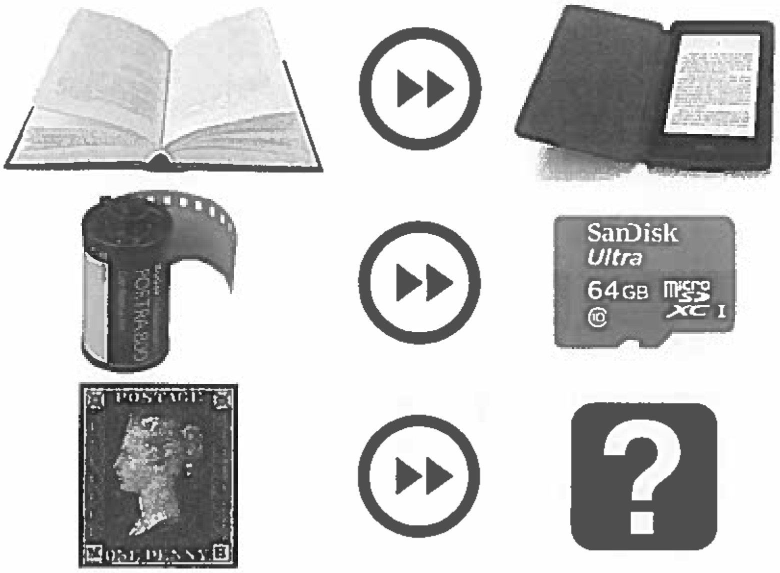
```
MSc (Computing Science) 2017-2018
C/C++ Laboratory Examination
```

Imperial College London

Monday 8 January 2018, 14h00 – 16h00



- ☞ You are advised to use the first 10 minutes to read through the questions.
- ☞ Log into the Lexis exam system using your DoC login as both your login and as your password (**do not use your usual password**).
- ☞ You must add to the pre-supplied header file **stamp.h**, pre-supplied implementation file **stamp.cpp** and must create a **makefile** according to the specifications overleaf.
- ☞ You will find source files **stamp.cpp**, **stamp.h** and **main.cpp**, and data files **message1.txt**, **message2.txt** and **message3.txt** in your Lexis home directory (**/exam**). If one of these files is missing alert the invigilators.
- ☞ **Save your work regularly.**
- ☞ Please log out once the exam has finished. No further action needs to be taken to submit your files.
- ☞ No communication with any other student or with any other computer is permitted.
- ☞ You are not allowed to leave the lab during the first 15 minutes or the last 10 minutes.
- ☞ **This question paper consists of 5 pages.**

Problem Description

Spam email, i.e. unsolicited bulk email, is a pervasive nuisance that accounts for around 60% of global email traffic[1]. Your task today is to implement a scheme that is designed to tackle this problem[2]. The scheme works by making senders "pay" for every email they send by having to invest computational effort in the creation of a "digital stamp" added to email headers. The validity of the stamp can be verified by a receiver at negligible computational cost. The idea is to transfer the cost of spam from receivers to senders, thus rendering the business models of spam email senders unprofitable.

Our solution will make use *hash functions*. A hash function is a function which maps data of arbitrary length onto a fixed size fingerprint known as a *digest*. The hash function we will use is called SHA1[3]. The digest for this function is 20 bytes long and is most conveniently displayed in human-readable form as 40 hexadecimal digits. For example, the SHA1 digest of the message

```
The bitcoin is under the carpet in the living room.
```

is `60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5`.

Hash functions are designed such that changing the input, even slightly, will result in a very different digest. For example, the SHA1 digest of the message

```
The bitcoin is under the carpet in the dining room.
```

is `29c22640a6df69a7fcb9d8d4923b90f817d36195`.

The process for sending an email works as follows:

- The sender computes the SHA1 digest of the *message body* they wish to send.
- The sender prepares a header of the following format:

  `<recipient email address>:<SHA1 digest of message body>:<counter>`

  where *counter* is an integer initalised to 0. For example:

  `wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:0`

- The sender then computes the SHA1 digest of the *header*. If the first 5 hex digits (i.e. the first 20 bits) of the digest are zero, then this is an acceptable header. If not, the sender increments the counter and tries the hash again[4]. In the example above, the first counter value that gives an acceptable header is 313889, since the SHA1 digest of the header

  `wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:313889` num -> strg

  is `00000e269e4da81b701ed39646eafe17ae9aadd0`.

