Password Recovery

For this assignment, you will write a program that will attempt to recover a salted and hashed password using a file of candidate passwords.  The file could be interpreted as either a list of easy passwords that a proactive password checker would wish to exclude, or possibly as a list of possible passwords that an attacker would wish to use for matching against a possibly stolen hashed password file. The inputs to the program will be a dictionary file of candidate passwords and a numerical hash value that you will need to examine. You will also know in advance the hash algorithm to use and how passwords are salted before they are hashed.

**Programming Language**

The program must be written in C, C++, or Java, whichever you find more convenient. No other programming or scripting languages are permitted. If you are coding in C or C++, you must use only the standard libraries, such as stdio.h, math.h, and the Standard Template Library. The reason for this is so that the graders will be able to compile and run your program on their machines.

If you are using Java, you must use only the classes and packages included in a standard "SE" edition of Java and the file you submit should not contain a package statement. If you develop your program in an IDE using a package statement, simply comment out the package statement before you test it from the command line and then submit it.

**What You Should Submit**

You should submit a single source code file (.c, .cpp, or .java)  for one of the permitted languages. Multiple submissions are permitted, but only the last submission before the deadline will be graded. Do not worry about the number suffix that Webcourses adds to the second and subsequent submissions -- the graders know how to handle it.  Now, if there are no submissions before the deadline, then only the first submission after the deadline will be graded with the point penalty described in the syllabus.

Your entire program should be contained in exactly one source code file, which should contain all classes, functions, and methods necessary to make your program run. C/C++ programmers should not use separate header files. Java programmers should not use a package statement.  This is so our test scripts can run without changes.

If you submit a C/C++ program, the suggested file name is "recoverpassword.c" or "recoverpassword.cpp". If your program is written in Java, the file (and hence the main class) must be named "RecoverPassword.java".

Your program source file should have a comment header at the top identifying you as the program author. The header should use the following form. If you are teaming, both names should appear in the header.

**Please note:** we will not accept compiled versions of your program, nor will we accept multi-file programs. You must submit exactly one file, which must be a source code file. However, you may submit as many updated versions of your program file as desired, up to the submission deadline.

**Teaming**

You may develop by yourself or in teams of two students currently in the course. Teams of more than two students are not permitted. If you choose to work as a team:

1. Both students must submit the ***same*** file on Webcourses. If you do not personally submit a program file, you will get a grade of zero for the assignment. If you are working as a team and the files submitted are different, then both team members will receive 20-point deductions.
2. The source file that you submit must contain the required course header. There will be a 20-point deduction if your source file does not contain the required course header.
3. The course header in the source file that you submit must name ***both*** team members as the authors. If only one of you submits a header identifying both team members, then whoever forgot to name the other team member will receive a deduction of 20 points.

**Compiling and Running from the Command Line**

Your program must compile and run from the command line because that is how we must test it. If you are unsure what is meant by this, please review the article on this topic in the “Programming Resources” section this Webcourse.

We will compile your program using one the following commands:

* C program: gcc –lm –o RecoverPassword recoverpassword.c
* C++ program: g++ –lm –o RecoverPassword recoverpassword.c
* Java program: javac RecoverPassword.java

Once the program is compiled, we will use a script to test your program against several different combinations of input arguments. Each program test configuration will be launched with command line parameters in the following form:

* C program: RecoverPassword <file\_name> <hash\_value>
* C++ program: RecoverPassword <file\_name> <hash\_value>
* Java program: java RecoverPassword <file\_name> <hash\_value>

Please note: the "<" and ">" brackets in the above command illustrations are for display purposes only. An example of an actual execution command is: "RecoverPassword shortlist 9560204" with no brackets or quotation marks.

**Command Line Arguments**

The program must read in two command line arguments. If you are unsure what is meant by this, or how to use command line arguments, please review the article on this topic in the “Programming Resources” section of this Webcourse.

The programming resources article contains complete programs in C and Java that illustrate how to input and read command line arguments. If you are unfamiliar with using command line arguments, you are strongly advised to key in the appropriate sample program and to make it work on your system before proceeding with the program development for this assignment. Once you have mastered the sample program, you can then proceed to develop a separate program for this assignment. For this assignment, your program will only need two arguments.

Please note: Most IDEs, like Eclipse and NetBeans, require you to configure your program's project to pass a set of command line arguments to the program when it is run. You may wish to use the inputs from the sample outputs included in this assignment for development purposes, so that you can compare the corresponding outputs.

Of course, setting up your IDE in this manner just configures it for just one particular set of command line arguments. Once your program works with this one set of arguments, it is more convenient to copy the source code into a new folder on your desktop where you will be able to use different files and parameters by simply typing them in on the command line and pressing the "Enter" button. Your program may NOT prompt the user to enter the arguments, nor wait for the user to enter the arguments, nor may it assume that they will have any particular names or values.

**The command line arguments for this program are as follows:**

1. The first argument will be full name of the dictionary file to use. If the name, as given, does not contain ".txt", do not add ".txt" to the name. To be sure your program can handle file names with and without the ".txt" file extensions, you can download the sample dictionary files [shortlist](https://webcourses.ucf.edu/courses/1263069/files/63617303/download?wrap=1) and [names.txtPreview the document](https://webcourses.ucf.edu/courses/1263069/files/63617302/download?wrap=1), which are identical except for the names of the files.

