Computation I 5EIA0
Exercise 3: Synesthesia & Double Dutch (v1.1, September 16, 2019)
Deadline Wednesday 25 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.

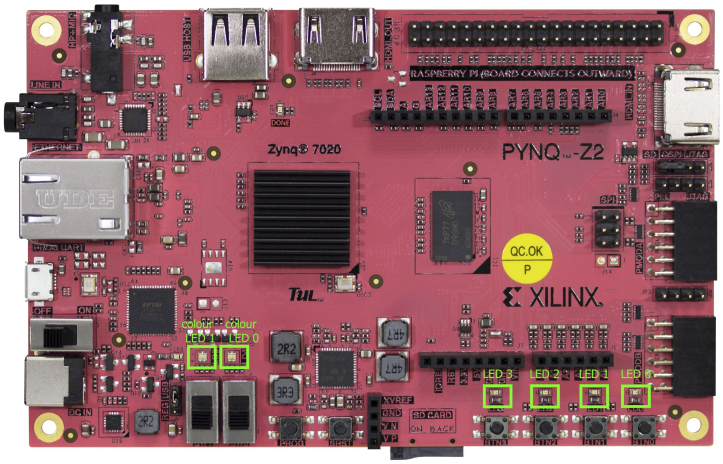

Figure 2: The colour LEDs on the PYNQ board.

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.

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.

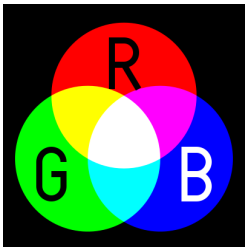

Figure 3: The RGB colour space.

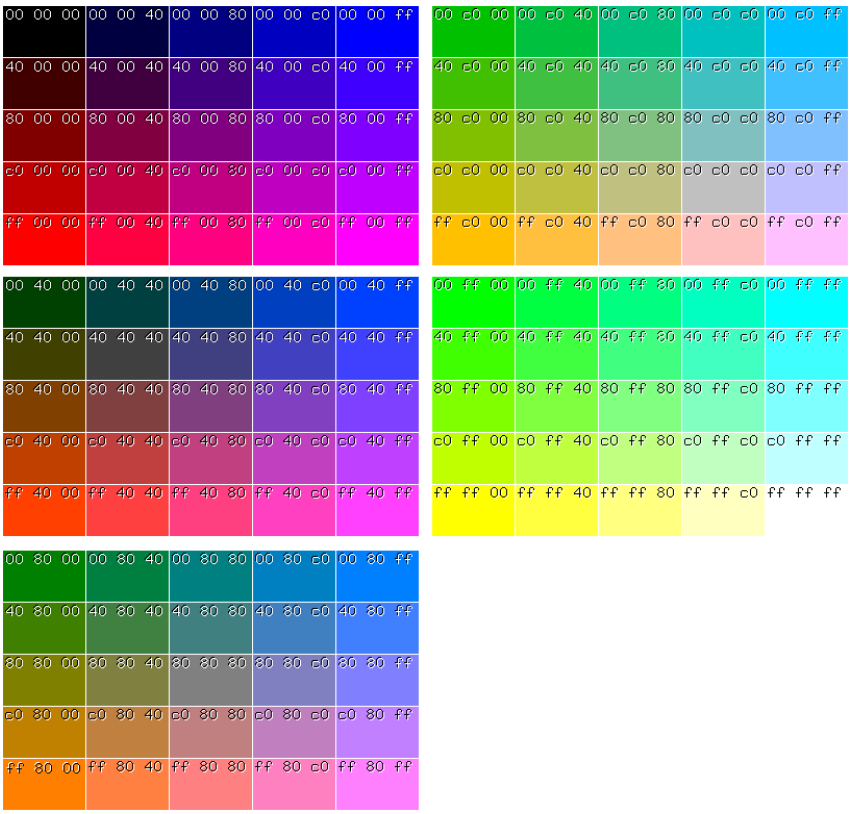

Figure 4: Hexadecimal 8-bit RGB representations of the main 125 colours.

