Winter 2012	**CPE 101:** Fundamentals of Computer Science I	Alexander Dekhtyar

Lab Exam 2 Sample Problems...

Instructions

- Time to complete the exam: 50 minutes.
- This exam is individual
- This exam is open book
- This exam is closed everything else, including lecture notes, electronic devices, etc
- The only programs you are allowed to have open on the computer during this exam are:
  - The text editor of your choice
  - Terminal window(s)
  - ssh connection to vogon
- Do not use any code you may have access to from earlier in this course or from other courses
- Before starting your work, open a terminal session, and create a directory for the exam. It will be referred to as `labexam01` below.
- The problem description includes all assumptions necessary to answer the problem. Please raise your hand, or approach the instructor if you have any questions.
- Please, collect candy wrappers after you! Additional candy is available, at the instructor's desk: help yourself!
- GOOD LUCK!

Notes

These are **sample** problems. The number of problems on this handout is not indicative of how many problems you will get on the exam.

The specific lab exam assignments will contain instructiosn for testing your code, in ensure that testing can be done expediently. These instructions are omitted from the sample problem specifications.

Sample Problem 1

Create a function `int doubleMin(int *x, int *y)` which

- takes as input pointers to two integer variables;
- doubles the value of the variable that contains the smaller value in it. If both values are equal, none get doubled.
- returns 1 if the value pointed to by `x` was doubled and returns 2 if the value pointed to by `y` was doubled, and returns 0 if neither value was doubled.

For example, if your function is called in the following context:

```
int a, b;
int flag;
a = 5;
b = 6;

flag = doubleMin(&a,&b);
printf("%d, %d, %d\n", flag, a, b);
```

the output should be

```
1, 10, 6
```

(`a` contains the smaller of the two values, it gets doubled, and the function returns 1.)

Put your function in the C file `double.c`.

Sample Problem 2

Write a `C` program `priceCheck.c` that does the following:

- Reads information about the sales of three different houses. For each house, two values are provided. The first value is the price of the house, in dollars. The second value is the size of the house in square feet. All values provided are integer numbers.
- Computes the price per square foot for each house.
- Determines the house with the lowest price per square foot.

- Prints 1 (on a separate line), if the first house has the lowest price per square foot, 2, if the second house has the lowest price per square foot, and 3 if the third house has the lowest price per square foot. In case of a tie between two or more houses, the function prints the smaller of the numbers. The output should end with a line break.

All functionality of your program shall be implemented in the `int main()` function.

A sample input to your program is a file `testPrices01.txt`:

```
100000 1000
100000 2000
100000 1500
```

If your program is compiled into an executable `priceCheck`, then running it by redirecting input from `testPrices01.txt` shall result in the following output:

```
> priceCheck < testPrices01.txt
2
>
```

Sample Problem 3

Write a function `char lhGamePlay()` that plays a *limerick or haiku* game from **Lab 5** according to the following rules:

1. The function takes as input three integer values, which we refer to in what follows as `first`, `second` and `third`.
2. The function output either `'h'` or `'l'` depending on which of the rules of the game is satisfied.
3. The rules stated below shall be checked in the order they are specified. The output shall be generated based on the first rule that is satisfied by the input values.
4. The rules are:
   (a) **Rule 1.** If `second` is the largest number of the three[1] output `'l'`.
   (b) **Rule 2.** If the difference between `first` and `third` is divisible by `second`, output `'h'`.
   (c) **Rule 3.** If `third` is greater than the difference between `first` and `second`, output `'l'`.
   (d) **Rule 4.** Output `'h'`.

This function can be tests using `checkit_char()` macro. The following tests shall all pass:

```
checkit_char(lhGamePlay(4,10,6),'l');    /* rule 1 */
checkit_char(lhGamePlay(21,5,11),'h');   /* rule 2 */
checkit_char(lhGamePlay(12,3,7),'l');    /* rule 3 */
checkit_char(lhGamePlay(10,0,4),'h');    /* rule 4 */
```

[1]This includes any of the cases when `second` is equal to one or more other numbers.

(notice the last test and the value of the `second` argument).

Place your function in the `limerick.c` file.

Sample Problem 4

Write a program `cubicEq.c` which performs (in `int main()`) the following actions:

1. Reads three integer values `a1`, `a2` and `a3` representing three roots of a cubic equation:
$$(x - a1)(x - a2)(x - a3) = 0$$
2. Outputs, one per line, the coefficients for x^3, x^2, x and x^0 for this equation.

A sample input to this program is a file `testCubic01.txt`:

```
3 -1 2
```

When run on this input, your program shall produce the following result:

```
> cubicEq < testCubic01.txt
1
-4
1
6
>
```

$((x - 3)(x + 1)(x - 2) = x^3 - 4x^2 + x + 6)$.

Sample Problem 5

Write a program `move.c` that takes as input the following information:

1. Two integer values, `x` and `y`, representing a location of a player on a two-dimensional, infinite in all directions, grid.
2. Three directional values. Each directional value is a single character, dictating the direction of a move of the player. There are eight possible directions, represented by the following values:

Value	Direction
`Q`	North-West (up and left)
`W`	North (up)
`E`	North-East (up and right)
`A`	West (left)
`D`	East (right)
`Z`	South-West (down and left)
`X`	South (down)
`C`	South-East (down and right)

The coordinate grid is traditional, the X axis values increase left-to-right, the Y axis values increase bottom-to-top.

The program shall output the new position of the player on the 2D grid, after the three moves are completed.

A sample input file `moveTest01.txt` will look as follows:

```
5 4
Q E D
```

The result of running your program on this file as input is:

```
> move < moveTest01.txt
7 5
```

(Nort-West - North-East - East of (5,4) is (7,5))