ENEE 140 Lab 12

Lab instructions

This handout includes instructions for the recitation sessions on Wednesday and Friday. **Follow these instructions** to review strings, then **submit the homework** as indicated below. To prepare for the next lecture, complete the **reading assignment** and try to solve the **weekly challenge**.

1 String reverse

Write a C program that reads in a string of no more than 15 characters (no white space) and then print it out backwards. For example, on input

```
Floatingpoint
```

your program should output

```
tniopgnitaolF
```

Try to implement this both using a for loop and a while loop.

Hint. The length of the input string is unknown, so you need to figure that out first. Remember that every string ends with the end of string character `'\0'`.

Homework

Due: April 29 at 11:59 pm.

Create two programs by following the instructions below. Submit them using the following commands:

```
submit 2016 spring enee 140 AAAA 12 multiplication_table.c
submit 2016 spring enee 140 AAAA 12 multidimensional_array.c
submit 2016 spring enee 140 AAAA 12 permutations.c
```

Note: you must replace `AAAA` with your own section number (0101, 0102, etc.)

1 Multiplication table

Use nested loops to print out the multiplication table as follows, use 2 spaces for the product and leave a tab between the equations:

```
1x1= 1 1x2= 2 ... 1x9= 9
2x1= 2 2x2= 4 ... 2x9=18
...
9x1= 9 9x2=18  ... 9x9=81
```

You can start from the `multiplication_table.c` program from the class public directory on GRACE and modify it so that it prints out the table in the following three formats:

- rotate the table

  ```
  1x1= 1 2x1= 2 ... 9x1= 9
  1x2= 2 2x2= 4 ... 9x2=18
  ...
  1x9= 9 2x9=18  ... 9x9=81
  ```

- only the lower triangle half

  ```
  1x1= 1
  2x1= 2 2x2= 4
  3x1= 3 3x2= 6 3x3= 9
  ...
  9x1= 9 9x2=18  9x3=27 ... 9x9=81
  ```

- only the upper triangle half

2 Multidimensional array

Write a complete program, called `multidimensional_array.c`, that manipulates a 10×10 matrix. Given a 2-dimensional array `M[10][10]` as shown below,

```
 0  0  0  0  0  0  0  0  0  0
-1  1 -1  0  0  0  0  0  0  0
 0 -2  2 -2  0  0  0  0  0  0
```

```
 0  0 -3  3 -3  0  0  0  0  0
 0  0  0 -4  4 -4  0  0  0  0
 0  0  0  0 -5  5 -5  0  0  0
 0  0  0  0  0 -6  6 -6  0  0
 0  0  0  0  0  0 -7  7 -7  0
 0  0  0  0  0  0  0 -8  8 -8
 0  0  0  0  0  0  0  0 -9  9
```

- Use a loop or nested loops to assign each element in `M` these values. Then print the values of `M` out, reserving 3 spaces for each array element.
- Overwrite the last element of each column (that is, the elements in the last row) by the sum of the first nine elements in the same column. Print out the values on the last row, which should be:

  ```
  -1 -1 -2 -3 -4 -5 -6 -7  1 -8
  ```

3 Generate random permutations

Write a complete program, called `permutations.c` to generate a random permutation of numbers from 1 to 10. That is, rearrange the 10 numbers randomly and print out the result. You need to use `rand()` and `srand()`. For example, all the following are valid:

```
 1  3  5  2  10  9  6  7  4  8
 2  8  9  1  10  7  4  5  3  6
10  6  3  8   1  5  7  9  2  4
```

Also keep track how many times you have called `rand()`; you should only call `srand()` once.

Hint. One idea to solve the optional question (but it may use a lot of rand() calls):

Step 1. generate a random number r between 1 and 10
Step 2. check whether r had been used or not
Step 3. if yes, go back to Step 1
Step 4. else

Step 4.1. print out r
Step 4.2. if all 10 number are generated stop
Step 4.3. else go back to Step 1

To implement this simple idea, (1) you will need an array to store the numbers generated for the check in step 2. (2) r needs to be put into the array at step 4.1 (3) You need to update the counter after step 1 to track the times we use rand(). You can improve step 2 by using another array `flag[10]` to track whether number `i+1` has been used or not. Set `flag[i] = 0` initially for all i and update `flag[r-1] = 1` when a new number `r` is generated.

(Challenging): Can you solve this by using only 9 `rand()` calls?

Reading assignment

K&R Chapter 5.11 (read about how to use `qsort`). Re-read chapters 3.5, 3.7.

Weekly challenge

Write a program that sorts an array of integers, so that `a[0] <= a[1] <= a[2] <= … <= a[n-1]`. In other words, implement the `selection_sort()` function so that the output of the program is as follows:

```
56 6 58 20 8 45 56 12 60
6 8 12 20 45 56 56 58 60
```

You can use the following template (also available in the GLUE class public directory, at `public/challenges/week12`):

```
/*
 * selection_sort.c
 *
 *  Sort an array, by gradually building up the sorted array.
 *  At each iteration:
 *  - The beginning part of the array contains the lowest elements,
 *    in sorted order.
 *  - Find the minimum element in the unsorted part of the array.
 *  - Add it to the end of the sorted part.
 */

#include <stdio.h>


void
print_array(int n, int a[])
{
        int i;

        for (i=0; i<n; i++)
                printf("%d ", a[i]);

        printf("\n");
}


void
selection_sort(int n, int a[])
{
}


int
main()
{
        int a[100] = {56, 6, 58, 20, 8, 45, 56, 12, 60};
```

```
        print_array(9, a);

        selection_sort(9, a);

        print_array(9, a);

        return 0;
}
```

The weekly challenge will not be graded. However, if you manage to solve it, you may submit it for extra credit. The deadline for submitting your solution to the weekly challenge is **Monday at 11:59 pm**. To be eligible for extra credit, you must implement correctly **all but two** of the weekly challenges. You can submit your program from a GRACE machine using the following command (replace `AAAA` with your section number):

```
submit 2016 spring enee 140 AAAA 1012 selection_sort.c
```