CS262, Lab Assignment 8:

Random Numbers, Permutations and Sorting

Due: Sunday, April 10 at 11:59 pm ET

Description:

In this assignment you will practice how to process arrays using dynamic memory allocation. You will code a C program to create permutations of numbers. Your program will take a random number seed `S`, and an array size `N`, as CLA. It will allocate, initialize, and print an array of `N` numbers in counting order. It will then use a random number generator to create a permutation (random shuffle) and print the shuffled array of numbers to output. Finally, it will use `qsort()` to sort the randomized array in ascending order, and print the sorted array to output.

In this assignment you need to implement the `srand()` and `rand()` functions, dynamic memory allocation with `malloc()`, get information from CLA, and the `qsort()` function.

Generating Random Numbers:

First, you must "seed" the random number generator. Your program will use `argv[1]` as its random number seed. Remember that you will have to convert the `argv[1]` to an actual integer, an alternative is to use `atoi()`. To seed the random generator, use `srand(S)` before generating the permutations. To use `srand()` and `atoi()` you need to include in your code:

```
#include <stdlib.h>
```

Using malloc:

The `malloc()` function is used to allocate memory for your programs when the size of the needed memory is unknown until runtime. It takes a single parameter, an integral value depicting the number of bytes necessary for allocation. Traditionally, this value is determined by multiplying the *size of the particular type* by the *number of elements* needed. Here an example to allocate memory for `N` integers:

```
malloc(sizeof(int)*N)  //sizeof(int) returns how many bytes are needed for storing an int
```

NOTE: For this assignment you can use `calloc()` instead of `malloc()`.
Because of the possibility that there may not be enough free, adjacent memory to handle the requested, every call to `malloc()` should be followed with a check of the return value:

```
int *myArray = malloc(sizeof(int) * N);
if(myArray == NULL){
     printf("Error allocating memory!\n");
     exit(1);
}
```

Also remember that for every call to `malloc()` there must also be a corresponding call to `free()`, once the memory is no longer needed.

The Perfect Shuffle Algorithm:

Since the program needs to generate integer values in the range [0,N-1], to swap positions, the algorithm you will use to generate permutations is the "*Perfect Shuffle*" algorithm.

```
void randperm(int *a, int n)
     for i from n-1 downto 1 do
          j = random integer of [0, …, i]
          swap a[j] and a[i]
```

Using qsort():

The C Library function `qsort()` is an implementation of the Quicksort sorting algorithm. Read the `manpage` to get an idea on how the function works. Attend your Lab session to get assistance in developing an appropriate comparison function for your program.

Instructions:

The source file for this assignment will be named `lab8_<username>_<labsection>.c`

Your program will have two integer values as its command line parameters (`argv[1]`, `argv[2]`):

1. A random number seed (`S`), and
2. The size of the array (`N`) to allocate in your program.

- It will seed the random number generator as described above.
- In `main()` it will allocate a local array `numArray`, that holds `N` values of type `int`.
- Call the `getMemory()` function to get the memory allocation. Make sure the value of `N` ranges in `[2-200]`; otherwise, display an **Err Msg**. and exit the program.
- Your program will print an informational message, stating what the program will do (you could take ideas from the first paragraph of the Description "text in blue").
- The program will then perform the following steps fifteen **(15) times** using a loop:
  1. Call a function that will intialize `numArray` with the values `1...N`
     The prototype for this function is:
     `void InitArray(int *numArray, int arrayLength);`
  2. After returning from the `InitArray()` function, call a function to print the list of `N` numbers. Use a comma `(,)` between each number.
  3. A separate function is then called to create a permutation. It has the following requirements:
     - It has two input parameters: `numArray`, and the **length** of `numArray`.
     - It uses the **Perfect Shuffle Algorithm** to shuffle the values in the array
     - The prototype for this function is
       `void ShuffleArray(int *numArray, int arrayLength);`
  4. After returning from the function described in 3, the shuffled list is printed to the screen.
  5. Sort your array in ascending order using `qsort()`.
     Note that you will also have to make a comparison function.
  6. After the call to `qsort()`, print the array one more time to show the list sorted in ascending order.

Makefile:

Modify the Makefile you used for lab7 so that it works for this assignment.

Submission:

You will submit a typescript file similar to the one of previous Labs:

1. Create a typescript file named `lab8_typescript_<username>_<labsection>`
2. Show that you are logged onto Zeus
3. Show the content of your source code.
4. Compile the code using your **`Makefile`**
5. Show a detailed listing of the directory
6. Run the code given `999` as seed (`S`) and 7 as array size (`N`)
7. Remove the executable using your **`Makefile`**
8. End the typescript
9. Be sure your directory ONLY contains the *source file*, *script* and **`Makefile`**
10. Create a `tarfile` of your `lab8_<username>_<labsection>` directory
11. Submit the `tarfile` to Blackboard
12. Verify that your submitted `tarfile` can be extracted and it's the right `tarfile`.

Congratulations! You have completed your assignment