UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

COMPUTER PROGRAMMING SKILLS AND CONCEPTS

Tuesday 20th December 2011

14:30 to 17:30

Convener: J Bradfield
External Examiner: A Preece

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.**

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. You are given a file `arith.c` containing a skeleton program. Complete the main routine to produce a program that demonstrates C integer arithmetic. The program should prompt the user for two integers, and then print the result of applying the C arithmetic operators `+ - * / %` to the two numbers. The output should appear exactly as in the following example execution:

   ```
   Enter first number: 56
   Enter second number: 7
   56 + 7 is 63
   56 - 7 is 49
   56 * 7 is 392
   56 / 7 is 8
   56 % 7 is 0
   ```

   You do **not** need to handle erroneous input.
   Note: Remember that to include a literal `%` in a `printf` format string, it must be doubled. [*10 marks*]

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

2. You are given the file `cards.c`, which contains `enum` and `struct` types for a playing card, together with an incomplete function `int beats(card_t, card_t)`. Read the following specification of `beats()` carefully, and complete the implementation. Note that the main program contains some simple tests for your answer.

   ```
   /* beats(c1,c2) returns 1 (true) if card c1 is better than c2,
      and returns 0 (false) otherwise.
      c1 is better than c2 if:
      the value of c1 is greater than the value of c2; or
      the value of c1 is the same as the value of c2 and the suit of
      c1 is numerically greater than the suit of c2. */
   ```

   [*10 marks*]

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

3. Write a program `times.c` which prompts the user for a number n, and then prints a multiplication square of size n. A multiplication square of size n has n rows, each containing n numbers: the ith row (starting from $i = 1$) contains the values $i, 2 \times i, \ldots, n \times i$. An example run is:

```
Enter size: 5
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
   5  10  15  20  25
```

[*10 marks*]

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

4. In the template file `rev.c`, you can see the function prototype:

```
void reverse(int nums[], int n)
```

together with some `main` code to allow the user to enter numbers from standard input, which are then saved in an array local to `main`.

Implement the `reverse` function so that it reverses the first `n` items of the array `nums`. [*10 marks*]

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

5. In this question we consider the problem of computing income tax with respect to British income tax rules for the tax year 2011/12.

The rule in the UK is that every individual has a personal allowance of value £7475 - any earnings up to this limit are not taxed. The "*taxable income*" for an individual refers to any income after the personal allowance of £7475 is subtracted.

The taxable income has tax calculated as follows:

- The first £35000 of taxable income is taxed at the rate 20% (basic rate).
- The next £115000 of taxable income is taxed at the rate 40% (higher rate).
- Any remaining taxable income is taxed at the rate 50%.

You have been provided with the template file `tax.c`. Please write code to prompt the user for an integer value representing annual salary, to read this value, to calculate the income tax due as a `double`, and then output the tax due (to two decimal places). [*10 marks*]

Your output should appear exactly as in the four following example runs:

```
Enter your earnings for 2011/12: 3000
No tax due for 2011/12.

Enter your earnings for 2011/12: 17475
Your tax for 2011/12 is 2000.00

Enter your earnings for 2011/12: 52475
Your tax for 2011/12 is 11000.00

Enter your earnings for 2011/12: 167475
Your tax for 2011/12 is 58000.00
```

When you are ready, submit the file `tax.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, you are asked to implement a slightly more sophisticated version of the Caesar cipher used in lectures.

   The cipher works thus: the user chooses a keyword, for example *`PENNY`*. The keyword is transformed to a *key sequence* of integers as follows: each letter is interpreted as its numerical position in the alphabet: *`A`* as `1` through to *`Z`* as `26`. Upper and lower case are treated the same, and any non-letters are ignored. Thus the keyword *`PENNY`* (or *`penny`* or even *`Pe+n!n**Y`*) generates the key sequence of integers `16, 5, 14, 14, 25`, which is repeated as often as necessary.

   To encipher a message, it is first converted to uppercase. Then each letter of the message is rotated forwards through the alphabet, by the amount of the corresponding element of the key sequence; non-letters are left unchanged.

Plaintext	T	H	I	S		I	S		A		T	E	S	T
Key sequence	16	5	14	14		25	16		5		14	14	25	16
Ciphertext	J	M	W	G		H	I		F		H	S	R	J

   To decipher a message, the procedure is the same, except that letters are rotated backwards instead of forwards.

   You are provided with a starting file `cipher.c`. You are asked to complete two functions:

   (a) Complete the function `ReadKey()`, according to the specification given above and in the comment before the function. Note in particular that a key letter *`A`* gives a key value of `1`, not `0`.

   Hint: Use `getchar()`, not `scanf()`. [*12 marks*]

   (b) Complete the function `cipher()` according to the specification above and in the comment before the function.

