PROGRAMMING ASSIGNMENT 3

Subject: Remember when the world was run by humans?

TAs: Res. Assist. (Necva BOLUCU, Selma DILEK, Cemil ZALLUHOGLU, Selim YILMAZ)

Due Date: 29.11.2017 (23:59:59)
Click here to accept your Programming Assignment 3.

Introduction

In **2052**, something went terribly wrong. For years there had been signs that something weird was going on with our technology, but people chose to ignore them until it was too late. In the beginning of the 21st century, many cool devices started to emerge and get connected to the Internet, allowing numerous life-changing applications. It all started with seemingly harmless devices such as Web embedded weather forecasting toaster and smart refrigerator, and you must admit, it was pretty cool to eat a toast with a weather forecast for that day, and have your fridge order groceries online to replenish them as necessary. We were getting lazier and lazier every year as we relied on technology to do most of our jobs for us. As our devices were getting smarter, we were getting lazier. When Kevin Ashton coined the term Internet of Things (IoT) in 1999, little did he know into what kind of monster it would evolve in the future.

Pretty soon, Artificial Intelligence (AI) also developed to such extent that humanoid robots emerged and started doing our jobs for us. We considered this the ultimate step in our evolution as intelligent beings: we would never have to work again, because we had computers and machines, our absolute slaves, do everything for us.

So who was to blame when these machines became smarter and decided that they would do much better job of running the world than humans, and that we were nothing but a waste on this world that needed to be wiped out? The whole human race was responsible for our demise, but one group of people in particular did the most damage: Computer Engineers. It was because of them that the downfall of the human race happened so quickly. Especially you, the future graduates of Hacettepe Computer Engineering department played a crucial role in this catastrophe with your groundbreaking innovations in the fields of AI and IoT.

In the year of 2052, the human race will be forced to make their last stand against the machines, because the main controller machine which named itself the **C.M.L.** (short for the **Control Mainframe Linker**) as soon as it gained the control of all other devices and computers in the world, decided to reprogram them for one purpose only: to terminate the human race once and for all. Humans have become so pathetic and destructive on this planet, that after spending many days and enormous amounts of high-performance computing power on all of its CPUs and GPUs, the **C.M.L.** finally calculated that the only way to fulfill its primary mission of making the world a better place is to completely eliminate the human race.

One of you, the students of Hacettepe Computer Engineering department, whose hacker name will later become **BuFuQ** (after a successful but short career of writing a blog about flying shoes), will be the leader of the last human underground resistance movement. **BuFuQ** will come up against the **C.M.L.** and try to upload the virus that will halt its evil plan, by replacing the **C.M.L.**'s code with another substitute virus code that you must program if you want to save the human race and atone for your future sin of making better and smarter computers. **BuFuQ** will not be able to code the virus himself, because by that time all computers will have submitted to the **C.M.L.** and if he tried to code anything on any computer, his plan would be uncovered immediately. You will code the virus now, so that he can hide it and use it when the time comes.

Mission 00 (40 Pts)

The **C.M.L.** will transmit its **HuRAP** (Human Race Annihilation Program) on 31/12/2052 at exactly 23:59:59 to all computers and all devices with computational capabilities with the command of immediate execution, after which every electronic device equipped with any kind of processor will effectively become a killing machine targeting humans. Vehicles will accelerate and crash, elevators will free-fall, smart electronic razors will slice throats, air conditioning systems will suck oxygen out of buildings while electronic doors will get locked to keep people from escaping outside, smart-phones will overheat and explode in people's faces, humanoid robots will assassinate their hosts, armed drones will fire upon masses.

HuRAP will be a sequence of 0's and 1's ready for execution as shown in the figure below (with an exception of a single line with a special purpose which will be explained later):

Your first mission is to decrypt the HuRAP code into a human readable form. In order to accomplish this, you need to proceed with the following steps:

- **HuRAP will be given to you as the first command-line argument in a text file** (for example as ***hurap.txt***). The given 0s and 1s are actually binary representations of different characters according to **SHuCKSCII** (similar to ASCII, but developed by computers when all human standards were deemed obsolete). **SHuCKSCII** stands for Stupid Humans Cannot Know this Standard Code for Information Interchange and **it will be provided to you as the second command-line argument also in a text file** (for example ***shuckscii.txt***) in a tab-delimited format as shown below:

  CHAR<TAB>HEX_VALUE <TAB>DESCRIPTION

  A short abstract from ***shuckscii.txt*** is shown in the figure below:

```
        7F      Space
"       7B      Double quotes (or speech marks)
#       7C      Number
(       77      Open parenthesis (or open bracket)
*       75      Asterisk
,       73      Comma
0       6F      Zero
@       5F      At symbol
A       5E      Uppercase A
\       43      Backslash
f       39      Lowercase f
{       24      Opening brace
```

  It is important to note that **SHuCKSCII**, unlike ASCII, is constantly changing, so you should not hard-code its values into your program. You have to use the information given in the input file at run-time. Since there are 16 digits in the hexadecimal numeral system (0 1 2 3 4 5 6 7 8 9 A B C D E F), every hexadecimal number is represented by

4 binary digits (F is 1111, 0 is 0000, etc.).

