iMTech C Programming lab, IIITB
Final Exam
Batch 1
(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. However 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 *final1* that hangs off your `~`(home) directory using 'mkdir' linux command. Create four directories inside the *final1* directory: *recursion*, *partition*, *sort*, and *game*. Your fifth directory will be created in Task 5. Now, while in the `~/final1` directory, type the system command:

```
du
```

You should see the following output:

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

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

Task 1: recursion (25 points)

Move into your `~/final1/recursion` directory. Your task is to print all the elements of 2D array using recursion. Create a program in a file print.c which will take n X m real (double datatype) elements of a 2D array and print these elements using recursion. Note that your recursion function should take 2D array as formal argument. Your code will not be graded if one-dimensional array is used instead. The value of n and m are supplied as command line arguments and n X m double elements are also supplied as command line arguments. The format of the command line argument for your program is shown below:

```
./print <dimension-1> <dimension-2> <element-0> <element-1> <element-2> ..
```

Your program on loading as shown below

```
./print 2 3 1.5 2.5 3.5 1.5 2.5 3.5
```

should print the following output.

```
1.50  2.50  3.50
1.50  2.50  3.50
```

Another example is:

```
./print 3 2 0.5 1.75 1 2 3 6
```

should print the following output.

```
0.50 1.75
1.00 2.00
3.00 6.00
```

It is assumed that your program will be supplied with atleast n X m elements and the value of n and m. Note that you need to use recursion. The solution will not be graded, if any loop or global variable is found in the program. Also only two decimal digits are expected in your output.

Task 2: Partitioning raw data (25 points)

Move into your `~/final1/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 (2) channel data to ordered mono (1) channel data. The format for partitioning is shown below:

```
<Number><space><data>
```

where Number represents filename (mono1.rra or mono2.rra or monoN.rra) parsed in the command line arguments and data represents RRA data to be stored in the separate files. For example if the raw stereo channel data stored in *stereo.rra* is represented as given below:

```
RRAUDIO
createdBy: songlib
samples: 8
modifiedBy: wnoise
%%
1 10
1 20
2 30
1 -300
2 -1000
2 -3000
2 -4000
2 100
```

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
-300
```

and mono2.rra contains

```
RRAUDIO
createdBy: songlib
samples: 5
modifiedBy: wnoise
30
-1000
-3000
-4000
100
```

Another example of stereo.rra is shown below:

```
RRAUDIO
createdBy: songlib
samples: 3
modifiedBy: wnoise
%%
1 10
2 20
3 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 and header format remains same for RRA file.

Task 3: Sorting array of strings (25 points)

Move into your `~/final1/sort` directory. Create a file *sort.c* in *sort* directory. In this task you are supposed to write a program which will sort the array of strings in ascending manner according to its length. The length of the string can be determined using the function *strlen*. For copying strings, you can use *strcpy* built-in function. Remember to include *string.h* header file, if you are using these built-in functions. You can also use man pages of these built-in functions to determine its usage. Different strings are parsed as command line arguments as shown below:

```
./sort hello good morning to all of you
```

should provide the following output

```
to of all you good hello morning
```

Another test case is shown below:

```
./sort C Programming
```

should provide the following output

```
./sort C Programming
```

Task 4: Pacman (15 points)

Move into your `~/final1/game` directory. Create a program *pacman.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 pacman and position of fruits is given in *positions.txt* file. The Pacman is designated as P in *positions.txt* file. The pacman when moves to the cell where the fruits are kept; consumes the fruit and thereby changes its speed as shown in the table below. By default *Pacman* has a unit speed which means that it moves one cell in the directions stated in *actions.txt* file.

designator	fruit type	speed change
A	Apple	2 times the previous
O	Orange	3 times the previous
G	Grapes	4 times the previous
B	Banana	No change
M	Mango	No change

The Pacman next movement is provided in *actions.txt* file. Your program *pacman.c* should move based on the actions provided in *actions.txt* file and update the two-dimensional board with the positions of the *Pacman*.

There are four possible movements by the Pacman: *up*, *down*, *left*, and *right*. The actions of the Pacman 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 Pacman is located at the periphery (outer line of the board) then your program should ignore any movements which moves beyond the board.

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

```
- - - - - - - - - -
- - - - - - - - - -
- - - A - P - - - -
- - - A - A - O - -
- - - - - - - - - -
- - - A - - - - - -
- - - - - - - G - -
- - 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
up
```

Your program on reading *actions.txt* file, moves Pacman down and checks whether any fruits are available, if so it consumes and then updates its speed by X times mentioned in the table above. It then moves right with updated speed. It checks whether any fruits are available and if so again consumes and updates its speed by Y times mentioned in the table above. So the final speed is XY.

After running your program as shown below, your program should produce the following output.

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

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

```
left
up
```

The output is

```
- - - - - - - - - -
- - - - P - - - - -
- - - A - - - - - -
- - - A - A - O - -
- - - - - - - - - -
- - - A - - - - - -
- - - - - - - - G - -
- - B - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

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

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

and *actions.txt* file is shown below.

```
down
down
```

Your program should throw the following result.

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

Note that on the way if *Pacman* sees fruits, it does not have time to consume. This means that *Pacman* consumes only when it stops in a cell and changes its speed.

Task 5: Structures (10 points)

Download air.tgz file from LMS and move to your final1 directory. Untar the file using the following commands:

```
tar xvzf air.tgz
```

After untar'ing go inside the air directory and read *air.h* file. In this task, you will have to complete the code. Read the *air.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 `~/final1/` directory with the command:

```
submit clab mr final1 <your-iiitb.org-email-address>
```