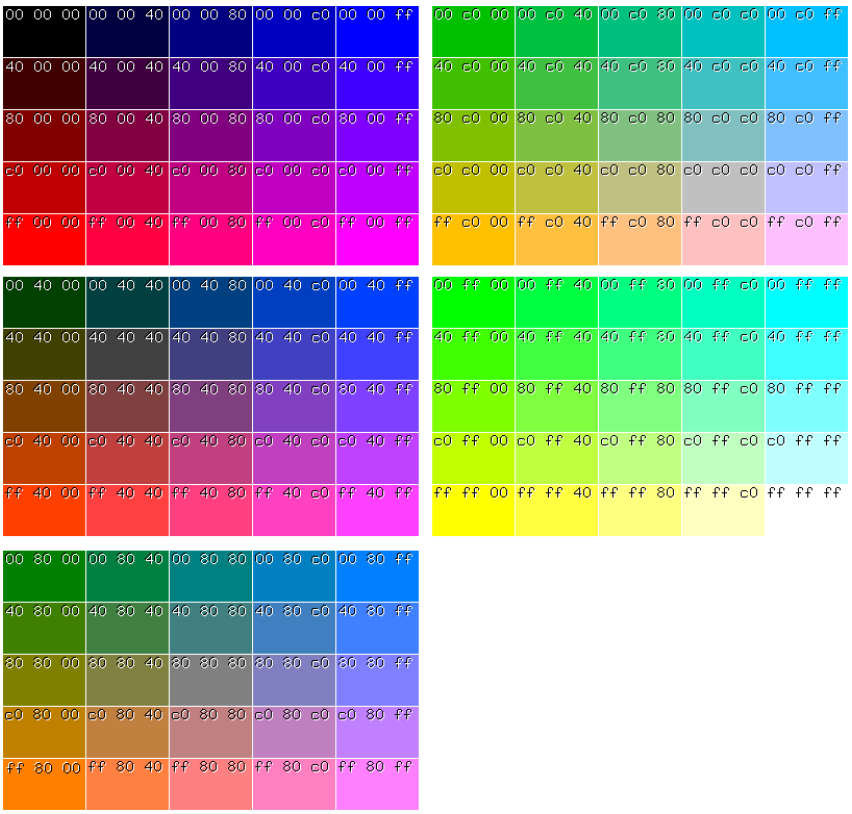

Figure 4: Hexadecimal 8-bit RGB representations of the main 125 colours.

Task 1. Create a new file `1-colours-cycle.c`. The first task is to write a program that cycles through the colours of the first colour LED. The colour LEDs are numbered from 0 to 1. The `libpynq` library of the PYNQ board contains the following relevant declarations:

```
int nrcolourleds;
int nrcolours;
char *colours[]; // { "red", "green", "blue" }
int red;    // 0
int green;  // 1
int blue;   // 2
void led_colour (int led, int colour, int onoff);
```

To switch on red of colour LED 0 you should call: `led_colour(0,red,on);` To switch on green of colour LED 0 you should call: `led_colour(0,green,on);` To switch on blue of colour LED 0 you should call: `led_colour(0,blue,on);` When you switch on all three, the result is white light.

Write a program that cycles through the three primary colours (red then green then blue then red, etc.) with a small delay (e.g. 500 milliseconds). In your first attempt you may find that the LED becomes white. Why?

Task 2. Create a new file `2-buttons-colour-led.c`. Write a program that lights up a colour LED when a button is pushed. Button 0 corresponds to red, button 1 to green, and button 2 to blue. To be able to mix colours it must be possible to press multiple buttons at the same time. It must also be possible to release buttons in a different order from which they are pushed. For example:

```
Press button 1 (green) -- the LED should be green
Press button 2 (blue) -- the LED should be green+blue
Release button 1 (green) -- the LED should be blue
Release button 2 (blue) -- the LED should be off
```

You can check the state of all buttons and light the right colour in a loop:
`for (button=0; button<3; button++) ....`

Task 3. Now that you know how to light the colour LEDs, let's use them to simulate synesthesia. This means that we light the LEDs when we print a string of characters.
Create a new file `3-synesthesia.c`, for which you can use the following code.

```
#include <stdio.h>  // for printf, scanf, putchar, etc.
#include <string.h> // for strlen
#include <ctype.h>  // for tolower
#include "libpynq.h"
void printchar (char c) {
  putchar(c);
  fflush(NULL);
  sleep_msec(100);
}
void printstring (char s[]) { .. to be written by you .. }
int main (void) {
  printstring("Hello there!\n");
}
```

(The command `void fflush(NULL);` is required to print one character at a time on the screen. Otherwise, for efficiency, all characters are saved up until a newline and then printed one line at a time. (See the screencast on Oncourse about printing to the terminal for more details.) The small delay allows you to see the colours better; feel free to change the duration.)