To validate a received email the receiver must:

- Confirm that the header is in the correct format (i.e. three fields separated by ':').
- Check the recipient mentioned in the first field of the header matches the expected recipient.
- Check that the SHA1 digest of the message body given in the header matches the actual SHA1 digest of the received message.
- Check that the SHA1 digest of the header is an acceptable header.

If all of these conditions are met, the email is valid (and can be accepted). Otherwise the email is invalid (and should be rejected).

[1] Source: `https://www.statista.com/statistics/420391/spam-email-traffic-share/`
[2] Inspired by the Penny Black Project at Microsoft Research and the Hashcash scheme proposed by Adam Back.
[3] Secure Hash Algorithm 1, designed by the National Security Agency of the USA.
[4] Thus, on average, the sender will have to try 2^{20} (±1 million) values to find an acceptable header.

Pre-supplied functions and files

You are supplied with a main program in **main.cpp**, and three data files containing the bodies of email messages: **message1.txt**, **message2.txt** and **message3.txt**. You can use the UNIX commands **cat** and **less** to inspect the data files, e.g.:

```
% cat message1.txt
The bitcoin is under the carpet in the living room.
```

You can verify what the corresponding SHA1 digests of these files are using the **sha1sum** utility:

```
% sha1sum message?.txt
60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5  message1.txt
29c22640a6df69a7fcb9d8d4923b90f817d36195  message2.txt
fe8e2cc69298b811f53dcfd54929d02cc65748e3  message3.txt
```

You are also supplied with the beginnings of the header file **stamp.h** (for your function prototypes) and the beginnings of the implementation file **stamp.cpp** (for your function definitions).

The file **stamp.cpp** includes the definition of a function:

```
void text_to_SHA1_digest(const char *text, char *digest);
```

which returns in **digest** the SHA1 digest of input string **text**. For example, the code:

```
text_to_SHA1_digest("The bitcoin is under the carpet in the living room.", digest);
cout << "SHA1 digest of message 'The bitcoin is under the carpet in the living room' is:"
  << endl << digest << endl;
```

has the output:

```
SHA1 digest of message 'The bitcoin is under the carpet in the living room' is:
60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5
```

Note that to link the code for this function you will need to add the "-lcrypto" flag at the end of the usual g++ arguments when linking. This makes g++ link your code with the OpenSSL cryptographic library that contains the definitions of the SHA1 hash function.

Specific Tasks

1. Write an integer-valued function **leading_zeros(digest)** which takes as its parameter a hexadecimal string representing a hash digest and returns the number of leading zeros it beings with. If the input string contains characters that are not valid hexadecimal digits (i.e. '0' to '9' and 'a' to 'f') then the function should return -1.

   For example, the code:

   ```
   cout << "The number of leading zeros in digest "
        << "'00000a184d72f39730312554e0df25e6f95a05d4' is "
        << leading_zeros("00000a184d72f39730312554e0df25e6f95a05d4");
   ```

   should display the output

   ```
   The number of leading zeros in digest '00000a184d72f39730312554e0df25e6f95a05d4' is 5
   ```

2. Write a boolean function `file_to_SHA1_digest(filename, digest)` which returns in `digest` the SHA1 digest of the contents of the file with name `filename`. If the file cannot be opened then the function should return `false` with `digest` set to "error"; otherwise return `true`.

   For example, the code:

   ```
     char digest[41];
     bool success = file_to_SHA1_digest("message1.txt", digest);
     cout << "SHA1 digest of file 'message1.txt' is " << digest << endl;
     if (!success)
       cout << "There was an error reading the file." << endl;
   ```

   should display the output

   ```
   SHA1 digest of file 'message1.txt' is 60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5
   ```

3. Write a Boolean function `make_header(recipient, filename, header)` which prepares a header for a given email message. Here `recipient` is a read-only input parameter specifying the email address of the message recipient, `filename` is a read-only input parameter specifying the file containing the message body, and `header` is an output parameter containing the header in form:

   ```
   <recipient email address>:<SHA1 digest of message body>:<counter>
   ```

   The return value of the function should be `false` if the file containing the message body, cannot be read, or if a valid counter is not found after 10 million attempts; otherwise the function should return `true`.

