Password Finder

Description

This application tries to show the functionality of the command crypt() used in C. The crypt command is used to encrypt information. It is largely used to encrypt passwords. Systems use this function in order to grant access to user accounts.

As a very simplistic model, a system contains a text file (password file) associating a username and its password in encrypted form. Granting access to a user consist in him/her inserting a string that represent candidate/potential unencrypted password of a username, then the system uses the command crypt(C) to encrypt this potential password and checks if this encrypted string matches the string corresponding the username's encrypted password form in the password file.

This application takes a password file and tries to guess the unencrypted form of all the usernames' passwords using a dictionary (collection of words). In other words, it checks if the encrypted form of any word in the dictionary matches the encrypted form of a certain username's password.

For more information on crypt():
https://en.wikipedia.org/wiki/Crypt_(C)

Format of the Password File.

This application requires a certain format for the password file. Each entry of the file has to be in a different line and this entry represents the username's info. Each entry starts with the username followed by a colon. After the colon, the encrypted password has to be set. At the end of each entry there has to be a colon. Ex:

root:126/NnV2Q0Pw2:
bin:237NQdqSPFYQM:
daemon:34HU/KORaOZa2:
lce:45./Wf9t6iYdY:
lp:56dgRwdjiKZeE:
sync:67tMyo1WVwZKk:
shutdown:78mFDxiM/cKFc:
halt:89fvq4HfGovag:
mail:90JALwKlIRbXw:
news:ab1qRV2ENz2Xw:

Format of the Dictionary.

This application requires a certain format for the dictionary of guesses. Each entry, word, of the dictionary has to be in a separate line. Ex:

rootpass64
devil
superman 12
G00dbye
male
cronkite
transfer1
operator&29
animal
transfer1

Files enclosed for demonstration.

The folder containing this program in the repository, has a two text files. The file passwordFile.txt contains the information about the users. The file dictionary.txt contains the dictionary or collection of words.

User's guide.

In order to execute the program it is necessary to compile and run it through command line. For executing it, it is necessary to input two arguments through command line. The first one is the name of the password file and the second one is the name of the dictionary to be used for cracking the passwords.

Input example of command line:

```
./crackPass passwordFile.txt dictionary.txt
```

Output example:

The program will output the name of the username, password and number of trials/guesses that were executed to find the unencrypted password. If a password was not found, the password label will indicate "NULL" and the trials label will indicate "NOT FOUND". Ex:

```
USER'S INFORMATION.

user:root password:NULL trials:NOT FOUND
user:bin password:NULL trials:NOT FOUND
user:daemon password:NULL trials:NOT FOUND
user:adm password:NULL trials:NOT FOUND
user:lp password:NULL trials:NOT FOUND
user:sync password:NULL trials:NOT FOUND
user:shutdown password:NULL trials:NOT FOUND
user:halt password:NULL trials:NOT FOUND
user:mail password:NULL trials:NOT FOUND
user:news password:NULL trials:NOT FOUND
user:uucp password:NULL trials:NOT FOUND
user:operator password:NULL trials:NOT FOUND
```

Program's Design.

Program's strategy to get passwords.

The strategy to crack the passwords was trying to match a user's hash-value with the hash-value obtained by applying the command crypt(3) on every word of a dictionary using an specific salt. The dictionary used for this purpose, was a list of common passwords used for the log-ins of the social network Facebook.

Important Elements.

The most important element of the program is the struct that is created to store the information of a user. It holds the variables char* username, char* hashValue, char* salt, char* password and int trials. The last variable indicates how many tries took to find a password with a specific dictionary.

Important Methods.

startPosition (char*).
Takes a string and returns the position of the first colon in the string.

endPosition(char*).
Takes a string and returns the position of the second occurrence of a colon in the string.

getHashValue(char*, int, int).
This method extracts a substring of a given string that contains the information that encompasses the hash-value in the password of a user.

getUserName(char*, int).
This method is responsible for extracting the username of a string that represents a user's entry in the password file.

countUsers(char*).
Counts how many entries of users are in the password file. This is useful at the moment of allocating memory to hold the users encompassed in the password file.

createUsers(char*, int).
This method parses through the password file to extract every user and returns an array of structs type user.

countLetters(char*).
Returns how many letters a dictionary's word has.

checkDictionary(char*, user)**
This method is responsible to check every word of the dictionary to find the passwords of the users.

printsUser(user).**
This method is responsible of printing the user's name, password and number of tries for finding the passwords.