# C Programming lab
# Batch1 Midterm Exam

Revision Date: September 30, 2013

## Preliminaries

This exam has a total of 100 points. All tasks are all or nothing. There is no partial credit given for this test. The duration of this exam is two hours. You must put all your programs in the correct directories and submit from the specified directory or you will receive zero credit.

Create a directory named *batch1* that hangs off your `/home/IMT2013XXX` directory using 'mkdir' linux command. Create five directories inside the *batch1* directory: *separate*, *quadratic*, *journey*, *vending*, and *filter* directories. You should see the following output:

```
4   ./separate
4   ./quadratic
4   ./journey
4   ./vending
4   ./filter
4   .
24  .
```

The names must be as shown, but the order and the numbers do not matter.

## Task 1: Separate Prime, Odd and Even numbers from a list of numbers(20 points)

Develop a program in *separate.c* in *separate* directory which will take command line arguments consisting of all positive numbers greater than ZERO. The data parsed from the command line arguments are separated by a space. We will pass only positive integer numbers. Your program should identify prime numbers and put it in *prime.dat*, identify odd numbers and put it into *odd.dat* and identify even numbers and put it into *even.dat* file.

Also note that some prime numbers are odd, hence if the number is identified as prime number, that number should be in *prime.dat* file instead of in *odd.dat*. Hence the *odd.dat* should consists only the odd numbers which are not prime. Same with even numbers. Even number which is a prime number should go to *prime.dat* instead of *even.dat*.

If your program is tested as follows

```
./separate 1 2 5 11 13 17 22 100 200 99
```

then your program should have the following data in prime.dat file

```
1 2 5 11 13 17
```

and following data in odd.dat file

```
99
```

and following data in even.dat file

```
22 100 200
```

Remeber to put a space between the numbers in output files as well. Note that your program will be tested with different numbers and different count of numbers.

## Task 2: Quadratic equation (20 points)

Write a single function *compute* in *util.c* file in *quadratic* directory to determine roots of the quadratic equation $ax^2 + bx + c = 0$ in *compute* directory. The roots of a quadratic equation is given by the below formula:

$$root1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

and

$$root2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

The *compute* function will take six arguments as formal parameters. You need to call compute function from other file *eval.c*, where you will write *main* function. The a, b, and c coefficients are supplied as command line arguments. Link and compile both files to produce an *eval* executable. Your program should demonstrate the following output.

```
./eval 1.0 -6 9
root1: 3
root2: 3
```

Another example:

```
./eval 1 3.0 1
root1: -0.38
root2: -2.61
```

Another example:

```
./eval 1 4 5.0
root1: -2+i
root2: -2-i
```

where i represents imaginary number. Note that you have to print the values only in main function in *eval.c* file. No printf's/fprintf's should be placed in *util.c* file. Again note that you have to write a single function in *util.c* file. If more than one function is written, your solution will not be considered for grading. Note that you need to include math.h file if you plan to use *sqrt* function from the math library. Also do not forget to use *-lm* to link the math library while compiling.

## Task 3: Calculation of gallons on your journey (20 points)

Develop a program in *journey.c* file in *journey* directory. Your program should take an input file from the command line argument and should calculate the total gallons consumed on your travel from one place to destination via multiple transport ways. The input file has data in the following format as shown below:

```
<transport-type> <mpg> <miles> <number-of-passengers>
```

One such example file is shown below:

```
taxi 17 70 1
air 53 1000 9
air 72 1500 81
```

In your calculation of total gallons, make sure that if the number of passengers are more than 1, then the gallons consumed will be equally shared by the other passengers as well for that route. The total number of gallons consumed by your travel suggested in above test case is

```
6.47
```

Note that your final answer should have two decimal values. Also your program will be tested with different test files which will have different routes and commutation ways.

## Task 4: Vending machine (20 points)

Write a program considering your code will be ported to embedded controller. Embedded system development represents using minimum number of variables in your code and thus minimizing the memory required to run program. You will have to choose the variable according to the size required. If the number of variables is more than the minimum required, your solution will not be evaluated.

Write a program in a file *vending.c* in *vending* directory. Your program should take one input from the user. The items format in the vending machine is shown below:

```
|water(1-bit)|Cold-drinks(2-bits)|Cookies(2-bits)|Chocolates(3-bits)|
```

The items format suggests that within their individual width format, the user can select the following beverages and chocolates:

```
Water
0: Kinley
1: Bisleri

Cold-drinks
0: Coke
1: Sprite
2: Pepsi

Cookies
0: Bourbon
1: Oreo
2: GoodDay
3: ParleG

Chocolates
0: Nestle
1: FiveStar
2: DairyMilk
3: Eclairs
4: Snickers
5: Hershleys
```

```
6: BarOne
```

The price of the items in the vending machine is supplied via command line argument and is of following format:

```
|water(4-bits)|Cold-drinks(6-bits)|Cookies(6-bits)|Chocolates(9-bits)|
```

Assuming that we have same prices for all items within the same group. i.e Kinley, Bisleri will have same price in group Water. Also Coke, Sprite, and Pepsi will have same price and so are the other items in the same group.

Based on the price command line argument and user inputs on selection of items, your program should print the items chosen, and total price of all items dropped from the vending machine.

For example if the user input is (1 01 10 111) in binary i.e. 183 and price argument is (1110 111110 111110 111111110) in binary i.e 31423998

```
./vending 31423998
```

then your program should print the following:

```
Items selected:
Bisleri
Sprite
GoodDay

Total price:
138
```

## Task 5: Low pass video filter (20 points)

Develop a low pass filter program in file filter.c in *filter* directory. Assuming that just like RRA-audio format, one of your classmate has developed an RRA based video format. One such RRA based video format (in.RRV) is given below.

```
RRVIDEO
samples: 5
createdBy: IMT2013XXX
%%
10
-20
30
-40
50
```

Assuming that the video has noise at higher amplitude sample values and hence given the threshold value, your filter should be able to remove the high absolute amplitude noise levels, retaining all lower sample values less than or equal to the threshold value. For example if the threshold-value is 30, the new filtered video file is as follows:

```
RRVIDEO
samples: 3
```

```
createdBy: IMT2013XXX
%%
10
-20
30
```

Note that the samples tag in the header is changed to reflect the exact samples retained by this filter. Your program should be developed to accept threshold value and input VIDEO file from the command line arguments and print the filtered RRVIDEO file into the standard output console. Your program will be tested in the following format:

```
./filter <threshold-value> <RRV-filename>
```

## Submitting your midterm exam

First, move into your `~/batch1` directory. Then run this system command:

```
du -a
```

and check whether all your files are located in appropriate directory.

Once you are done and ready to submit, stand up and wait for your instructor to come to your place before submission. You need to submit your midterm exam only infront of the instructor. In case if you submit without instructors presence, your submission will be unaccepted. Also sign the attendance sheet with the time of submission before leaving the room. Submit your solutions while in the `~/batch1/` directory with the command:

```
submit clab mr batch1 <your-iiitb.org-email-address>
```

Write your name, email-id and roll numbers in the question paper as well. Give your exam question paper back to your instructor. Have a nice break!!