COMPSCI 210 Assignment 2

Due date: 21:00 Thursday 29th October

Total marks: 70

This assignment aims to give you some experience on writing simple C programs.

Important Notes

- There are subtle differences between various C compilers. We will use the GNU compiler gcc on login.cs.auckland.ac.nz for marking. Therefore, you **MUST** ensure that your submissions can be compiled and run on login.cs.auckland.ac.nz. Submissions that fail to compile or run on login.cs.auckland.ac.nz will attract **NO** marks.
- Markers will use files and password that are different from the examples given in the specifications when testing your programs.
- The files containing the examples can be downloaded from Canvas and unpacked on server with the command below:
    - `tar xvf A2Examples.tar.gz`
- Given that the assignment is due in the middle of the last week of lectures, there is **NO** possibility to extend the deadline for this assignment.

Academic Honesty

Do **NOT** copy other people's code (this includes the code that you find on the Internet). We will use Stanford's **MOSS** tool to check all submissions. The tool is very "smart". Changing the names of the variables and shuffling the statements around will not fool the tool. In previous years, quite a few students had been caught by the tool; and, they were dealt with according to the university's rules at:

https://www.auckland.ac.nz/en/about/learning-and-teaching/policies-guidelines-and-procedures/academic-integrity-info-for-students.html

Assignment goal

In lectures we discussed implementing a program, `xor.c`, that used the bitwise XOR operation to encrypt and decrypt files. We will use this as the basis of our assignment writing a modified version of the program which we will describe next.

We have provided you with a skeleton file called `A2.c` which you will use to implement this program. You can see that we have included the `stdio.h` and `string.h` header files. Do **NOT** include any other header files. Several functions which you will need to define in order to implement this program have also been declared. Do **NOT** declare or define any other functions. You may define and use macros if you wish.

Part 1 (5 marks)

When the A2 program is run, apart from the name of the program, the program should take two command line arguments:

1. The name of the file to be encrypted/decrypted using the XOR operation and
2. The password for encrypting/decrypting the file.

For example, if the name of the file to be encrypted/decrypted is "auckland.jpg" and the password is "abcd1234", the command below is used to run the program ("$" is the command window prompt):

```
$ ./A2 auckland.jpg abcd1234
```

The result of the encryption/decryption should be stored in a file. The name of the file is the name of the original file with a "new-" prefix. For example, if the name of the original file is "auckland.jpg", the name of the file containing the result of the encryption/decryption operation should be "new-auckland.jpg". You can assume that the name of the file to be encrypted/decrypted consists of at most 15 characters.

In this part of the program you will need to make sure that the program is passed two command line arguments. If not, the following message should be displayed:

```
Usage: ./A2 filename password
```

This must be handled in the `main()` function of the program.

Part 2 (15 marks)

In this part of the program you will need to define two functions:

1. The `make_new_name()` function that takes two pointers to character arrays as its parameters. The first parameter `new_name`, points to the character array that will be used to store the new filename. In other words, the name of the file that will store the result of the encryption/decryption operation. The second parameter `original_name`, points to the character array that contains the characters of the original filename. This function should populate the character array pointed to by `new_name` so that it contains the characters "new-" followed by the original file name as discussed above.
2. The `length_of_password()` function that takes one parameter, `password`, which is a pointer to the character array that stores the password characters. This function should return an integer value indicating the length of the password.

You can call these functions in the `main()` function and then print the values of both the name of the new file and the length of the password in the format shown below. The output of the program is highlighted in red. Please note that "auckland.jpg" is the exact name of the file. That is, the file name does **NOT** have a ".txt" and any other suffix.

```
$ ./A2 auckland.jpg abcd1234
New filename = new-auckland.jpg
Password length = 8
```

Note: Markers will probably use a file with a different name and a different password.

Part 2 (30 marks)

In this part of the assignment, you will need to define the following 3 functions:

1. The `is_alpha()` function that takes a single parameter `c` of type char. The function returns 1 if `c` is an alphabetical character and 0 otherwise. Both lowercase and uppercase alphabetical characters should be considered.
2. The `is_digit()` function that takes a single parameter `c` of type char. The function returns 1 if `c` is a numerical character between 0 and 9 inclusive, and 0 otherwise.
3. The `is_valid_password()` function that takes one parameter, `password`, which is a pointer to the character array that stores the password characters. This function will determine whether a password is valid. If a password is valid, the function will return the value 1. If the password is invalid the function will print out one or more messages indicating what is wrong with the password and then return the value 0. The `is_valid_password()` function will need to use both the `is_alpha()` and `is_digit()` functions to determine a password's validity.

A password is considered valid if:

- It has a length of at least 8 characters. You may assume that the length of the password will be at most 15.
- It has at least one alphabetical character (a character between a-z and A-Z).
- It has at least one digit (a character between 0-9).

Once you complete implementing these functions, you will need to call the `is_valid_password()` function in the `main()` function to evaluate the password passed to the program as the second command line argument. If the password is invalid, your program **must not proceed** with the encryption/decryption process. Here are some examples of running the program once these three functions have been defined. The outputs of the program are highlighted in red.

Example 1 (the password is valid):
```
$ ./A2 auckland.jpg abcd1234
New filename = new-auckland.jpg
Password length = 8
```

Example 2 (the password is invalid):
```
$ ./A2 auckland.jpg abcd
New filename = new-auckland.jpg
Password length = 4
The password needs to have at least 8 characters.
The password needs to contain at least 1 digit.
```

Example 3 (the password is invalid):

```
$ ./A2 auckland.jpg abc123
New filename = new-auckland.jpg
Password length = 6
The password needs to have at least 8 characters.
```

Example 4 (the password is invalid):

```
$ ./A2 auckland.jpg 1234567
New filename = new-auckland.jpg
Password length = 7
The password needs to have at least 8 characters.
The password needs to contain at least 1 alphabetical
character.
```

Note: The markers will probably use a file with a different name and different password.