First write a function `void printstring(char s[]);` that uses `printchar` to print the argument string one character at a time. Recall that strings are terminated by the NULL character `\0`, i.e. you should stop printing when you encounter it. Newlines, i.e. the character `\n`, should be printed.
When running the program the string is printed one character at a time, with a small delay between the printing of each character.

```
Hello there!
```

Task 4. Now extend the `printchar` function to set colour LED 0 to

- red if the character that is printed is a consonant (bcdfghjklmnpqrstvwxyz, BCDFGHJKLMNPQRSTVWXYZ)
- blue if the character that is printed is a vowel (aeiou, AEIOU)
- green if the character that is printed is a digit (0-9)
- white if the character that is printed is the NULL character `\0`
- and off otherwise.

Write three functions

```
int isconsonant(char c) { ... }
int isvowel(char c) { ... }
int is0to9(char c) { ... }
```

that you can call in `printchar`. The function `tolower` that is part of the `ctype.h` library may be useful. Kernighan and & Ritchie Appendix B2 contains information on the library. Typing `man tolower` in the terminal also returns the manual page for the `tolower` function.
Define two strings:

```
  char string1[] = "Ooooooh, 0a0a0a0a0a0a0a, --x--x--x--, what is happening now?\n";
  char string2[] = "D.I.S.C.O. D.I.S.C.O. D.I.S.C.O. D.I.S.C.O.\n";
  printstring(string1);
  printstring(string2);
```

Compile and run the program. Colour LED 0 should flash and you should get the following output:

```
Ooooooh, 0a0a0a0a0a0a0a, --x--x--x--, what is happening now?
D.I.S.C.O. D.I.S.C.O. D.I.S.C.O. D.I.S.C.O.
```

Are the colours what you expect?

Task 5. We now use the green LEDs too. The ASCII value of a character is between 0 and 255 (inclusive). You can convert this decimal value into its binary equivalent and light LED i when bit i is 1. This is much simpler than you may think! C has binary operators AND `&`, OR `|`, shift left `<<`, shift right `>>`. Here are some hints on how to check whether bits of a character are zero or one.

```
char c = 'A';
if (c & 1) printf("bit 0 is 1\n");
if (c & 2) printf("bit 1 is 1\n");
if ((c >> 0) & 1) printf("bit 0 is 1\n");
if ((c >> 2) & 1) printf("bit 2 is 1\n");
```

Some examples: since the character 'A' has ASCII value 65, i.e. `01000001` in binary, only LED 0 should be lit. Character 'B' has ASCII value 66 (`01000010`) and only LED 1 should be lit, and for 'C', both LED and 0 should be lit. Character 'o' has ASCII value 79 (`01001111`) and thus all four LEDs are lit.
Rather than writing out the same line for four LEDs, consider using a loop.

Task 6.
Now that we have the synesthesia part working, let's do some experiments on characters, strings, and arrays to really understand the differences. Copy your program to a new file `4-string-fun.c`. Modify the strings in the program to the following:

```
char string1[96] = "Ooooooooh, what is happening now?\n\0Invisible!\n";
char string2[96] = "My telephone number is 06 0123456789\n\0Give me a call!\n";
printstring(string1);
printstring(string2);
```

Notice the explicit length of 96 in the declaration, and the `\0` NULL and `\n` newline characters in the strings. Your program should now produce the following output:

```
Ooooooooh, what is happening now?
My telephone number is 06 0123456789
```

Why are `Invisible!` and `Give me a call!` not printed?

Task 7. Write a function `void printarray (char s[], int length);` that prints the first `length` characters in the array using the printchar function. Modify and run your program:

```
char string1[96] = "Ooooooooh, what is happening now?\n\0Invisible!\n";
char string2[96] = "My telephone number is 06 0123456789\n\0Give me a call!\n";
printstring(string1);
printarray(string1,96);
printstring(string2);
printarray(string2,96);
printarray(string2,strlen(string2));
```

Explain the output of the program:

```
Ooooooooh, what is happening now?
Ooooooooh, what is happening now?
Invisible!
My telephone number is 06 0123456789
My telephone number is 06 0123456789
Give me a call!
My telephone number is 06 0123456789
```

The colour LED white should have been white for quite some time. Why?