   Note: to do 'a becomes $a - b \mod N$' in C, you should write
   `a = (a + N - b)%N;` and **not** simply `a = (a - b)%N;` [*13 marks*]

   You should test your program. The supplied program deciphers if the command-line argument `-d` is given, and otherwise enciphers. A screen shot of a successful test looks like this (`pula:` is the command prompt from the machine the test was run on):

```
pula: ./a.out
penny
this is a test
JMWG HI F HSRJ
pula: ./a.out -d
penny
JMWG HI F HSRJ
THIS IS A TEST
pula:
```

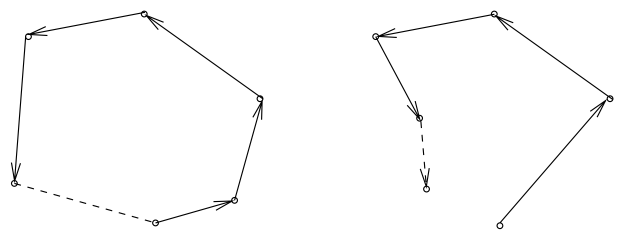
When you are ready, submit the file `cipher.c`

2. In this question we consider the problem of reading a sequence of points through the `descartes` graphics window, drawing the line segments connecting each adjacent pair of points, as long as the sequence defines a *convex polygon.*

   You will create your solution in the template file `convex.c`.

   A single point p in the plane is represented by an x and a y coordinate, $p = (x, y)$. The `descartes` library defines a structured type `point_t` to store points, and a line segment type `lineSeg_t` to store line segments (ordered pairs of points). It also offers a suite of functions for working with points and line segments through a graphics window - all these type declarations and function prototypes can be examined in the file `descartes.h` made available to you. *These also appear on the C Quick Ref sheet.*

   The question of whether a sequence of points is *convex* depends on the order in which the points are given. The following pictures (in which the arrows indicate the order points were entered in) give examples of convex, and also non-convex, sequences of points:



   A sequence of points is convex if exactly one of the following conditions is satisfied:

   - Every three consecutive points form a "left or straight turn", *or alternatively*
   - Every three consecutive points form a "right or straight turn".

   In this question, you will develop a set of functions to help test the direction of a "turn", and then use this to write a `main` function to accept a sequence of points, drawing the new line segment after each point is entered, until the user terminates the input (using a right-click - which passes a point with *negative coordinates*) **or** the program detects the polygon is no longer convex. The program will detect that the polygon is "no longer convex" if the most recent line segment drawn indicates a change in turn direction from "left" to "right" or vice versa. The `main` must either print `No longer convex` or `This was convex` before terminating.

   You will need to compile with the `descartes` library, linking to `SDL`, in doing this question, as follows:

```
gcc -Wall descartes.o convex.c -lSDL
```

   Your tasks are:

(a) Complete the following function to compute the cross-product `p`×`q` of two points `p` and `q`. Recall that for $p = (p_x, p_y)$ and $q = (q_x, q_y)$, the cross product is defined as $p \times q = p_x q_y - p_y q_x$. [*5 marks*]

```
int cross (point_t p, point_t q)
```

(b) Complete the following function to take two points `p` and `q` and return the point `p-q`. [*5 marks*]

```
point_t subtract (point_t p, point_t q)
```

(c) Complete the following function to take three points `p`, `q` and `r` and return 1 for a left turn, 0 for three colinear points, and -1 for a right turn. [*5 marks*]

We can test whether the three points `p` → `q` → `r` form a "left turn" by evaluating the *cross product* `(q-p)`×`(r-p)` - the value of this cross-product will be `>0` for a left turn, `0` for a straight line, and `<0` for a right turn.

```
int turnDirection (point_t p, point_t q, point_t r)
```

(d) Complete the loop in `convex.c` to read a series of points from the user through the graphics window, and print each line segment, until **either** the user enters a middle-mouse click (negative co-ordinates), **or** the program detects that the most-recently drawn segment has changed the turn direction. [*10 marks*]

After the input of points ends, the program must output `No longer convex` or `This was convex`.

In the case where the points are terminated by a middle-mouse click (and not by a change in turn direction), you must draw a segment from the final true point to the very first point. Then you must test this final point with two final "turn tests" (with respect to the first point and second point), before determining whether the figure is convex, and printing the appropriate message.

You may assume that the user will always enter at least 3 points.

As well as the functions of (a), (b), and (c), we have also provided the following function in `convex.c` for your use:

```
/* This function takes two directions (in the format returned
 * by turnDirection()) and returns 1 if there has been a change
 * from "left" to "right" or vice versa, and 0 otherwise. */

int dirChanged(int olddir, int newdir) {
  return (olddir*newdir == -1);
}
```

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