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, green, or blue) 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. 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.

```
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.
Define two strings:

```
  char string1[] = "Ooooooh, what is happening now?\n";
  char string2[] = "My telephone number is 06 0123456789\n";
  printstring(string1);
  printstring(string2);
```

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

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

Task 5. We now use the green LEDs too. The ASCII value of a character is between 0 and 255. 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. Rather than writing out the same line for four LEDs, consider using a loop.

```
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.

Task 6.

Now that we have the synesthesia part working, let's do some experiments on characters, strings, and arrays to really understand their differences. Copy your program to a new file `4-string-fun.c`. When the game is started, the user should be greeted properly. Extend your C program to print the following text to the terminal:

```
** Welcome to the Double Dutch game **
```

Modify the strings in the program to the following:

```
char string1[96] = "Ooooooh, what is happening now?\n\0Invisible!";
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 characters in the strings. Your program should now produce the following output:

```
Ooooooh, 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] = "Ooooooh, what is happening now?\n\0Invisible!";
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:

```
Ooooooh, what is happening now?
Ooooooh, 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
```

Why was the colour LED white for quite some time?

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. As part of this process you may have to delete the content of the `main` function several times. Instead of deleting the code that you no longer need, you may of course also place this code inside a comment (i.e., in between `/*` and `*/`). 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.

Task 8. Copy your program to a new file `5-double-dutch.c`. 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 being cut at the positions `index1` and `index2`, and the last part is placed at the start, the first part is placed in the middle and the middle part is placed at the end. Use `strncpy` and `strcat` to achieve that. Read `str`, `index1` and `index2` from the keyboard.

```
Please enter a string:  and Ernie Bert
Please enter two indices: 5 10
Part 1:  and
Part 2: Ernie
Part 3:  Bert
Result:  Bert and Ernie
```

(Note that there is a space at the start of the input: " and Ernie Bert")

Hint: The function call `strncpy(myOtherString, &myString[4], 6)` copies a string of 6 characters from `myString[4]` to `myOtherString`. This function can be used inside the function `stringReorder`. To use `strncpy` the header file `<string.h>` must be included, i.e., add the following line to the top of the source file.

```
#include <string.h>
```

Task 9. Write a function `int findFirstOccurance(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` has not been found, the value `-1` is to be returned. Read the necessary function arguments from the keyboard.

Example

```
char myString[] = "whatever";
index = findFirstOccurance(myString, 'a');  /* index = 2 */
index = findFirstOccurance(myString, 'e');  /* index = 4 */
index = findFirstOccurance(myString, 'x');  /* index = -1 */
```

Task 10. Write a function `replaceChars(char str[], char sChar[], char rChar)` that replaces all occurrences of any characters in the array `sChar` in the string `str` by the character `rChar`. The return value is the number of replaced characters. Hint: The `findFirstOccurance` function can be used here. Read the strings `myString` and `sChar` and the character from the keyboard.

```
Please enter a string: I like writing C code
Which characters to replace? ic
With which character? -
The new string is: I l-ke wr-t-ng C -ode
```

Task 11. Write a function `insertChar(char str[], char aChar, int index)`, that inserts character `aChar` at position `index` in the string `str`.

```
Please enter a string: I like writing C code
Which characters to insert? d
At which position? 6
The new string is: I liked writing C code
```

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 12. 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:

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

An example output of your program should look as follows:

```
** Welcome to the Double Dutch game **
Please enter a string: I like writing C code
Double Dutch translation: aywratang ayC aycada I aylaka
```

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. Implement the program using a step-by-step approach (e.g., implement the rules one by one and test the program each time you implemented a new rule). Also note that you have to insert an extra space between the last word in the original sentence and the words you place at the end.

Task 13. Finally, combine the `printstring` function developed for synesthesia with your double Dutch program.

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. You must also demonstrate that your program works to one of the teaching assistants during one of the labs, before Thursday 26 September 12:30.

- **2/9 v1.1** Corrected >>.