School of Computer Science, McGill University

# COMP-206 Introduction to Software Systems, Winter 2021

## Mini Assignment C: Intro to C Programming

Due Date March 17th, 18:00 EST

**This is an individual assignment. You need to solve these questions on your own. If you have questions, post them on Piazza, but do not post major parts of the assignment code. Though small parts of code are acceptable, we do not want you sharing your solutions (or large parts of them) on Piazza. If your question cannot be answered without sharing significant amounts of code, please make a private question on Piazza or utilize TA/Instructors office hours. Late penalty is -15% per day. Even if you are late only by a few minutes it will be rounded up to a day. Maximum of 2 late days are allowed.**

**You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment.** You must not use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can access `mimi.cs.mcgill.ca` from your personal computer using **ssh** or **putty** and also transfer files to your computer using **filezilla** or **scp** as seen in class and in Lab A and mini assignment 1.

For this assignment, you will have to turn in one C program source code file and one shell script. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. No points are awarded for commands that do not compile/execute at all. (Commands that execute, but provide incorrect behavior/output will be given partial marks.) All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.

**Please read through the entire assignment before you start working on it. You can loose several points for not following the instructions.** There are also some helpful hints given at the end of this document that can be useful to you.

Lab G provides some background help for this mini assignment.

**Total Points: 20**

### Ex. 1 — Implementing A Simple Text Cipher (14 Points)

In this assignment, we will implement a simple text cipher, that accepts an integer key and a message and encrypts the message. We will not pursue the decryption aspect in this assignment, but your are free to explore that on your own.

The cipher's encrypting logic works in the following way. Let us say that you want to encrypt the message `THISISASECRETMESSAGE` using a key 5

The logic involves first distributing the characters in the message in a zig-zag manner across 5 (the key) "levels".

The dots (.) are merely place holders to help you visualize this better.

```
T . . . . . . . . E . . . . . . . . S . . .
. H . . . . . . S . C . . . . . . S . A . .
. . I . . . A . . . R . . . E . . . G . .
. . . S . S . . . . . E . M . . . . . E
. . . . I . . . . . . . T . . . . . . . .
```

Then, the encryption basically involves sequentially producing the characters from each level, starting from the top to the bottom, to the output. For the above example, the encryption produced will be `TESHSCSAIAREGSSEMEIT`. Where `TES` is from the first level, `HSCSA` is from the second level, and so forth.

Clearly, if we change the key, it will change the encryption. For example, with a key of 3, the encryption logic makes the following arrangement.

```
T . . . I . . . E . . . T . . . S . . .
. H . S . S . S . C . E . M . S . A . E
. . I . . . A . . . R . . . E . . . G .
```

Resulting in an output of `TIETSHSSSCEMSAEIAREG`.

1. Use `vim` to create a C program source file `cipher.c` . All of your source code logic should be contained in this file. Further, all of your logic should be in a single function, `main`. **Please read through the entire assignment before you start working, as the programming constructs that you are allowed to use are limited.**
2. **(1 Point)** The source code file should include a comment section at the beginning that describes its purpose (couple of lines), the author (your name), your department, a small "history" section indicating what changes you did on what date.

   ```
   /*
    Program to implement a text cipher
    ************************************************************
    * Author          Dept.                Date          Notes
    ************************************************************
    * Joseph D        Comp. Science. Mar 10 2021     Initial version.
    * Joseph D        Comp. Science. Mar 11 2021     Added error handling.
   */
   ```

   The code should be properly indented for readability as well as contain any additional comments required to understand the program logic.
3. The source code should be compilable by (exactly) using the command `gcc -o cipher cipher.c`. If not, TAs will not grade it and no points will be awarded for the assignment.
4. **(1 Point)** Compilation should not produce any warnings!
5. The program accepts 3 arguments, the first is either `-e or -d` to indicate whether the user wants to encrypt or decrypt (we are only doing encryption). The second argument is a key, which must be an integer that is $\geq 2$ but $\leq 10$. The third argument is the message to be encrypted which can be of any length (including 0 which is represented by an empty string argument, `""`). You can however, assume that the message argument will contain only non-space, uppercase alphabets, such as the example given before.
6. **(2 Points)** If the program is not passed 3 arguments, it should print a usage message and terminate with code 1.

   ```
   $ ./cipher
   Usage : ./cipher [-e|-d] <key> <MESSAGE>
   $
   echo $?
   1
   $
   ```

7. **(4 Points)** The program should print error messages and terminate with error code 1 if any of the first 2 arguments are not valid options, as shown below

   ```
   $ ./cipher -k 5 HELLOTHERE
   Error -k is not a valid option
   $
   $ echo $?
   1
   $
   ```

   If the first argument is good, the same goes if the user enters a non-valid key as the second argument.

