iMTech C Programming lab, IIITB
Final Exam
Batch 2
(2.5 hours)

Prepared by: M. Rao

Revision Date: December 16, 2013

Preliminaries

This exam has a total of 100 points. Partial credit will be given only if your program fails for one test case and runs for multiple other test cases. Your solution will be evaluated on at least four test cases. You must put all your programs in the specified file names in correct directories and submit from the mentioned directory or you will receive zero credit. In case of arrays (one, two or n-dimensional), you are supposed to use dynamic memory allocation. Use *malloc* to allocate memory. If static memory allocation is used, then your solution will not be graded. Makefile and README files are not required in this exam. Commenting your code is not mandatory. Make sure that your output should be in the format specified. For example, spaces, newlines and decimals should be followed as specified in the test cases.

Create a directory named *final2* that hangs off your `~`(home) directory using 'mkdir' linux command. Create four directories inside the *final2* directory: *recursion*, *partition*, *frequent*, and *game*. Your fifth directory will be created in Task 5. Now, while in the `~/final2` directory, type the system command:

```
du
```

You should see the following output:

```
4   ./recursion
4   ./partition
4   ./frequent
4   ./game
4   .
20  .
```

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

Task 1: Highest value via recursion (25 points)

Move into your `~/final2/recursion` directory. Create a program in a file max.c which will take n real (double) numbers from command line arguments and will find the highest of them. You are supposed to write a recursive function to evaluate the result. You are allowed to define *head* and *tail* function. However determining highest should be done in a single recursive function. Your program on loading as shown below

```
./max 2 4.45
```

should print the following output.

```
highest: 4.45
```

Another example is:

```
./max 0.5 7.99 2.36 4.67 3.45
```

should print the following output.

```
highest: 7.99
```

Note that your program should run for any number of arguments. You can make use of *argc* variable to determine the size of the command line arguments. Note that you need to use recursion. The solution will not be graded, if any loop or any global variable is found in the program. Also note that only two decimal digits are expected in your output.

Task 2: Partitioning raw data (25 points)

Move into your `~/final2/partition` directory. Create a file *partition.c* in *partition* directory. In this task you are supposed to write a program which will partition the raw stereo channel data to mono(1) channel data. Your program should search for the tag "!partition" in the RRA file and separate out the RRA files. For example if the raw stereo channel data is represented as given below:

```
RRAUDIO
createdBy: songlib
samples: 8
modifiedBy: wnoise
%%
10
20
30
!partition
-300
-1000
-3000
-4000
100
```

then after loading your program as shown below:

```
./partition stereo.rra mono1.rra mono2.rra
```

The partitioned data in the separate file is shown below: mono1.rra contains

```
RRAUDIO
createdBy: songlib
samples: 3
modifiedBy: wnoise
%%
10
20
30
```

and mono2.rra contains

```
RRAUDIO
createdBy: songlib
samples: 5
modifiedBy: wnoise
%%
-300
-1000
-3000
-4000
100
```

Note that the sample tag value is changed according to the number of samples stored in the file.

Another example of stereo.rra is shown below:

```
RRAUDIO
createdBy: songlib
samples: 3
modifiedBy: wnoise
%%
10
!partition
20
!partition
30
```

Then after running your program as shown below:

```
./partition stereo.rra mono1.rra mono2.rra
```

the partitioned data in the separate file is shown below: mono1.rra contains

```
RRAUDIO
createdBy: songlib
samples: 1
modifiedBy: wnoise
%%
10
```

mono2.rra contains

```
RRAUDIO
createdBy: songlib
samples: 1
modifiedBy: wnoise
%%
20
```

and mono3.rra is not mentioned in the command line argument and hence the sample value for mono3.rra is ignored. Note that the sample tag value is changed according to the number of samples stored in the file. Note that all the header format remains the same for RRA file.

Task 3: Most frequent word in the file (25 points)

Move into your `~/final2/frequent` directory. Create a file *frequent.c* in *frequent* directory. In this task you are supposed to write a program which will find the most frequent word (string) given in the file. Your program should be case sensitive. For example "Hello" and "hello" are treated as separate words. The file is parsed as command line argument to the program. If you are using *strcpy* or *strlen* function, remember to include the header file *string.h* file.

