UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08022 COMPUTER PROGRAMMING SKILLS AND CONCEPTS

Thursday 12 th December 2013

09:30 to 12:30

INSTRUCTIONS TO CANDIDATES

1. **Answer all questions.**
2. **Consult the separate printed sheet of instructions for details of how to get the files for the exam and submit your answers.**
3. **Sections A and B each account for half the marks.**
4. **Questions in section A are all worth 10 marks, but are not necessarily of equal difficulty. You are advised to answer them in order.**
5. **The two questions in section B are of approximately equal difficulty.**

Convener: J. Bradfield
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

Section A

This section contains five short questions worth 10 marks each, in which you are asked to implement either one function of a program, or a short complete program. **Little credit will be given for incomplete solutions.**

1. Complete the program `hextimes.c` so that it prints out a multiplication table for the *hexadecimal* numbers 0 to F. That is, the table should contain 16 rows and 16 columns, and the jth entry in the ith row (starting from 0) should be the number $i \times j$, printed in hexadecimal. The table should be neatly aligned, with the numbers padded on the left with spaces as necessary. [*10 marks*]

   When you are ready, submit the file `hextimes.c`

2. The *digital root* $\mathrm{dr}(n)$ of a non-negative integer n is a calculation used in ancient fortune-telling and modern puzzles. $\mathrm{dr}(n)$ is calculated as follows: add up the decimal digits of n to obtain n'. Set n to n', and repeat the process, until you have a one-digit number. That is $\mathrm{dr}(n)$. For example:

   $$65536 \to 6 + 5 + 5 + 3 + 6 = 25 \to 2 + 5 = 7$$

   so $\mathrm{dr}(65536) = 7$.

   Complete the program `root.c` so that it reads an integer from the user, computes its digital root, and prints it. You are given a skeleton program that does the reading and printing.
   Hint: The last digit of `n` is `n % 10`. [*10 marks*]

   When you are ready, submit the file `root.c`

3. The Royal Mail classifies postal items as *Letters*, *Large Letters*, or *Parcels*, according to their weight and size. The rules are:

   - If the weight is at most 100 g, the length at most 240 mm, the width at most 165 mm, and the thickness at most 5 mm, then the item is a Letter.
   - If the weight is at most 750 g, the length at at most 353 mm, the width at most 250 mm, and the thickness at most 25 mm, then the item is a Large Letter.
   - Otherwise, the item is a Parcel.

   In the file `mail.c`, you will find type definitions for a structure type representing items, and for the classes of items. Complete the function
   `MailClass classify(MailItem)`
   which takes an item of mail, and returns its class. [*10 marks*]

   The main routine contains some test cases; you are free to extend these if you wish, but code added there will not be marked.

   When you are ready, submit the file `mail.c`

4. In card games, a *riffle shuffle* is a procedure where the pack of cards is divided into two equal halves, and then the two halves are interleaved. Thus if the original order of cards is

$$c_1, c_2, c_3, c_4, \ldots c_{51}, c_{52}$$

the order after riffling will be

$$c_1, c_{27}, c_2, c_{28}, \ldots c_{26}, c_{52}$$

In the file `riffle.c`, complete the function
`void riffle(int a[], int n)`
which riffle-shuffles the first `n` elements of the array `a`. In the case that `n` is *odd*, you should riffle the first `n-1` elements and leave the last element in place.

If you wish to declare temporary arrays, you may assume that `n` is less than or equal to the constant `NMAX`. [*10 marks*]

When you are ready, submit the file `riffle.c`

5. In the computer representation of colours, colours are often specified by an *RGB* triple, three floating point numbers between 0.0 and 1.0. A different specification, which takes account of the different responses of the human eye to red, blue and green light, is the *XYZ* system. An RGB specification (r, g, b) is converted to an XYZ specification (x, y, z) by the following rules:

$$\begin{aligned} x &= 0.412453 \times r + 0.357580 \times g + 0.180423 \times b \\ y &= 0.212671 \times r + 0.715160 \times g + 0.072169 \times b \\ z &= 0.019334 \times r + 0.119193 \times g + 0.950227 \times b \end{aligned}$$

Write a program `rgbtoxyz.c` which reads three floating point numbers from the user, treats them as RGB specifications, and prints the corresponding XYZ specification as three floating point numbers separated by spaces. You should include messages as in the example run below:

```
Input r g b values: 0.5 0.5 0.5
x y z are:
0.475228 0.500000 0.544377
```

You do **not** need to check for erroneous input. [*10 marks*]

When you are ready, submit the file `rgbtoxyz.c`

Section B

This section contains two longer questions worth 25 marks each. The questions have several parts. Each part of the question is marked independently of any errors in previous parts.

1. In this question, we will use the Descartes graphics library and the maths library. The program we will complete is in the file `lengthjudge.c`, so to compile it you need to do

   ```
   gcc -Wall lengthjudge.c descartes.o -lSDL -lm
   ```

   The objective of the program is to implement a simple test of the user's ability to judge the length of lines. The program draws ten random lines on the screen, and then the user is asked to click on an endpoint of the shortest line.