2. The second argument will be a numeric string representing the numeric hash value to examine, for example "9560204" (but without any quotation marks).

**Dictionary Files**

All dictionary files of candidate passwords that your program will be tested against will be text files that will contain random 6-character passwords, all upper case, where each character may be any letter of the alphabet from A to Z. There will be one password per line. All passwords will contain exactly 6 upper case letters. The following is an example of what a password file may look like:

The maximum size of the test dictionary will be 1,000 entries.

**Program Operation**

Your program must perform the following operations each time it is executed:

1. The program must retrieve the command arguments and output them in the output header, which must also identify the course and program author(s), as described in the section on required output below.
2. The program must preprocess the dictionary file by reading up all entries, computing their ASCII values (as described below), and then report each candidate password and its ASCII value in a numbered list, as also illustrated in the required output section.
3. Following preprocessing, the program must do the following:
   * initialize a counter to count the number of salt-password combinations examined
   * for each candidate password in the dictionary
     + for each possible salt value
       - increment the counter of combinations
       - prepending the salt value to the ASCII value for the candidate password
       - compute the hash value of the salted password
       - compare the computed hash value to the hash value that was received as the second command argument:
         * if the two values are the same, report the password found, the salt value used, and the current count, as illustrated in the required output section
         * else if the values do not match, loop to test the next salt-password combination
   * if no match was found after testing all combinations, report that the password is not in the dictionary and the total number of combinations examined, as illustrated in the required output section

**Computing the ASCII Value for a Password**

The C, C++, and Java programming languages all read and interpret characters as integer values. The integer values for alphabetic characters are their ASCII values, so no special processing is required. For the upper case letters from A to Z, the ASCII values are in the range from 65 (for A) to 90 (for Z). The program preview lecture and slide set contains sample programs in C and Java that you can use to confirm this.

Therefore, the procedure for computing the ASCII value for a candidate password is to use your programming language to interpret each character of the password as an integer, and simply concatenate the ASCII decimal values for each character to get the ASCII value for the entire password.

For example, consider the candidate password KNZAVM.  The ASCII values for the individual upper case characters are:  75 (for K), 78 (for N), 90 (for Z), 65 (for A), 86 (for V) and 77 (for M).  Using these values, the ASCII value for the password is therefore 757890658677.  Please note that the ASCII value for all candidate passwords will contain exactly 12 decimal digits since the value for each individual upper case letter will be in the range from 65 to 90.

**How Passwords are Salted**

The salt values for this assignment will consist of 3 decimal digits.  Since each digit can be in the range from 0 to 9, there are therefore 1,000 possible salt values (from 000 to 999) for each candidate password.

The salt value is always prepended to the password, that is, it is added at the left of the ASCII value for the password.  For example, if a salt value of, say, 372, is appended to the ASCII password value, the result will be the salted password 372757890658677.  Please note that a salted password will always consist of exactly 15 decimal digits.

**Computing the Hash Value**

For this assignment, we will use a simple linear congruential generator as our hash function.  This is how it should be applied:

1. Split the 15-digit salted password into two parts.  The 7 leftmost digits will comprise the "left" part, and the remaining 8 rightmost digits will comprise the "right" part.
2. Interpret "left" and "right" as long integers in your programming language. For C/C++ programs, you can use the "atol(s)" function to do this, and for Java programs, you can use the Long.parseLong(s) method.
3. Compute the hash value as follows:

Please note that the numbers 243 and 85767489 are fixed numbers and will not change. Feel free to hard code these numbers.

Please also note that the result of the remainder operator ("%") is to produce a value in the range from 0 to 85767488. This should be a sufficiently large range of values that is should be impractical for us to figure out the password and salt value given the hash value.  This is why we need to use a dictionary for this assignment.

**Required Output**

All program output should be to the screen.  The required format is illustrated in the screen shots below.  The sections of the output are:

1. **Output header**section, containing:

* 40 hyphens (used to separate different test runs)
* the course name and author(s)
* echo of command arguments

2. **Dictionary report** section, containing:

* a numbered list of each candidate password in the dictionary and its computed ASCII value

3. **Success report**, if the password was found in the dictionary, containing:

* the password that was recovered
* the ASCII value for the password
* the salt value that was used
* the count of the total number of salt-password combinations that had been examined at the time the password was recovered

4. Or a **Failure report**, if the password was not found in the dictionary, containing:

* the statement, "Password not found in dictionary"
* the total number of salt-password combinations that had been examined in the search

**Program Testing**

You are strongly advised to test your program in the same manner that we will use to grade it. This means: (a) you should be able to compile and run it from the command line within a Command or Terminal window, and (b) you should use command line arguments. A good test will be to use the input arguments shown in the sample outputs above so you can compare your output to what is shown here.

If you are writing your program in C/C++ and you do not currently have the ability to compile and run your program from within a Command Window, you may wish to review the "Programming Resources" article on installing MinGW for Windows, which will give you access to the free gcc and g++ compilers for compiling and running C/C++ programs.