If the given file *test.dat* has the following contents:

```
Good morning to all of you. all
the best for your exam.
```

```
./frequent test.dat
```

should provide the following output

```
Frequent word: all
```

Task 4: Ghost in pacman game (15 points)

Move into your `~/final2/game` directory. Create a program *ghost.c* in *game* directory. In this program, you are supposed to develop a part of popular game: *Pacman*. Your program is supposed to read the total number of cells (game area), the game is played. The total number of cells, with initial position of ghost and position of fruits is given in *positions.txt* file. The Ghost is designated as G in *positions.txt* file. The ghost when moves to the cell where the fruits are kept, consumes the fruit and thereby changes its direction depending on fruit type as shown in the table below. By default *Ghost* has a unit speed which means that it moves one cell in the directions stated in *actions.txt* file.

designator	fruit type	direction change
B	Bananna	left
M	Mango	right
O	Orange	up
A	Apple	down

The Ghost next movement is provided in *actions.txt* file. Your program *ghost.c* should move based on the actions provided in *actions.txt* file and display the two-dimensional board on updating the positions of the Ghost.

There are four possible movements by the Ghost: *up*, *down*, *left*, and *right*. The actions of the Ghost are given in *actions.txt*. The action: *up* represents moving up the board by one row remaining in the same column. The action: *down* represents moving down the board by one row remaining in the same column. The action: *left* represents moving left to the board by one column remaining in the same row. The action: *right* represents moving right of the board by one column remaining in the same row. If the Ghost is located at the periphery (outer line of the board) then your program should ignore any movements which moves beyond the board. Note that Ghost when consumes the fruit, it moves one more time and then does the next movement according to the *actions.txt* file.

The initial positions of the pacman is given in *positions.txt* file. One such positions is shown below.

```
- - - - - - - - - -
- - - - - - - - - -
- - - B - G - - - -
```

```
- - - B - M - M - -
- - - - - - - - - -
- - - A - - - - - -
- - - - - - - - - -
- - B - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

The *positions.txt* file is passed as first command line argument. Notice a space in each character in *positions.txt* file. Similar format will be used to generate different test files. The *actions.txt* is passed as second command line argument.

One such *actions.txt* file is shown below.

```
down
right
```

This means that Ghost is moving down and then it will check whether any fruits are available, if so it consumes and then updates its movement and then it moves right. After running your program as shown below, your program should produce the following output.

```
./pacman positions.txt actions.txt
- - - - - - - - - -
- - - - - - - - - -
- - - B - - - - - -
- - - B - - - - G -
- - - - - - - - - -
- - - A - - - - - -
- - - - - - - - - -
- - B - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

Another example of *actions.txt* is as follows:

```
left
up
```

The output is

```
./pacman positions.txt actions.txt
- - - - - - - - - -
- - - - G - - - - -
- - - B - - - - - -
- - - B - M - M - -
- - - - - - - - - -
- - - A - - - - - -
- - - - - - - - - -
- - B - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

Another example of *positions.txt* file is shown below:

```
G A O
B M -
- - -
```

and *actions.txt* file is shown below.

```
down
down
```

Your program should throw the following result.

```
- A O
- M -
G - -
```

Task 5: Structures (10 points)

Download route.tgz file from LMS and move the file to your final2 directory. Untar the file using the following commands:

```
tar xvzf route.tgz
```

After untar'ing go inside the route directory and read *route.h* file. In this task, you will have to complete the code. Read the *route.h* file to understand the implementation. You will have to add code only in places of the tag: *ADD CODE HERE* . Complete the code and compile using makefile already given to you. On running the completed and correct code, you will not see any output, rather the program ends. If you see any output, that suggests that you need to work on the code. Test your program thoroughly with different test files. You are supposed to add code only in the tag: *ADD CODE HERE* positions. Any other changes in the code will lead to losing points.

Submitting your final exam

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 final exam only in front 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 `~/final2/` directory with the command:

```
submit clab mr final2 <your-iiitb.org-email-address>
```