   ```
   $ ./cipher -e 4.r HELLOTHERE
   ```

```
Error 4.r is not a valid key
$ echo $?
1
$
```

8. **(6 Points)**
   For valid arguments, the program should produce the encrypted message in the output (see example below) and terminate with 0 code.

```
$ ./cipher -e 3 THISISASECRETMESSAGE
TIETSHSSSCEMSAEIAREG
$ echo $?
0
$
```

   Make sure that the program prints a newline character, following the encrypted output, so that your shell prompt starts in the next line (As seen in the above example. This applies even if the message is empty).

```
$ ./cipher -e 3 ""

$ echo $?
0
$
```

9. If the program is invoked with `-d option`, as long as other arguments are valid, just terminate with 0 without producing anything in the output.

```
$ ./cipher -d 3 TIETSHSSSCEMSAEIAREG
$ echo $?
0
$
```

10. Your entire C program must be contained in the `main`. You are not allowed to write any functions of your own. Further, the only library functions that you are allowed to use are `printf and putchar`. **-3 points** deduction if any of this is not followed.
11. You are not allowed to create/define any new arrays. You may (and have to) use the command line argument arrays given to main (argv, etc.), and can have additional variables referencing these arrays as needed to make your program logic simpler. However, you must not modify the contents of any of these arrays either directly or through other variables referencing them. **-3 points** deduction if any of this is not followed.
12. If your program crashes for ANY test case case, it will result in a **-3 point** deduction.
13. If your program is stuck (infinite loop) and the TA has to terminate forcefully, it may not get tested for remaining test cases in TA's tester script.

## Ex. 2 — A Test Script (6 Points)

1. Write a simple tester script for your cipher program. Name this script, `testcipher.sh`. Your script need not be fancy. It just needs to functional. In other words, a person who is executing the script should be able to observe the input and output of the individual program invocations (including the termination codes of the process) for each test case and come to the conclusion as to whether the program passes the test case or not. The test script need not inform the observe whether the test case is passed or not, you can assume that the observer knows what a valid outcome should look like for each input scenario and make that judgement for themselves.

   The script will be invoked as below:

```
$ ./testcipher.sh
```

Once the user starts the script, there is no further interaction with the script. (The script just goes and executes a bunch of test cases you have put inside it). The output thus produced by the script should be send to standard output and/or standard error as you may chose.

2. **(2 Points)** The tester script should start by checking if there is a `cipher.c` in the current directory, if so, then try to compile it and make an executable (using the `gcc` command as mentioned previously). After that it should check if the executable, `cipher` was built. If any of these steps does not work out, inform the user of the appropriate situation and exit the test script without attempting to execute any more test cases.
3. **(4 Points)** At a minimum, your tester script should include the test cases explicitly listed as examples in Ex. 1. You are encouraged to include a few more to ensure that you catch any other potential bugs and make it easy for you to test your logic.
4. Every other detail is left to you as to how you want to implement the tester script. Remember, you can get points for your tester script even if your C program does not work correctly for all the test cases.

- You must write a reasonable amount of comments (to understand the logic) in your tester script. **-1 point** for not writing sufficient comments.
- As stated in the description, the script output should be understandable to the user (input, output, return code, etc.) - think how the mini tester in the past assignment was useful for you to judge if your program had a bug by looking at its output. Lack of this readability in the script output can result in **-2 points** deduction.

## WHAT TO HAND IN

Upload your C program `cipher.c`, and script, `testcipher.sh` to MyCourses. If MyCourses is not allowing files with `.sh` extension, you may use the `tar` command as discussed in class to make a `tar` file or a `tar.gz` file that contains your C program and shell script and submit them. Please download your submitted tar files to double check if they are good. Erroneous invocation of the tar command can sometimes result in a bad file. Such submissions cannot be unfortunately graded.

DO NOT turn in the executable `cipher`. TAs will compile your C program on their own as indicated in the problem descriptions above.

## ASSUMPTIONS

- The first and second arguments that might be passed to `cipher` can be anything, your program should not crash, but should take appropriate actions.
- You can assume that the third arugment is always made up of ONLY uppercase alphabets (no spaces) and can be of any length (including 0).

## HINTS

This is a high-level outline to get you started with the logic. You are not obliged to follow it.

- Observe the example given at the beginning of Ex.1. We can see that the message has to be visualized into as many levels as the value of the `key` is.
- Now look at the space (dots) between consecutive characters in each level. They follow a certain "step-size" between consecutive characters. This step size is different depending on whether the zig-zag is going "up" or "down". See if you can formulate an equation for these two step-sizes based on the key and the level. (Try to work it out with a pen and paper first).
- Using the above two concepts, you can iterate through the input message as many times as there are levels and for each level, print ONLY the characters that fall into the step size(s) for that level.
- Understanding the ASCII table will help you come up with a logic to convert the key argument (which is a character string/array) into an integer variable. For example, `int digit = '6'-'0'` will store the value `6` into the integer variable `digit`.
- The concept of computing the length of a character string/array is not very hard, you just count characters till you encounter a `'\0'` character.

## RESTRICTIONS

For your shell script, you may use any bash features, Unix commands, options provided already in `mimi`.

Please follow the instructions given for C program at the end of Ex.1, on the features and functionality that your are allowed to use.

## TESTING

There is no mini tester associated with this assignment. You are encouraged to use your tester script to test your program's behaviour. Start with very short messages so that you can easily look at the output to see if the logic is working before getting into longer messages.

TAs will test using their own scripts, but it will follow the assumptions that have been setforth in the assignment description.

## QUESTIONS?

Please use piazza. However, before posting your question, use the search functionality to check if it has been already discussed. You should also look under "Mini 4 general clarifications" pinned post to check if a popular question has been already included there. They will not get individual responses again.

TAs will not help you with logical errors, you are already expected to have some idea of developing basic programming logic and debugging them from your introductory programming courses like COMP 202. So do not ask questions of the sort "My code is not printing the correct output" - that is because your logic is not correct. Use `printf` statements to print the values of loop indexes, etc., as you are debugging the code to verify if your programming logic is correct. The core of the programming logic is already given to you in the HINT section which you should be able to develop into the full code.

TA help is limited to clarifying assignment requirements and helping with C programming language syntax and semantics related questions.