Bitcoin Mining

It's quite a simplification, but Bitcoin mining is the search for structure in a cryptographic hash output. To mine a Bitcoin your miner must find an input that when hashed produces an output that begins with a certain number of 0 bits. The Bitcoin ecosystem makes mining a Bitcoin harder or easier by changing the number of 0s needed. Finding a hash input whose hash output begins with twenty 0 bits is a lot easier than finding one whose output begins with thirty.

Write an OpenSSL program that takes a positive integer n as a command line argument and outputs a datum whose SHA2-256 hash output begins with n or more 0 bits. To simplify the task a little, you may assume n will be a multiple of 4 and you should restrict your input to ASCII character sequences of no more than 60 characters, where all the characters are letters and digits. Your program should print your found ASCII sequence as text followed by its SHA2-256 hash in hex. For example the hash of "abcdefghijklmn" (excluding the null-terminating byte in the C string) is `0653c7e992d7aad40cb2635738b870e4c154afb346340d02c797d490dd52d5f9`. If I ran the program with command line input 4, then a possible output would be:

```
> miner 4
abcdefghijklmn
0653c7e992d7aad40cb2635738b870e4c154afb346340d02c797d490dd52d5f9
```

In the example I am invoking a program I named "miner" with a command line input of 4. The program then finds an ASCII sequence whose hash of its bytes (0x6162636465666768696A6B6C6D6E in this case) produces a hash output beginning with hex 0 (which is four bits of 0).

Your solution must use the EVP interface for hashing as described in the OpenSSL Wiki.

Some suggestions:

1. Learn how to use OpenSSL for cryptographic hashing be reading about Message Digests at the OpenSSL Wiki and reading about each called function in the OpenSSL man pages (for example if you internet search for "man EVP_DigestInit_ex" you will find a description of what it does).
2. Search the internet for "reading C command line arguments" or something similar to see some examples of how to read values from the command line. You will want to use `atoi` or `strtol` to convert the string in `argv[1]` into an integer. So, read the man pages for these and/or search the internet for examples of their use. You may assume that your program is run with a positive multiple of 4 on the command line, so you do not need to make your program error-check this.
3. In C, characters are just small integers (ie, 0-255 for unsigned char). This means that you can increment them to get the next character. For example, `'a' + 1 == 'b'` evaluates to true. The following little snippet of code will cause `x` to cycle through all of the letters 'a' through 'z' over and over in an infinite loop.

   ```c
   unsigned char x = 'a';
   while (x <= 'z')    // As long as x is 'z' or before
       x = x + 1;      // Increment x to the next letter
       if (x > 'z')    // If x is now past 'z'
           x = 'a';    // Reset x to 'a'
   ```

4. One way to generate a sequence of ASCII strings is to begin with a legal string and then repeatedly "increment" it to another legal string. See Program 3's `increment_block` function for an example of how to structure such an increment.
5. If you initialize a string like `unsigned char str[] = "abc"`, the C compiler creates an array of length 4 initialized with the values 0x61, 0x62, 0x63, 0x00 (search the internet for "ascii table" for a list of every character's ascii value). You could then hash the buffer `str` as a length 3 buffer (you are not supposed to include the terminating 0x00 in your hashes in this problem), and `printf("%s\n", str)` would work for printing `str` because it's a legal C string.
6. I often use a function like this to print memory as a sequence of hex.

```c
void pbuf(void *buf, int len)
{
    unsigned char *p = (unsigned char *)buf;
    for (int i = 0; i < len; i++)
        printf("%02x", p[i]);
    printf("\n");
}
```

7. You can verify your hashes on `ecs-pa-coding1` like so.

```
ecs-pa-coding1> echo -n abcdefghijklmn | sha256sum
0653c7e992d7aad40cb2635738b870e4c154afb346340d02c797d490dd52d5f9  -
```

Submission

1. This time you are submitting a file that contains a main program. Name your file "miner.c". Make sure your file compiles and runs without warnings or errors. I will compile your code with gcc flags `-Wall -Werror -fsanitize=address`. Wall "enables all the warnings about constructions that some users consider questionable, and that are easy to avoid". Werror turns all warnings into errors that prevent successful compilation. Fsanitize=address causes runtime checks to detect some (but maybe not all) illegal memory accesses.
2. The file you turn in should have a small comment block at the top containing your name, the date, anyone you worked with, and the URL of any online source that provided significant help to you. It is an important habit to give credit to important sources of help. For example:

```c
// contains.c by Ted Krovetz. Submitted for CSC 152 July 4, 1776.
// Pair-programmed with Ellen Watermellon
// Used the "reverse" code from https://www.techiedelight.com/reverse-a-c-string
```

3. Read carefully Fileinbox submission. Follow that procedure to submit only the file miner.c

Collaboration

You may collaborate with *one or two* other students on this homework if you wish, or work alone. Collaboration must be true collaboration however, which means that the work put into each problem should be roughly equal and all parties should come away understanding the solution. Here are some suggested ways of collaborating on the programming part.

- Pair programming. Two or three of you look at the same screen and only one of you operate the keyboard. The one at the keyboard is the "driver" and the other is the "navigator". The driver explains everything they are doing as they do it and the navigator asks questions and makes suggestions. If the navigator knows how to solve the problem and the driver does not, the navigator should not dictate solutions to the driver but instead should tutor the driver to help them understand. The driver and navigator should switch roles every 10-15 minutes. Problems solved this way can then be individually submitted.
- Code review. The members of the collaborative each try to solve the problem independently. They then take turns analyzing each other's code, asking questions trying to understand each other's algorithms and suggesting improvements. After the code reviews, each of you can then fix your code using what you learned from the reviews. Do not copy code. If the result of code review is that your code needs changes to be more like your partner's, do not copy it. Instead recreate your own variant without looking at your partner's.

The goal is to learn enough from one another so that you each can do similar problems independently in an exam situation.

If you want a collaborator but don't know people in the class, you can ask on Discord and/or use the group-finding post on Piazza.