Part 3 (10 marks)

If the password is valid, the program will proceed to encrypt or decrypt the file with the filename passed to the program as the first command line argument. File encryption/decryption will be handled by the `perform_XOR()` function.

This function takes three pointers to character arrays as parameters. The first parameter, `input_filename`, points to the character array that stores the filename of the file to be encrypted or decrypted. The second parameter, `output_filename`, points to the character array that contains the filename of the file that will store the output of the encryption/decryption process. The third parameter, `password`, points to the character array that contains the password used in the encryption/decryption process.

You can use the code covered in lectures when we discussed the `xor.c` program as the basis for this part of the assignment. The details are as follows:

- You must divide the input file into blocks. The size of each block (except the last block) must be equal to the length of the password.
- Apply the XOR operation to each block and the password as what the program xor.c does.
- Save the results of the XOR operations into the output file.

No additional output is generated by this part of the program as the result of the XOR operations are stored in a file. An example of the program running with this part of the assignment complete are shown below. The outputs of the program are highlighted in red.

```
$  ./A2 auckland.jpg abcd1234
New filename = new-auckland.jpg
Password length = 8
```

A couple of sample files, "sample1" and "sample2", are given to check whether your file has been encrypted correctly. The commands for checking the correctness of your implementation are as below:

Check 1:
```
$ ./part3 auckland.jpg abcde1234
$ cmp new-auckland.jpg sample1
```

If the execution of command "cmp" does not produce any output, it means the two files, "new-auckland.jpg" and "sample1", are identical. This means you have implemented the encryption/decryption correctly. If the execution of command "cmp" shows message like "new-auckland.jpg sample1 differ: char 1, line 1" or something similar, it means your program has **NOT** implemented the encryption/decryption correctly.

Check 2:
```
$ ./part3 auckland.jpg abcde12345
$ cmp new-auckland.jpg sample2
```

If the execution of command "cmp" does not produce any output, it means the two files are identical. This means you have implemented the encryption/decryption correctly. If the execution of command "cmp" shows message like "new-auckland.jpg sample2 differ: char 10, line 1" or something similar, it means your program has NOT implemented the encryption/decryption correctly.

Note: The markers will probably use a file with a different name and a different password.

Part 4 (10 marks)

Complete the `print_first_five()` function that prints out the values of the first 5 bytes in the file that stores the results of the XOR operations. The function takes one parameter, `filename`, which is a pointer to the character array that stores the name of the file. The value of each byte must be shown as a 2-digit hexadecimal number, where each line of output shows the value of one byte. The letter digits "a" to "f" must be shown in lowercase letters. Here are some examples of executing the program. The outputs are highlighted in red.

Example 1:
```
$ ./A2 auckland.jpg abcd1234
New filename = new-auckland.jpg
Password length = 8
9e
ba
9c
84
31
```

Example 2:

```
$ ./A2 auckland.jpg xyz0123456
New filename = new-auckland.jpg
Password length = 10
87
a1
85
d0
31
```

Note: The markers will probably use a file with a different name and different password.

Submission

1. You **MUST** thoroughly test your program on login.cs.auckland.ac.nz before submission. Programs that cannot be compiled or run on login.cs.auckland.ac.nz will **NOT** get any mark.
2. Submit "A2.c" through the Assignment Dropbox at https://adb.auckland.ac.nz/. The markers will only mark your latest submission.
3. **NO** email submission will be accepted.

Resources

- The 4 files, the skeleton file "A2.c", "auckland.jpg", "sample1" and "sample2", that you will need to implement and test your assignment are packed in file "A2Resources.tar.gz" that can be download from Canvas.
- Follow the steps below to extract the files in" A2Resources.tar.gz". Put "A2Resources.tar.gz" in the directory in which your programs are stored.
- Use command "tar xvf A2Resources.tar.gz" to unpack "A2Resources.tar.gz".

Debugging Tips

1. Debugging is a skill that you are expected to acquire. Once you start working, you are paid to write and debug programs. Nobody is going to help you with debugging. So, you should acquire the skill now. **You can only acquire it by practicing.**
2. If you get "segmentation faults" while running a program, the best way to locate the statement that causes the bug is to insert "printf" into your program.
3. If you can see the output of the "printf" statement, it means the bug is caused by a statement that appears somewhere after the "printf" statement. In this case, you should move the "printf" statement forward. Repeat this process until you cannot see the output of the "printf" statement.
4. If you cannot see the output of the "printf" statement, it means the bug is caused by a statement that appears somewhere before the "printf" statement.
5. Combining step 3 and 4, you should be able to identify the statement that causes the "segmentation faults".
6. Once you identify the statement that causes the "segmentation faults", you can analyse the cause of bug, e.g. whether the variables have the expected values.

Hints

1. Do not use global variables.
2. Macros can be useful.
3. All the C programming techniques required to do this assignment can be found in the "Basic C Programming" lecture slides. You are welcome to use techniques not covered in lectures if they do not require you to include another header file. If you do decide to use a technique not covered in lectures, it will be up to you to figure out how to use it properly.
4. Think about how you will use the arguments passed to a function and what local variables you will need to declare for the function to use.
5. Think about how you will use the functions you define in the `main()` function.
6. Don’t wait until you have defined all the functions required for the assignment before you begin debugging. It is best to thoroughly debug your code every time you define a function.