FOR INTERNAL SCRUTINY (date of this version: 17/12/2015)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

COMPUTER PROGRAMMING SKILLS AND CONCEPTS

Tuesday 1st April 2014

00:00 to 00:00

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.**

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

FOR INTERNAL SCRUTINY (date of this version: 17/12/2015)

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. The 'Body Mass Index' or 'BMI' is a crude measure of how thin or fat a person is. It is defined as $b = w/h^2$, where w is the person's weight (in kilogrammes), and h is their height (in metres). An alternative measure is the 'Corpulence Index' or 'CI', defined as $c = w/h^3$.

   Write a program `bmi.c` which reads from the user a height **in centimetres** and a weight in kilogrammes, and prints the height in centimetres without decimal places, the weight in kilogrammes to 1 decimal place, and the resulting BMI and CI to one decimal place. Your program should accept floating point inputs from the user. Your program should print prompts exactly as in the following example run:

   ```
   Enter height in centimetres: 176
   Enter weight in kilogrammes: 69.5
   A weight of 69.5 kg and a height of 176 cm
   give a BMI of 22.4 and a CI of 12.7
   ```

   [*10 marks*]

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

2. One part of the British imperial system that is still quite common in the UK is the use of feet and inches for measuring the height of a person. There are 12 inches in a foot, and one inch is exactly 2.54 cm.

   Complete the program `height.c` which prints a conversion table from feet and inches to centimetres. The number printed in row i (counting from zero), column j (counting from zero) of the table should be the centimetre equivalent of i feet and j inches. i should go from 0 to 7, and j from 0 to 11. Each column should be five characters wide, and the cm figures should be **rounded** to the nearest integer. You do **not** have to print labels for the rows and columns.

   If you have implemented the program correctly, the first four lines of the output will be:

   ```
     0    3    5    8   10   13   15   18   20   23   25   28
    30   33   36   38   41   43   46   48   51   53   56   58
    61   64   66   69   71   74   76   79   81   84   86   89
    91   94   97   99  102  104  107  109  112  114  117  119
   ```

   [*10 marks*]

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

3. The Collatz or '$3n + 1$' problem is a famous unsolved problem in number theory. Here is a traditional statement:

> Think of a positive integer, and call it n. Now if n is even, divide it by 2; if n is odd, multiply it by 3 and add 1. Repeat this process until n becomes 1, if it ever does. Is it the case that n always becomes 1?

Collatz conjectured in 1937 that n always does become 1, but to this day, we don't know. (It has been checked for all numbers up to about 10^{18}.)

Complete the program `collatz.c` so that it reads a number from the user, and then applies the Collatz procedure, printing n on a new line at each stage until n has reached 1. (It should include the original value of n as the first line of output.) The reading of the input has been done for you.

Here is an example of a run:

```
Please input a positive integer: 6
6
3
10
5
16
8
4
2
1
```

[*10 marks*]

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

4. In many disciplines of target shooting, there are ten concentric rings on the target, with scores of 10 (innermost ring) down to 1 (outermost ring), or zero if the target is missed entirely. A 'card' has ten shots on it, and the score is the sum of the scores of the shots.

   There is also a small 'dot' within the innermost ring, which is used to break ties: if two cards have the same numerical score, the card with more shots in the dot wins.

   In the file `target.c`, you will find type definitions to record the details of a card: a struct type `shot_t` for the score of one shot, and a struct type `card_t` with the details of the ten shots for the card. Complete the function
   `bool_t beats(card_t c1, card_t c2)`
   so that it calculates the scores as described above, and returns `TRUE` if `c1` is a (strictly) better card than `c2`.

   You may assume that cards contain valid scores. [*10 marks*]

   The main routine contains a few test cases; you are free to extend these if you wish. Only code in the marked area will be graded.

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

5. A *pangram* is a sentence that contains every letter of the alphabet at least once (ignoring case). A well known example is 'The quick brown fox jumps over the lazy dog.'

   Complete the program `pangram.c`, so that after reading a sentence from the user, it checks whether the input is a pangram, and prints 'Your sentence is a pangram.' or 'Your sentence is not a pangram.' The reading of the sentence from the user has already been done for you; the sentence is stored in the character string `buf`.

   The program contains in a comment some pangrams and non-pangrams that you can cut and paste into the terminal.

   Hint: Remember that the character `'A'` (for example) has the integer value 65; if you are counting uppercased characters, you may find it useful to use `c - 'A'` to convert character `c` to its offset in the alphabet. [*10 marks*]

   When you are ready, submit the file `pangram.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. The program we will complete is in the file `polydraw.c`, so to compile it you need to do
`gcc -Wall polydraw.c descartes.o -lSDL -lm`

   Summaries of the Descartes functions are on the quick reference sheet, and also in the comments in the `descartes.h` file.

   The objective of the program is to test the user's ability to draw equilateral polygons freehand – while most people can draw triangles and squares, it is surprisingly difficult to draw a pentagon accurately.

   The user is asked for a number n, and then instructed to click n points on the screen to form an equilateral n-gon.

   In the supplied program, you are given a function

```
/* This function draws a small cross at the given point */
void MarkPoint(point_t p);
```

   (a) The first task is to complete the main routine, which asks the user for points, adds them to the array `points`, and draws them on the screen. At the marked point in the main routine, add code to do the following:

   One at a time, read `num_points` points from the user, use `MarkPoint()` to mark each point on the screen, draw a line from the previous point to the current point, and with the last point draw a line to the first point.

