CIPHER TECH SOLUTIONS

Cipher Tech Solutions, Inc.
407 North Highland Ave
Upper Nyack, NY 10960
www.CipherTechSolutions.com
Phone: 888-948-8324
Fax: 845-230-6632

Applicants,

Cipher Tech uses various technical challenges to assess the skills and aptitude of potential hires seeking software development positions within our company. You must successfully complete all of the challenges to be considered for the next round of interviews.

We understand these challenges may require varying levels of research. If you choose to use a code snippet or module directly from a different author, you are required to give credit to that author. This includes code segments and/or modules directly provided by professors, peers, and internet sources. These challenges are designed to test your technical knowledge and ability, so please ensure that the solutions you submit reflect your ability.

These challenges are language agnostic, so feel free to submit your code in any language you choose.

We look forward to reviewing your responses!

- The Cipher Tech Technical Recruiting Team

Challenge 1

A common technique for obfuscating data is to use exclusive-or (xor) and some key; it is inexpensive and symmetric. A problem occurs when this is used on file formats like portable executable where there are lots of nulls since xor'ing nulls with your key ends up writing your key out.

A slightly more complex algorithm is to implement a linear congruence generator (LCG) to generate a key stream which will be xor'ed with the data. The generic form of an LCG is:

```
Value = (M * Value + A) % n
```

where M is the multiplicative constant, A is the additive constant, and n is the modulus. These three values are the parameters of the LCG.

For this problem you'll be using an 8-bit LCG (i.e. n = 256), with M = 0xa5 and A = 0xc9. The LCG is initialized with a value and stepped to produce the key stream. The next byte of key is read from the LCG each step.

For example, if the initial value of the LCG is 0x02, then the next value, which would be the first key byte, would be:

`(0xa5 * 0x02 + 0xc9) % 256 = 0x13.`

Your task is to implement this algorithm. We're only interested in algorithm implementation here; other code will be discarded.

The function should adhere to the following signature:

```
unsigned char *LCG(unsigned char *data, int dataLength, unsigned char initialValue)
```

Example Tests:

data	“apple”
dataLength	5
initialValue	0x55
result	“\xF3\x93\x68\x2D\xCB”
data	“\xF3\x93\x68\x2D\xCB”
dataLength	5
initialValue	0x55
result	“apple”

Challenge 2

Solution design is an extremely important aspect of software engineering. Given a task, can you organize your solution in a comprehensible, unambiguous, and efficient manner such that development can progress quickly and that non-developer personnel can understand your solution? Your proposed design should include **what** the design is and **why** you chose to do it that way.

Cipher Tech has recently come into possession of a large amount of reports in PDF format. The reports contain data that will be used to populate a SQL database for a demo of a new tool. Unfortunately, manually extracting all of the required data points would either take too many people or too much time, and of course doing manual extraction on this set of reports wouldn't help with future sets. Therefore, you have been asked to design a tool to automate this task. To aid you with this, the PDFs will be first run through a converter which converts PDF documents to HTML format. The database will be constructed at a later time, so you won't need to handle that step.

Keep in mind that while this is sure to be a fun challenge to actually implement, all submitted code will be disregarded. You may use whatever format(s) you desire to effectively and efficiently convey your design; however your response **must** include a text portion explaining your design. Included is an example HTML document (Challenge2.htm) made for this challenge. You'll notice that it doesn't include many of the data points you'll be responsible for, so be sure to consider your solution carefully.

The reports will contain text from which we want to extract various data points. Your program must be able to initially extract the following data point formats: IP addresses, proper names, 10 digit phone numbers, email addresses, and URLs. You will also be responsible for coming up with the output format, though it is recommended that you use a common/standard format.

You will need to be able to identify data points in basic text, as well as tables, like the one below.

Name	**Address**	**Phone**
Cipher Tech MD	6810 Deerpath Rd., Suite 415, Elkridge, MD 21075	888-948-8324
Cipher Tech Boston	20 Park Plaza, Suite 528, Boston, MA 02116	888-948-8324

In an effort to reduce ongoing dependence on developers, your program must be able create, update, and delete data point formats. For example, if a new set of reports contained times like "9:45 PM", your program would need to be able add this new data point format. You will be responsible for deciding how this will work. Similarly, if a data point's detection turns out to be suboptimal, your program needs to support updating it. For example, adding the ability to detect times of the format "**PM** 9:45". Finally, if a particular data point is no longer needed, your program should support removing it. For each of these three operations, your program should not require recompilation (requiring a restart is acceptable).

Challenge 3

An image's metadata is a goldmine of information. It can contain data about the model of the camera that took the picture or the lighting details when the picture was taken. Over the past few years, privacy concerns were raised with sites ranging from Facebook to 4chan after images that included GPS data were uploaded and users were subsequently tracked in the real world. This challenge will involve traversing such a structure to extract GPS data.

You are given a block of EXIF data (challenge3.bin) from an image and asked to determine where the image was taken. We expect our developers to be able to handle a wide variety of formats, including the obsolete, the esoteric, and the customized, and we want to see that you can follow a specification. As such, the use of EXIF specific libraries is prohibited for this challenge.

To complete this challenge, your code should:

- Parse the relevant TIFF structures in the file (see the included TIFF6.pdf specification).
  - ImageFileHeader
  - ImageFileDirectory
  - ImageFileEntry
- Extract the entries (IFEs) that pertain to GPS info (see table below).
- Convert the GPS info into coordinates (degrees/minutes/seconds to decimal degrees).

Once your program gives you the coordinates, you should manually look them up. Where was this image taken? Use property_tags.txt to help you find other IFEs. What else can you tell us? Submit a file in addition to your program with the location (a name, not coordinates) and any other information you find.

Your program should accept a file path via standard in and report the coordinates to standard out. You are **not** required to write a component to do the GPS lookups automatically.

IEF Tags of Note	
Value	**Definition**
0x0001	PropertyTagGpsLatitudeRef
0x0002	PropertyTagGpsLatitude
0x0003	PropertyTagGpsLongitudeRef
0x0004	PropertyTagGpsLongitude
0x8825	PropertyTagGpsIFD

IEF Types of Note	
Value	**Definition**
0x0001	Array of bytes
0x0002	Null-terminated ASCII string
0x0003	Array of UInt16 integers
0x0004	Array of UInt32 integers
0x0005	Array of rational numbers (UInt32 pairs comprised of a numerator and denominator)
0x0006	Array of bytes that hold an unspecified data type
0x0007	Array of Int32 integers
0x000A	Array of rational numbers (Int32 pairs comprised of a numerator and denominator)