Lab: SHA256

File `SHA256.cry` contains Cryptol functions that implement the SHA256 hash plus create a digest for a given text string. Of specific interest is the function `SHA256 msg` which produces a digest from input text `msg`. The function is defined like this:

```
SHA256 : {a} (fin a, 64 >= width (8*a)) => [a][8] -> [256]
SHA256 msg = join (SHA256' [ split x | x <- preprocess(join msg)])
```

An example run of this function is as follows:

```
SHA256> SHA256 "Hello World Folks"
0xd14155c5fb4dbbb2f8d1d3ade275982a610bc50ff85389a1093875b85993cfeb
```

Notice from the signature of `SHA256` that the output is 256 bits wide. File `sha-256.c` contains C code for producing a SHA256 hash plus a digest from function `calc_sha_256` on input `const void* input`. Adding a `main` like this:

```
   int main (int argc, char** argv) {
   if (argc != 2) {
      printf("Yikes!\n");
      exit(0);
   }
   int i;
   uint8_t hash[32];
   calc_sha_256(hash, argv[1], strlen(argv[1]));
   for (i=0 ; i < 32 ; i++) printf("%2x",hash[i]);
   printf("\n");
}
```

and compiling allows one to display the digest given an input string like this:

```
[prompt]$ sha-256 "Hello World Folks"
d14155c5fb4dbbb2f8d1d3ade275982a610bc50ff85389a1093875b85993cfeb
```

Observe that for the same input, the digest is the same. It is desired to show that the C function for computing the digest is functionally identical to the Cryptol “gold standard”. But the output of `calc_sha_256` is in array `hash` which is an array of 32 bytes. In Cryptol this is expressed as `[32][8]`. But the Cryptol code outputs a digest of 256 bits, expressed as `[256]`. Something must be done to change the Cryptol code so that it can verify the functionality of the C code.

Exercise 1:
Run Cryptol, then load `SHA256.cry`. At the prompt run `SHA256 "Hello World Folks"`. Verify that the result is the same as above. ■

Exercise 2:
Compile `sha-256.c` to `sha-256` from the command line. Run

```
sha-256 "Hello World Folks"
```

from the command line. Verify that the result is the same as above. ■

Exercise 3:
Create a function digest_in_bytes in `SHA256.cry` with signature

```
digest_in_bytes : {i} (fin i, 64 >= width (8*i)) => [i][8] -> [32][8]
```

That takes a message `msg` as input and outputs 32 bytes that is the digest of `msg`. Run

```
digest_in_bytes “Hello World Folks”
```

and verify the result matches the above except that the output is now a sequence of bytes. ■

Exercise 4:
The digest function of Cryptol does not require the length of the input message as input. The C function should not either. Write a C function named SHA256_Buf_Wrapper that takes, as input, the message, as a `char[]`, and the 32 byte digest array and inserts the digest of the message into the digest array. The prototype for the wrapper function is the following:

```
void SHA256_Buf_Wrapper(char *input, uint8_t digest[32]);
```

Then change `main` to look like this:

```
int main (int argc, char** argv) {
   // usage: sha256 <input>
   int i;
   uint8_t digest[32];
   SHA256_Buf_Wrapper(argv[1],digest);
   for (i=0 ; i < 32 ; i++)
      if (digest[i] < 16) printf("0%x",digest[i]);
         else printf("%x",digest[i]);
         printf("\n");
```

Run sha-256 “Hello World Folks” and verify the output is as above. ■

Exercise 5:
Following the pattern of the previous lab, construct a saw file that verifies that the C function `SHA256_Buf_Wrapper` on input `msg` produces the same output as the Cryptol function `digest_in_bytes` on `msg`. ■