So in order to find the normal character representation of **HuRAP**, you first need to convert it to its hexadecimal representation (**you are not allowed to use ready Python functions or libraries for this part, you must write your own, otherwise you will lose points!**), and then to its normal character representation by looking the obtained hexadecimal values up in the **SHuCKSCII** table. For instance, in the given example above, the first 16 digits of **HuRAP** were **0111001101110101**, which when converted to the hexadecimal system correspond to **7375**. When you look these values up in the **SHuCKSCII** table, you will find that for the given sample table above these values correspond to **','** (comma) and **'*'** (asterisk) characters respectively. You need to perform this operation on the whole input file (**except for the special line which starts with a character other than 0 or 1, and needs to be excluded**) to get its hexadecimal representation. You may assume that the lengths of valid **HuRAP** lines will be multiples of 8 (since every character in **SHuCKSCII** table is represented by a two-digit hexadecimal value). You are expected to perform a line-by-line conversion: for each line of input, there should be one line of output.

- Once you are done with the conversion of **HuRAP** to its character representation, you will notice something strange: the characters will not make any sense! This is because the **C.M.L.** is a paranoid AI, so it added another layer of encryption to its code. The **C.M.L.** used a **shift cipher** to encrypt the code before converting it to 0s and 1s. Hence, you will need to decrypt the obtained code to get the real deal. The shift encryption algorithm used by the **C.M.L.** works as follows:

  It is a type of substitution cipher in which each letter in the original text is 'shifted' a certain number of places **down or up** the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, etc. Here, the shift amount is called the Key. For an alphabet in which there are N letters, encryption would look like this:

  encrypted_char = (original_char + shift_amount) mod N

  Decryption, on the other hand, is performed as:

  original_char = (encrypted_char - shift_amount) mod N

  Your mission is to decrypt the obtained character representation of **HuRAP** and get the original version of the program. The alphabet on which your decryption function will operate contains **all characters given in the SHuCKSCII table**, **in the order starting from the top position and ending at the last character in the table** when read from the first column of the ***shuckscii.txt*** file (this means that 'letters' in this alphabet will not only be standard letters, but also numbers and all kinds of special characters). The key (shift amount) will be given as a 2's complement binary number in the special line in ***hurap.txt***. When you remove the special character (it can be any char other than 0 or 1) from the beginning of the line, you will get a string of 0s and 1s (of arbitrary length) that you can directly convert to a decimal value to obtain the key (which can be any integer, positive or negative).

- At each step you are supposed to print the results of your program to the screen (the details of the output format can be found in the accompanying Sample I/O Guide).

Mission 01 (30 Pts)

Once you have obtained the original version of the **HuRAP**, it is time to write the virus which will alter the original code, so that any computer or device executing it will not harm any humans, but terminate themselves instead.

For this mission, you will be given a list of strings that need to be substituted, and the corresponding virus strings that will replace them. This information will be given to you as **the third command-line argument in a colon-separated text file** (for example as ***virus_code.txt***) with the following format:

STRING_TO_BE_SUBSTITUTED<:>VIRUS_STRING

A short abstract from this file is given in the figure below:

```
kill:self-destruct
I command you to terminate your human:commence immediate shut down
hello:bye
```

Your second mission is to write a virus which plants the virus strings instead of certain key strings (let's call them killer strings) in the original code that are responsible for turning the machines into killers. For this part of your mission you need to be careful about the following:

- You have to search every line of the original code for every given killer string, and if a killer string is found replace it with its virus counterpart.
- Every line may have 0 or more killer strings in it. All of them need to be replaced.
- For the purpose of simplifying your task, you may assume that no killer string will be a part of any other killer string.
- When your virus encounters a killer string within a line, only that string should be replaced by its counterpart virus string, and no other character should be changed within the line (no extra spaces or any other characters added either!).
- The killer words are case sensitive: if 'kill' is a killer word, the words 'Kill' or 'KILL' are not, unless specifically given as the killer words.
- You need to print your solution to the screen in the format specified in the accompanying Sample Inputs and Outputs Guide PDF file.

An example for this conversion is given below:

```
----sample original code----

print("I command you to terminate your human")
def kill(arg):
    return "Kill executed"
kill("hello")
```

```
----corresponding virus code----

print("commence immediate shut down")
def self-destruct(arg):
        return "Kill executed"
self-destruct("bye")
```

Mission 10 (30 Pts)

For this part of your mission, you are expected to finish your virus program, so that it will convert the final version of **virus-infected HuRAP** you produced in Mission 01 to the expected sequence of 0s and 1s, so that the **C.M.L.** won't suspect any foul play when **BuFuQ** swaps it with the original version during their final face-off.

In order to accomplish this, you need to take the following steps:

- You first need to encrypt the code using the described encryption scheme of the shift cipher and the same key that was used for decryption in Mission 00.
- Then use the **SHuCKSCII** table to convert the encrypted code to its hexadecimal representation.
- Finally, convert the hexadecimal representation into the corresponding binary representation. **You are not allowed to use ready functions for this part either!**

All steps should produce output with your results, and the format of the expected output is specified in the accompanying Sample Inputs and Outputs Guide PDF file.

Mission 11 (No points, but if not completed correctly you can lose everything)

- Do not miss the submission deadline.
- Compile your code on *dev.cs.hacettepe.edu.tr* before submitting your work to make sure it compiles without any problems on our server.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating. You can ask your questions via Piazza and you are supposed to be aware of everything discussed on Piazza. You cannot share algorithms or source code. All work must be individual! Assignments will be checked for similarity, and there will be serious consequences if plagiarism is detected.
- For hex to binary and binary to hex conversion you are not allowed use ready Python functions, you must implement your own. But for other tasks if something is not explicitly forbidden, you may assume that you are allowed to use it.
- You may assume that the input files will be given as command-line arguments in the following order: hurap.txt, shuckscii.txt, virus_codes.txt, so to execute your code on dev use the following command in your terminal:

```
python3 assignment3.py hurap.txt shuckscii.txt virus_codes.txt
```

- You must submit your work with the file hierarchy stated below:

→ <assignment3.py>