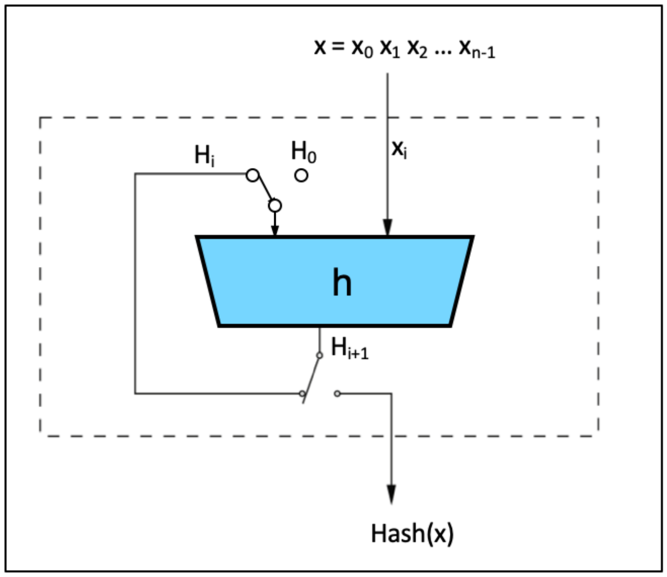
Learning Outcome: Students will gain experience on the construction of a simple cryptographic hash function and blockchain in C.

Lab Part 1: SHA_40 the Cryptographic Hash Function

Your first task in lab 5 is to construct a 40-bit cryptographic hash function named SHA_40 and investigate its security properties. A hash function is any function that can be used to map data of arbitrary size to fixed-size values. Hash functions play an important role in cryptography. They have been used for establishing data integrity and authentication. The value returned by a hash function is called a **digest** which can be considered as a fingerprint of the input message. A secure hash function possesses the following properties:

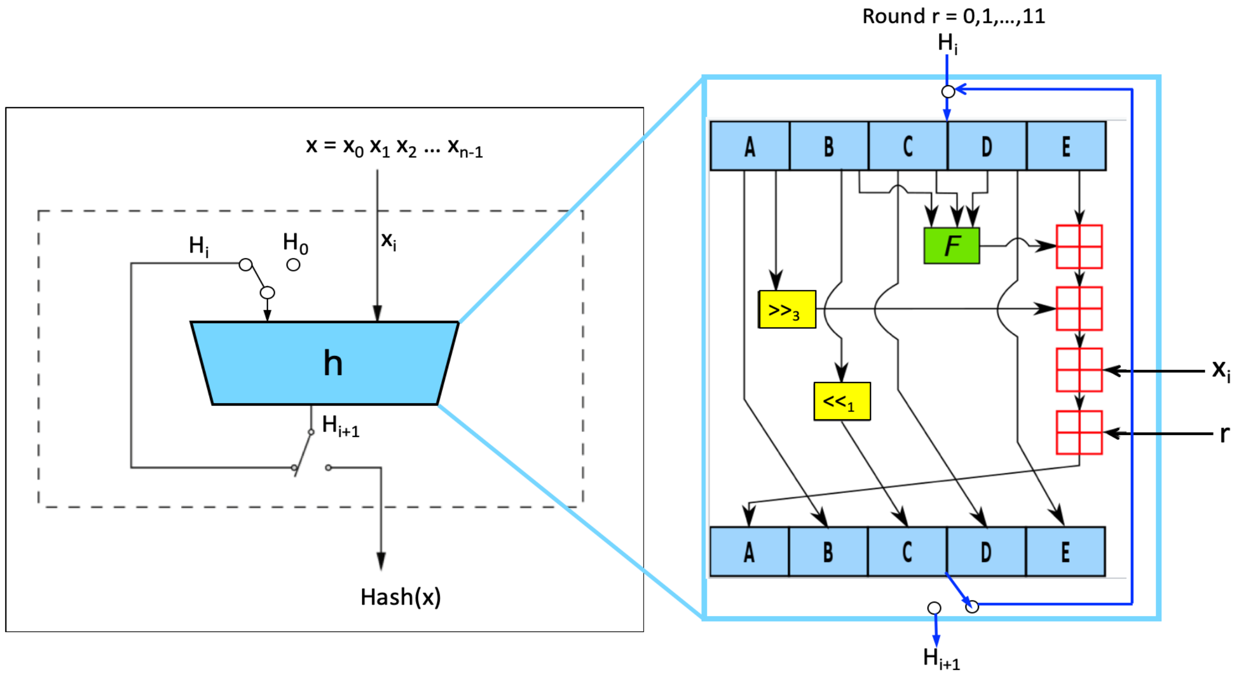
- Randomness – Two similar inputs must produce two seemingly unrelated random looking outputs.
- Onewayness – It should be easy to compute hash digest for a given input, whereas it should be hard to compute an input message that produces a given output.
- Collision resistance – It should be hard to find two different messages that result in the same hash digest.

How can hash functions process an arbitrary-length message and produce a fixed-length output? In practice, this is achieved by segmenting the input into a series of blocks of equal size. These blocks are processed iteratively by a function called compression function, denoted as **h** in the diagram below. The current output of the compression function is used as input in the next iteration. This iterative design is known as Merkle–Damgard construction which is in fact used by many cryptographic hash functions including the SHA family and the one you are going to implement here.



