ENEE 140 Lab 8

Lab instructions

This handout includes instructions for the recitation sessions on Wednesday and Friday. **Follow these instructions** to review strings and arrays, 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 Array review

Go over `array.c` and answer the following questions (you can find `array.c` in the class public directory):

Question 1: Why define 10 as a constant variable SIZE instead of just using the value 10?

Question 2: Why initialize sum with 0 rather than 1 or -1?

The following 3 lines of code read in 10 integers one by one.

```c
for (i=0; i<SIZE; i++) {
    printf("Enter integer number %d: ", i);
    scanf("%d", &a[i]);
}
```

Question 3: How is the sum of the 10 integers calculated?

Question 4: Add the following line after you print out sum, re-compile the code, run it. Observe the output and try to guess/reason why this happens.

```c
printf("overflow: a[SIZE] = %4d\n", a[SIZE]);
```

2 `printf` with string arguments

The `%n.ks` format option in a `printf()` call will reserve n spaces in total but print out only the first k characters from the string. What will happen if the string has less than k characters? What would happen if k>n? Write a simple program to verify your answer.

Homework

Due: April 1 at 11:59 pm.

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

```
submit 2016 spring enee 140 AAAA 8 caesars_code.c
submit 2016 spring enee 140 AAAA 8 primality_testing.c
```

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

1 Caesar's code

Write a program, called `caesars_code.c`, to print your name using Caesar's code. Caesar's code works by substituting each letter with the letter that is 3 positions down the alphabet (for example, 'A' becomes 'D'). When shifting the letters from the end of the alphabet you must wrap around to the beginning (for example, 'z' becomes 'c'). Do not encrypt the space character. For example:

```
Name:           Tudor Dumitras
Encrypted name: Wxgru Gxplwudv
```

Hint. Using a loop, convert each character in the string to the corresponding character in Caesar's code. Pay attention to the difference between uppercase and lowercase characters.

2 Primality testing

Write a complete program, called `primality_testing.c`, which tests whether a given number is prime. An integer is a prime number if it is greater than 1 and can only be divided evenly by 1 and by itself. Write a program to test whether a user provided positive integer `k` is a prime number or not, and, if it isn't, provide a counter-example (a number, other than 1 and `k` that divides `k`). For example, when theuser enters 37, the output should be:

```
37 is prime.
```

When user enters 24, the output should be

```
24 is not prime because it is divisible by 2.
```

Hint. You can use a `while` loop and an `if` statement to solve this question. An `if` statement will be needed to check whether `k` can be evenly divided by an integer. You may also find it helpful to declare a variable to keep track whether a factor of `k` has been found. Initialize this variable to be 0 and update it to 1 when a factor of `k` is found. Then you can check the value of this variable to decide whether `k` is a prime or not.

Reading assignment

K&R Chapters 2.11, 2.12, 3.4, 3.5, 3.6, 3.7, 3.8.

Weekly challenge

Write a program that reads a string and then checks if the string meets the quality rules for UMD Directory passwords.

You can use the following template (also available in the GRACE class public directory, at `public/challenges/week08`):

```c
/*
 * check_password_rules.c
 */

#include <stdio.h>
#include <string.h>

#define MAX_PASSWORD_SIZE 256



/*
Given a string, check whether it meets the quality rules for
UMD Directory passwords

Implement:
   * A password must be at least 8 and no more than 32 characters
     in length.
   * A password must contain at least one uppercase letter.
   * A password must contain at least one lowercase letter.
   * A password must contain at least one character
     from the set of digits or punctuation characters
     (such as # @ $ & among others).
   * A password may not begin or end with the space character.
   * A password may not contain more than two consecutive identical
     characters.

Do not implement
   * A password may not be (or be a variation of) a dictionary word
     in English or many other languages. This includes making simple
     substitutions of digits or punctuation that resemble alphabetic
     characters (such as replacing the letter S in a common word with
     the $ symbol).
   * You may not reuse a password you have already used.
*/
int
check_password_rules(char s[])
{
}
```

```
int
main ()
{
        char password[MAX_PASSWORD_SIZE];

        // Read password and check UMD rules

        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 1008 check_password_rules.c
```