INTRODUCTION TO COMPUTER PROGRAMMING
BERKELEY CITY COLLEGE
FALL 2019
LAB 3

1. **Reverse**
   Write a program that asks for an integer and prints it in reverse order. The program should contain a **recursive** function named *reverse* that takes an unsigned int:

   reverse(537)→735.

   Your function must take the input as an unsigned integer, and you *cannot* at any point convert the integer to a string. Your solution **must** use recursion. Your function should be declared with the following parameters and return type:

   ```
   unsigned int reverse(unsigned int number)
   ```

   You do not have to do the recursion in the reverse function -- you can create a "helper function" in which the recursion actually happens. It is up to you what parameters your helper function takes. One straightforward solution involves creating a helper function that passes in the result as calculated so far:

   ```
   void reverseHelper(unsigned int number, int currentResult)
   ```

   Write a main function to test your solution.

2. **Recursive Binary Search**
   Write a function named binarySearch declared as follows:

   ```
   int binarySearch(int sortedArray[], int value, int start, int end)
   ```

   This function should use recursion to find the index of `value` in the `sortedArray` array. Return -1 if value isn't found.

   Write a main function to test your binarySearch function.

3. **Name sort**
   Write a complete program that allows you to input up to 20 names, sorts these names into alphabetic order, then outputs them. You may use any sorting algorithm you please, but will get extra credit for using more efficient algorithms. Break up your program into relatively short functions that perform each stage of this task.

4. **Get Date**
   Write a function as follows:
   `getDate(int *dayPtr, int *monthPtr, int *yearPtr)`

   that prompts the user to enter the date as a string in the form dd-mm-yyyy (for example, 24-10-2017) and stores these values in the addresses pointed to by dayPtr, monthPtr, and yearPtr.

   Write a main function to test the getDate function.

5. **Resize Array**
   Write a function named resizeArray that accepts an `int` array and size as arguments.

   ```
   int* resizeArray(int array[], int size)
   ```

   The function should create a **new** array this is twice the size of the argument array. The function should copy the contents of the argument array to the new array and initialize the unused elements of the second array with 0. The function should return a pointer to the new array.

   Write a main function to test your function.

6. **String Search**
   Write a program that asks the user for a file name and a string for which to search. The program should search the file for every occurrence of a specified string. When the string is found, the number of the line that contains it should be displayed. After all the occurrences have been located, the program should report the number of times the string appeared in the file.

   You are expected to define functions to perform the various steps in this program.

7. **Ackermann's Function**
   Ackermann's function is a mathematical function often used to test how well a computer performs recursion. Write a function declared as followed:

   ```
   long ackermann(int m, int n)
   ```

   that implements the Ackermann algorithm. Use the following logic:

   If m == 0 then return n + 1
   If n == 0 then return ackermann(m-1, 1)
   Otherwise, return ackermann(m-1, ackermann(m, n-1))

   Write a main function to test your ackermann function. It may be a good idea to test it using relatively small numbers.

8. Simple Encryption

For this question you will implement a (very) simple encryption program. Your file should read the contents of one file, change them as follows, then write the contents into a second file.

Encryption strategy:

* Replace each alphabetic character (uppercase or lowercase) with the letter that comes 13 letters after it in the alphabet. 'a' becomes 'n', 'b' becomes 'o', and so forth. Loop around when you get to the end of the alphabet; for example, the character 'z' should be replaced with the character 'm.' Replace uppercase letters with uppercase letters and lowercase letters with lowercase letters.

* Replace each numerical digit with the digit 5 after it. 0 becomes 5, 1 becomes 6, and so forth. Loop back to 0 when you reach 9. (so, for example, the number 8 should become 3).

* Leave all other characters unchanged.

Write another program to decrypt files encrypted using this method. Alternately, if you can present a good argument for *not* having to write a separate decryption program, give that argument as a comment at the start of your encryption program.