   (a) The first task is to complete an auxiliary function. Complete the function `int Near(point_t p1, point_t p2, int dist)` so that it returns true (`1`) if `p1` and `p2` are less than or equal to `dist` apart in both the x and y coordinates, and returns false `0` otherwise. [*5 marks*]

   (b) The second task is to complete the function that draws the random lines.

   The global array `lines` stores the lines (i.e. Descartes `lineSeg_t`s) that have already been drawn, and the number of lines is in `num_lines`.

   Complete the function `DrawNewLine()` so that it chooses two random points on the screen, and checks that the new points are more than `2*NEARNESS` (where `NEARNESS` is a constant defined in the supplied code) away from the start or end points of any of the lines currently in the `lines` array, using the `Near()` function to do this. (It should keep choosing random points until this becomes true.) It should then create a line segment from the two points, draw it, and add it to the `lines` array.

   The Descartes screen is 500 pixels wide and high, and you can obtain a random integer between 0 and 499 by evaluating `random() % 500`. [*10 marks*]

   (c) Now complete the function `int FindShortest(void)` so that it returns the index of the shortest line segment in the `lines` array. You do not need to consider what happens if two lines have equal length. [*5 marks*]

   (d) Finally, complete the function `int ClickedLine(void)`. This function should read a point from the user's click, and see whether it is within `NEARNESS` of an endpoint of one of the lines on the screen. If it is, it should return the index of that line. If not, it should print the message

   ```
   Try again!
   ```

   and repeat, until the user clicks near an endpoint. [*5 marks*]

When you are ready, submit the file `lengthjudge.c`

2. This question is about the *Playfair cipher*, a cipher invented in 1854 by Charles Wheatstone, used up until WW2, and whose solving is demonstrated in Dorothy L. Sayers' novel *Have His Carcase*.

   The cipher works on an alphabet of 25 upper case letters – the letter 'J' is ignored, and replaced by 'I' whenever it occurs in the source text. A 5×5 grid is filled with the alphabet according to a *key* word, and then the encryption uses this grid to change letters according to a procedure we will consider in part (b). For technical reasons, the encryption procedure sometimes inserts extra 'X's or 'Q's into the input text; these additions will appear on decryption, and are left to the human reader to filter out.

   You are provided with a file `playfair.c`, containing three functions that you must complete for this question. The compiled program is used as follows:

   - To encipher with the default key, call `./a.out` and type your message. Non-letters in the input are ignored.
   - To decipher with the default keyword, call `./a.out -d` and type the ciphered message. Non-letters in the input are ignored.
   - To specify the key, call, for example, `./a.out -key mykey` (remembering that if you have spaces in your key, you should enclose your key in single quotes).
   - If you add the `-test` option, e.g. `./a.out -test -d -key mykey`, you will get a menu for testing or diagnosing your answers.

   (a) The first task is to write the function `char sanitize(char c)`, which cleans up the input text ready for ciphering. Complete the function as follows:

   - if `c` is a letter, it should be converted to uppercase, and if the letter is `'J'`, changed to be `'I'`, and the result returned;
   - otherwise `c` is returned.

   [*5 marks*]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

(b) The second task is to implement the function
`void cipher(char c1, char c2, int decipher)`
which ciphers and prints a pair of upper-case letters. The procedure to *encipher* (when `decipher` is false) is:

- Look up `c1` and `c2` in the key tables: set `i1` to `row[c1]` and `j1` to `col[c1]`, which give the row and column of the 2-dimensional array `key` where `c1` occurs. Similarly, let `i2, j2` be the row and column of `c2`. (The `key`, `row` and `col` arrays are described further in part (c).)
- If the letters are in the same row (`i1 == i2`), then print the letters one column to the right (i.e. print the letters in `key[i1,j1+1]` and `key[i2,j2+1]`. (If one of the input letters is at the end of the row, wrap round to the beginning of the same row.)
- If the letters are in the same column (`j1 == j2`), then print the letters one row down, wrapping round to the top if necessary.
- Otherwise, print the letter at `i1,j2`, then the letter at `i2,j1`.

The case where `c1 == c2` is handled by the supplied code as an error, since the caller of `cipher()` should ensure that this does not happen.

The procedure for *deciphering* (when `decipher` is true), is the same except that if the input letters are in the same row, we print the letters one to the left, and if they are in the same column, we print the letters one row up. **NOTE:** To calculate 'one column to the left, wrapping round', you should use `(j1 + 4) % 5`, *not* `(j1 - 1) % 5` (because negative numbers give negative remainders in C). [*10 marks*]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

(c) The final task constructs the key table from a supplied keyword. The key is kept in the global variable `key` which is a two-dimensional 5×5 array of characters. Complete the function `void make_key(char *keyword)` so that it fills in the key table thus: the table is filled left to right along each row (top to bottom), using first the letters in the keyword (ignoring any duplicates), and then the remaining letters in alphabetical order (ignoring 'J', of course). The letters in the keyword must be sanitized before use; and any non-letters are ignored.

As you store letter `c` in the key, save the current row value in `row[c]` and the current column value in `col[c]`.

For example, the keyword `PLAYFAIR` gives the key table

```
P L A Y F
I R B C D
E G H K M
N O Q S T
U V W X Z
```

and then `row['C']` will be `1`, and `col['C']` will be `3`. [*10 marks*]

When you are ready, submit the file `playfair.c`