Now that you are familiar with the difference between characters and strings, it is time to start the development of our game. As a first step, you will now develop several functions that are useful for our game. To test the correctness of each of these function you will have to modify the `main` function each time you complete a task. Make sure that each function is correct before proceeding to the next one, otherwise it is much harder to debug the program as a whole.
You will write an interactive program that you can give commands. These are the commands that your program will support.

command	operation
q	quit program
s	enter a string
p	print string using printstring
a	print string using printarray
r	reorder string
o	find first occurence of character in string
R	replace characters in string
i	insert a character in string
1	run rule 1 on string
2	run rule 2 on string
3	run rule 3 on string

All of the C exams will use a similar style, so it's a good idea to do this exercise yourself.

Task 8. Copy your program to a new file `5-double-dutch.c`. Make a loop in your main function in which you ask for a single character command, like this:

```
char cmd;
do {
  printf("Command [qsparoRi123]? ");
  // the leading space in the format string means:
  // read the first non white-space character
  scanf(" %c",&cmd);
  switch (cmd) {
  case 'q':
    printf("Bye!\n");
    break;
  default:
    printf("Unknown command '%c'\n",cmd);
    break;
  }
} while (cmd != 'q');
```

Also implement the 'p' command to print the string using the `printstring` function, and the 'a' command to print the string using the `printarray` function. Print the error message shown below if an invalid command is given:

```
Command [qsparoRi123]? A
Unknown command 'A'
Command [qsparoRi123]? p
The current string is ""
Command [qsparoRi123]? a
The current array is ""
Command [qsparoRi123]? q
Bye!
```

Hint: ensure that your arrays are initialised. Your program must produce the *exact* output, including all spaces, capitalisation, quotes, etc.

Note that all C exams will use a similar structure to ask for input. It is worthwhile knowing it by heart!

Task 9. Add the 's' command to ask for a string, like this:

```
case 's':
   printf("Please enter a string: ");
   // skip initial white spaces, and then read everything up to the newline
   scanf(" %[^\n]s",myString);
   break;
```

The format string in `scanf` is very versatile (see Kernighan & Ritchie, Appendix B1.2). Specifying `%c` reads a character and specifying `%s` reads a string, which is a sequence of non-white-space characters. A space in the format string tells `scanf` to skip white spaces (e.g. space, tab, newline). Thus `scanf(" %c",&cmd);` reads a single character after skipping white spaces. By specifying `" %[^\n]s"` it first skips white space, and then reads a sequence of characters that does not contain newlines (`'\n'`). In other words, it reads from the first non-white space character up to the newline. (The newline is left in the input buffer, but we will skip over it in any following `scanf`s. See the screencast "Keyboard input from the terminal with getchar & scanf" on Oncourse for more details.)

```
Command [qsparoRi123]? s
Please enter a string:   Bert and Ernie
Command [qsparoRi123]? p
The current string is "Bert and Ernie"
Command [qsparoRi123]? a
The current array is "Bert and Ernie"
Command [qsparoRi123]? q
Bye!
```

Task 10. Write a function `insertChar(char str[], char aChar, int index)` that inserts non-space character `aChar` at position `index` in the string `str`. Add the command 'i' (insert), and read the necessary function arguments from the keyboard.

```
Command [qsparoRi123]? s
Please enter a string: I like writing C code
Command [qsparoRi123]? p
The current string is: "I like writing C code"
Command [qsparoRi123]? i
Insert which (non-space) character? d
At what index? 6
Command [qsparoRi123]? p
The current string is: "I liked writing C code"
Command [qsparoRi123]? q
Bye!
```

Task 11. Write a function `int findFirstOccurrence(char str[], char aChar)` that searches for a specific character `aChar` in the string. The return value of this function is the index in `str` of the first occurrence of the character `aChar`. When `aChar` is not found, the value `-1` is to be returned. Add the 'o' (occurrence) command, and read the necessary function arguments from the keyboard.

```
Command [qsparoRi123]? s
Please enter a string: whatever
Command [qsparoRi123]? o
Find first occurrence of which (non-space) character? a
The first occurrence of 'a' is at index 2
Command [qsparoRi123]? o
Find first occurrence of which (non-space) character? e
The first occurrence of 'e' is at index 4
Command [qsparoRi123]? o
Find first occurrence of which (non-space) character? x
The first occurrence of 'x' is at index -1
```

Task 12. Write a function `replaceChars(char str[], char sChar[], char rChar)` that replaces all occurrences of the characters in the array `sChar` in the string `str` by the character `rChar`. The return value is the number of replaced characters. Add the 'R' (replace) command and read the strings `myString` and `sChar` and the character `rChar` from the keyboard.

```
case 'R':
  printf("Replace which (non-space) characters? ");
  scanf("%s",replace);  // read a sequence of non-space characters
  printf("with which (non-space) character? ");
  scanf(" %c",&c);      // skip leading spaces
  replaceChars(myString, replace, c);
  break;
```

```
Command [qsparoRi123]? s
Please enter a string: I like writing C code
Command [qsparoRi123]? p
The current string is: "I like writing C code"
Command [qsparoRi123]? R
Replace which (non-space) characters? ioe
with which (non-space) character? -
Command [qsparoRi123]? p
The current string is: "I l-k- wr-t-ng C c-d-"
Command [qsparoRi123]?
```