<Merkle–Damgard construction>

In SHA_40, each block x_i of message x is a byte and hash output is 40 bits, a sequence of 5 bytes. The compression function **h** takes a byte $\mathbf{x_i}$ and the previous output $\mathbf{H_i}$ as input and produces 40 bits output $\mathbf{H_{i+1}}$,for each i = 0, 1, …,n-1,



<Internal Structure of SHA_40>

- SHA_40 can process an arbitrary-length message and produces a 40-bit output.
- **x** denotes a sequence of bytes (unsigned characters) $x_0x_1\dots x_{n-1}$. These bytes are processed sequentially by the **h** function. The function consists of 12 rounds, each of which performs identical operations. More precisely, i-th round takes x_i and H_i as input and generates a sequence of 5 bytes H_{i+1}. Each H_i comprises five bytes, denoted by A, B, C, D, and E.

- The hash digest, SHA_40(**x**), is then defined as the output of the last iteration of the **h** functions.
  - H_0 is the initial seed value. Let H_0 = {11, 22, 33, 44, 55}.
  - $H_{i+1} = h(H_i, x_i)$, for i = 0, 1, …, n-1
  - H_n = SHA_40(x)

- The **h** function uses bit shifting and boolean operators to scramble the bits efficiently.
  - **<<** and **>>** are bitwise shift left and shit right operators respectively.
  - **F**(B, C, D) = (B & C) ^ D where **&** is a bitwise AND operator and **^** is a bit wise XOR operator.

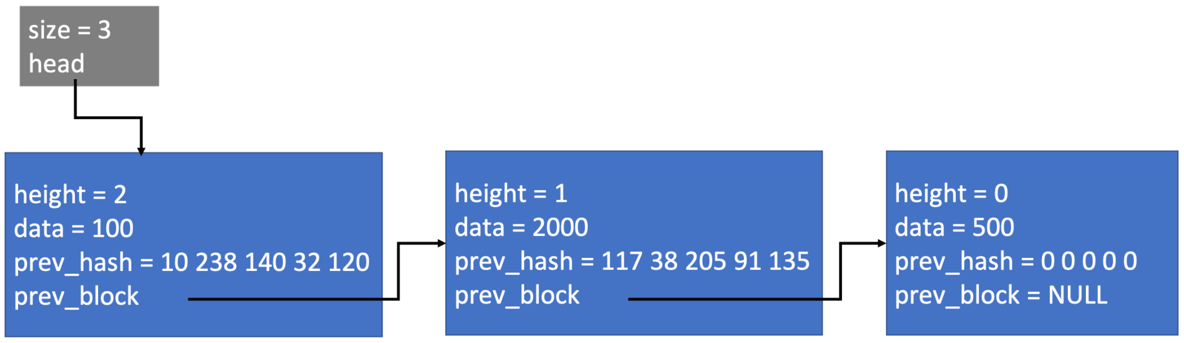
Requirements

First, open **hash.h** file and read it. Create a file named `hash.c` and write the following functions:

- **SHA_40**, which takes a message of arbitrary length and generated a 40-bit hash value using the sha_40 algorithm
- **digest_equal**, which indicates whether or not two digests are equal
- **main**, which tests the functions above

Lab Part 2: Blockchain

Equipped with linked lists from lab 4 and the hash function SHA_40, you are now ready to implement a simple blockchain from scratch. A blockchain is a growing list of records, called blocks, that are linked with cryptographic operations. In addition to data and a pointer to the previous block, each block in a blockchain contains a hash digest of the previous block. Therefore, a block not only tells us where the previous block is but also allows us to check integrity of the blockchain. In other words, we can verify that the block hasn't been tempered by malicious attackers.



- First, read the **blockchain.h** file. Open **blockchain.c** and define the following functions in order:

  - **set_digest,** which used to set the SHA_40 digest of a block
  - **add**, which inserts a new value into the blockchain
  - **verify**, which verifies that a blockchain has not been tempered. This function must compute digest of the previous block and compare it against the digest value stored in the current block.

- There are three attack functions you will add to **blockchain.c**.
  - **delete**, which deletes a block
  - **alter_block**, which alters data in a block
  - **alter_2_blocks**, which alters data in a block and sets the hash value stored in the next block to the digest of the altered block

- More details about the functions can be found in the **blockchain.h** file.

- If you want to get **extra 10 points**, do this.
  - First, add a new attribute `int nonce;` to `struct Block`.
  - Copy-paste your **add** function and rename it **mine_block**.
  - This function implements the proof of work system that Bitcoin blockchain uses in order to control block creation time. You will need to select a `nonce` of the new block that makes hash digest of the block "small" enough; the first 12 bits of digest must be all zeros.

Compile and Run:

- Comment out the main function in hash.c.
- $ gcc -o blockchain hash.c blockchain.c
- $ ./blockchain

Deliverables:

- You must upload working code, **hash.h, blockchain.h, hash.c and blockchain.c** under the Lab5 directory in GitHub Assignment 5 by 9/29/2021 7 PM.

Sign offs – Each signature is worth 1/N of your lab grade where N is the number of signatures

- (25 points) Student wrote the **add** function properly using SHA_40.
- (25 points) Student could **verify** the integrity of blockchain.
- (25 points) Student could emulate attacker behaviors: **alter_block, delete**
- (25 points) Student could emulate attacker behaviors: **alter_2_blocks**
- (Extra 10 points) Student could write **mine_block** function.