   You may assume that the user will behave correctly; if the user middle-clicks (so returning a point at `(-1,-1)`) you may treat this as a valid point. [*8 marks*]

   (b) The second task is to complete the function `Equilaterality()`, which calculates a simple measure of how close to equilateral a polygon is. It takes an array of points, and an integer giving the length of the array.

   Complete the function so that it finds the longest side in the polygon and the shortest side, and returns the ratio of the shortest to the longest, so giving a measure between 0.0 and 1.0 (for fully equilateral).

   You may assume that `n >= 2`, and that at least one side has non-zero length. [*7 marks*]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

(c) The third task is to complete the function `FindNearest()`. This function takes an array `pts` of points, an integer `n` giving the length of the array, and a point `p`. It should return the index in `pts` of the point that is closest to `p` (measured as the length of the line segment from the point to `p`). [*6 marks*]

(d) Finally, complete the function `HighlightPoint()`. This function takes a `point_t p`, and should draw a square of side length 5 around it: that is, if `p` has coordinates (x, y), the function should draw a square with corners at $(x \pm 2, y \pm 2)$. [*4 marks*]

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

2. In this question, you will implement some routines for a simple book catalogue.

   You are provided with two files, a skeleton program `book.c` and a small sample catalogue `book.txt`. In the first two parts, you will use a tiny three-entry catalogue built in to the program; in the third part, you will read in `book.txt`.

   The main program reads the catalogue file if one is given on the command line, and then goes into a loop, allowing you to test the routines you will implement.

   The catalogue is stored in a global array `db` of `book_t` catalogue entries; its maximum size is `DB_SIZE`, and the global variable `num_entries` contains its current size. The struct type `book_t` contains three fields: `author` and `title`, character arrays of length `MAX_LENGTH+1`; and `isbn`, a character array of length `ISBN_LENGTH+1`.

   To make testing easier, all the included examples have been reduced to single-word authors and titles, although the code should not rely on this.

   (a) Your first task is to implement the `Lookup()` function. This function has the type
   ```
   int Lookup(int start, char *author, char *title, char *isbn)
   ```
   The function searches through the catalogue, starting at entry number `start`, looking for the next (including `start` itself) entry that matches all the given arguments, and returns the index of the result, or `-1` if there is no such entry. If `Lookup()` is given an invalid argument – a negative `start` – it should also return `-1`.

   If the `author`, `title` or `isbn` argument is `NULL`, then it is not checked in the search. So, for example,
   ```
   Lookup(0, NULL,"Hobbit",NULL)
   ```
   will search for a book whose title is 'Hobbit'. On the built-in database, this will return `0`.

   The lookup requires an exact match: `"Hobbit"` matches only `"Hobbit"`, not `"hobbit"` or `"The Hobbit"`. [*8 marks*]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

(b) The second task is to implement the `int CheckISBN(char isbn[])` function, which checks that an ISBN (International Standard Book Number) is valid, returning `1` if it is, and `0` otherwise. For simplicity, we consider pre-2007 ISBNs, which had ten characters. The first nine characters are digits '0', ..., '9'; the last character may be a digit or the letter 'X', which is treated as a digit with numerical value 10.

The last digit is a check digit, which works thus: let the ten 'digits' of the ISBN be $x_0x_1 \ldots x_9$. The *checksum* is the value $\sum_{i=0}^{9} x_i(10-i)$, or written out longhand

$$10x_0 + 9x_1 + 8x_2 + 7x_3 + 6x_4 + 5x_5 + 4x_6 + 3x_7 + 2x_8 + x_9.$$

To be valid, the checksum must be zero modulo 11. For example, the ISBN 354063648X has checksum $10 \cdot 3 + 9 \cdot 5 + 8 \cdot 4 + 7 \cdot 0 + 6 \cdot 6 + 5 \cdot 3 + 4 \cdot 6 + 3 \cdot 4 + 2 \cdot 8 + 1 \cdot 10 = 220$, which is a multiple of 11.

Complete the `CheckISBN()` function according to this specification. You should also return `0` if the ISBN is invalid because of invalid characters (a non-digit in the first nine places, or a non-digit other than `'X'` in the last place), or the wrong length.

Hints: The `isbn` field is an array of *characters*, not numbers. A convenient way to convert from a digit character `c` to its value as a digit is `c - '0'`. You must check for `'X'` specially. Recall that `%` is the modulo operator. [*7 marks*]

(c) The third task is to implement the `Update()` function. This function has the type

```
int Update(char *author, char *title, char *isbn)
```

It searches for the first entry that matches `isbn`, and updates the author and title of that entry. If there is no existing entry with that `isbn`, it adds a new entry at the end of the catalogue with the given data, provided that there are currently less than `DB_SIZE` entries in the database.

If the supplied author or title is longer than `MAX_LENGTH` characters, it should be truncated.

If the supplied ISBN is invalid, or the supplied author or title is `NULL`, or the database has no room for a new entry when needed, `Update` should *not* modify the database, but just return `-1`.

If `Update` succeeds, it should return the index of updated or new entry. [*10 marks*]

Once you have completed part (c), you can test it and your other answers on the supplied catalogue `book.txt` (the code to read in the catalogue uses the `Update()` function). To load the catalogue, run your program with

```
./a.out book.txt
```

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