Task 13. Write a function `void stringReorder(char str[], int index1, int index2)` that divides a string in three parts, and puts them together in a different way. The string `str` is cut at the positions `index1` and `index2`. Recall that array indices start at zero. After splitting the string into substring1, substring2, substring3 they are reordered to substring3, substring2, substring1. Add the 'r' (reorder) command to your main loop, and read `str`, `index1` and `index2` from the keyboard, such that the output looks like this:

```
Command [qsparoRi123]? s
Please enter a string: Ernie and Bert
Command [qsparoRi123]? r
Please enter two indices? 5 10
Command [qsparoRi123]? p
The current string is "Bert and Ernie"
```

In this example the three parts are "Ernie " (notice the trailing space), "and " (notice the trailing space), and "Bert".

Hint: You can use `strncpy` or `strcpy`, and `strcat` from the `string.h` library. `strcpy` copies the entire second string argument, `strncpy` is the same except that at most n characters are copied. See Kernighan & Ritchie Appendix B3) or `man strncpy` for more information. **It is useful to get to know these functions because you can also use them in the exams.**
As an example, the function call `strncpy(myOtherString, &myString[4], 6)` copies a string of at most 6 characters from `myString[4]` to `myOtherString`.
Note that `str(n)cpy` and similar functions *do not support copying overlapping source and destination strings*. See K&R B3.

Some more test cases:

```
Command [qsparoRi123]? s
Please enter a string: 0123456789
Command [qsparoRi123]? r
Please enter two indices? 4 10
Command [qsparoRi123]? p
The current string is: "4567890123"
Command [qsparoRi123]? s
Please enter a string: 0123456789
Command [qsparoRi123]? r
Please enter two indices? 0 4
Command [qsparoRi123]? p
The current string is: "4567890123"
Command [qsparoRi123]? s
Please enter a string: 0123456789
Command [qsparoRi123]? r
Please enter two indices? 0 10
Command [qsparoRi123]? p
The current string is: "0123456789"
Command [qsparoRi123]?
```

After completing these tasks, you have a set of basic functions available that you can use to implement the "Double Dutch" game. Despite the fact that you have these functions available, it is still a complex task to implement the game. Therefore you should use a step-by-step approach and implement the game in small steps. Make sure that you can test the program after every step to spot programming errors quickly.

Task 14. Implement inside the `main` function a program that asks the user to input a string. Once the user has input this string, the program should output its translation to Double Dutch. This translation needs to follow the following rules:

- **Rule 1, command '1'**: Replace all lower case vowels (a, e, i, o, u) with the letter 'a'.
- **Rule 2, command '2'**: Add to each word that starts with a consonant the letters "ay" at the front of this word.
- **Rule 3, command '3'**: Place the first two words at the end of the sentence.

An example output of your program should look as follows:

```
Command [qsparoRi123]? s
Please enter a string: I like writing C code
Command [qsparoRi123]? p
The current string is: "I like writing C code"
Command [qsparoRi123]? 1
Command [qsparoRi123]? p
The current string is: "I laka wratang C cada"
Command [qsparoRi123]? 2
Command [qsparoRi123]? p
The current string is: "I aylaka aywratang ayC aycada"
Command [qsparoRi123]? 3
Command [qsparoRi123]? p
The current string is: "aywratang ayC aycada I aylaka"
Command [qsparoRi123]? q
Bye!
```

The colour LED and the green LEDs should light up according to the synesthesia rules when you print the Double Dutch translation.

> **Hint:** Make use of the functions that you have developed in previous tasks to implement your program. For example, in step 3 you should use the string reordering function.

Submission: Submit your file `5-double-dutch.c` that implements the last task on Oncourse (`Exercise 3: Synesthesia & Double Dutch (due 25 Sept 23:55)`). You can resubmit as often as you want until the deadline.

- 2/9 v1.1: Corrected >>.
- 20/9, v1.2: Made exercise similar to how exams are structured.
- 3/9, v1.3: Simplified the string inputs with scanf.
- 7/9, v1.4: Minor corrections.
- 8/9, v1.5: Note on skipping LED tasks initially.
- 15/9, v1.6: Added warning on overlapping str(n)cpy, and screencast on terminal input.
- 18/9, v1.7: Minor corrections.