CS2110 Computer Programming Lab

Lab 2: Learning to master Pointers

Release Date: August 18, 2014
Due Date: August 24, 2014 11:55 PM

1 Reading on Pointers

For this assignment, you are expected to read up the tutorial on Pointers, that has been made available on Moodle. If you have any questions, please post them on Moodle for any clarifications.

2 Using pointers

Once you understand the basics of pointers, you can solve these problems that would get you familiar with using pointers and the concepts of memory management. Please follow the submission guidelines given at the end of this section before you make your submission on Moodle.

1. Write a program to take two strings as input and output a string which is the concatenation of alternate characters of two strings. Note that the two strings maybe of unequal length. Use `malloc()` for dynamic memory allocation.

   Input:
   ```
   Length of string1
   Length of string2
   string1
   string2
   ```
   Output:
   ```
   Combined String
   ```

2. **Game of Life**
   Conways game of life is a cellular automaton where the game depends on the initial configuration and nothing else. The initial configuration is a two dimensional grid of cells each of which is either dead or alive. The game proceeds in steps where every cell interacts with the vertical, horizontal or diagonal neighbors and decides on its status in the next step. At each step, the following rules are used:

   - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
   - Any live cell with two or three live neighbors lives on to the next generation.
   - Any live cell with more than three live neighbors dies, as if by overcrowding.
   - Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction. The initial configuration is called the seed and starting from this all the cells take steps in tandem.

   Input:
   ```
   Number of rows (M)
   Number of columns(N)
   Number of steps (K)
   ```

Initial Configuration (Boolean matrix - 1 is alive and 0 is dead)
a_{11} a_{12} .... a_{1N}
.
.
.
a_{M1} a_{M2} .... a_{MN}

Output:

Configuration after K steps:
b_{11} b_{12} .... b_{1N}
.
.
.
b_{M1} b_{M2} .... b_{MN}

3. Create matrices of size r rows and c columns, where both are read as parameters. Take another integer k as input. Fill up an r x c matrix such that the entries in the matrix are integers which are uniformly randomly chosen from 1 to k. It is necessary that the entries are random, there should be no discernible pattern based on the position, value and so on. There are several tests to check if a matrix is sufficiently random. Perform the simplest: take a histogram of the values in each row, each column and the whole matrix. The histograms must be almost same for each of 1 to k. r, c can each be upto 10,000 and the matrix should not be necessarily square.
Generate two matrices as above and perform matrix multiplication. Refer to the internet or use the man pages to find out how one can generate random numbers in C.

Input:
```
No of rows for matrix 1
No of columns for matrix 1
No of rows for matrix 2
No of columns for matrix 2
k
```

Output:
b_{11} b_{12} .... b_{1N}
.
.
.
b_{M1} b_{M2} .... b_{MN}

4. This is a part-reading and part-implementation problem. You are **not** allowed to discuss the answer with your friends. For this problem alone, you are not allowed to post doubts on the Discussion Forums.

Typically in order to resize an existing memory block, one must reallocate a new block, copy the old values to the new block and then free the old memory block.
`void *realloc ( void *ptr, size_t size );`
`realloc()` changes the size of the memory block pointed to by `ptr` to size bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If ptr is NULL, the call is equivalent to `malloc(size);` if size is equal to zero, the call is equivalent to `free(ptr)`. Unless `ptr` is NULL, it must have been returned by an earlier call to `malloc()`, `calloc()` or `realloc()`.

Can you write the realloc function using malloc and free? If yes, please provide an implementation with prototype:
`void *my_realloc ( void *ptr, size_t size );`
If not, justify your answer.

5. **Bonus Problem**: Modify program 3 to perform Matrix multiplication on BigInt that you have written in the previous lab session. You should generate 2 random matrices of Big Integers with the same input format and perform Matrix Multiplication.

Submission Guidelines

The submitted code should be properly indented and commented. Make separate folders for each of the problems above. You should have 4 (or 5) folders. For problem number X, the associated folder should be named `rollNo_lab2_pX` (all in lower case only), where rollNo should be replaced by your roll number and X should be replaced by the problem number. All these folders should be put inside a folder named `rollNo_lab2`. Compress this folder and submit a tar ball that is named `rollNo_lab2.tar.gz`

Please make sure you have removed all the temporary files and executables before you make your submission on Moodle. Each of the folders should have Makefiles. Running make should create the executable named `cs10b037_lab2_pX` where X is the problem number (inside the relevant folder.)

The directory hierarchy should look like the following. Assuming your roll number is CS10B037:
`cs10b037_lab2.tar.gz` [Compressed tar ball]
. `cs10b037_lab2` [Tar ball should contain this folder/directory]
. `cs10b037_lab2_p1` [All the files related to problem 1 should go inside this folder]
. `cs10b037_lab2_p2`
. `cs10b037_lab2_p3`
. `cs10b037_lab2_p4` (This folder should contain a C file if yes is the answer a txt file if no is your answer)
. `cs10b037_lab2_p5`(Optional)