   For example, the code

   ```
     char header[512];
     bool success = make_header("wjk@imperial.ac.uk", "message1.txt", header);
     if (success)
       cout << "A valid header for 'message1.txt' is "<< header << endl;
     else
       cout << "There was an error generating the header." << endl;
   ```

   should result in the output

   ```
   A valid header for 'message1.txt' is
   wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:313889
   ```

4. Write a function `check_header(email_address,header,filename)` which features three input parameters: the user's email address, a received message header and the name of a file containing the text of a received email body. The return value should of enumerated type `MessageStatus` (see **stamp.h**), which can be printed in a human-readable manner using the `message_status(...)` internal helper function in **main.cpp**.

   For example, the code:

   ```
   cout << "Checking email with header " << endl
        << "'wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:313889'"
        << endl << "sent to 'wjk@imperial.ac.uk' with body 'message1.txt':" << endl;
   MessageStatus result = check_header("wjk@imperial.ac.uk",
     "wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:313889",
     "message1.txt");
   cout << message_status(result) << endl;
   ```

   should result in the following output:

   ```
   Checking email with header
   'wjk@imperial.ac.uk:60eb8db1a3c0ea1ad3d5548f248b4a3e73711ee5:313889'
   sent to 'wjk@imperial.ac.uk' with body 'message1.txt':
   Valid message.
   ```

(The four parts carry, resp., 20%, 25%, 25% and 30% of the marks)

What to hand in

Place your function implementations in the file **stamp.cpp** and corresponding function declarations in the file **stamp.h**. Use the file **main.cpp** to test your functions. Create a **makefile** which will compile your submission into an executable file entitled **stamp**.

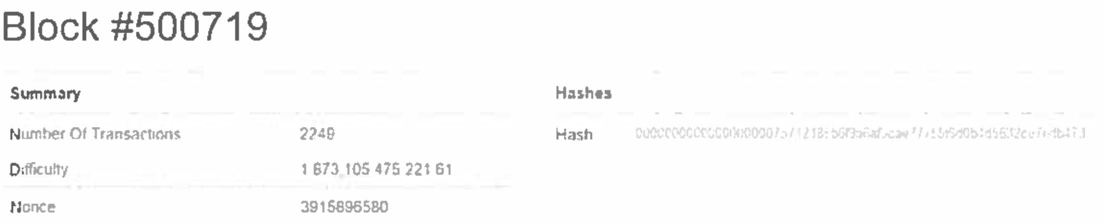
Hints

1. You will save time if you begin by studying the main program in **main.cpp**, the pre-supplied implementation file **stamp.cpp**, the pre-supplied header file **stamp.h** and the given data files **message1.txt**, **message2.txt** and **message3.txt**.
2. Question 2 will be **much** easier if you exploit the pre-supplied helper function.
3. The standard header `<cctype>` contains some library functions that you may find useful. For example, `int isxdigit(char ch)` returns nonzero if `ch` is a valid hexadecimal digit.
4. Feel free to define any of your own helper functions which would help to make your code more elegant. This will be particularly useful when answering Question 4.
5. Try to attempt all questions. If you cannot get one of the questions to work, try the next one.
6. You are not explicitly required to use recursion in your answer to any of the questions. However, you may make use of recursion wherever you feel it would make your solution more elegant.
7. **Do not forget to include "-lcrypto" as the last argument to g++ in your linking step.** If you receive the error **undefined reference to 'SHA1'** then you have forgotten to do this.

Additional Context

You should only read this section when you have completed all tasks and checked your solution, or after the examination.

You may be interested to learn that the activity of mining cryptocurrencies like Bitcoin is similar to the digital stamping technique presented above. To publish a block of transactions, you need to find a so-called "nonce" (number-used-once, like our counter) which, when added to the block header, results in a hash with sufficient leading zeros. Consider the following bitcoin block:



Block #500719

Summary		Hashes	
Number Of Transactions	2249	Hash	0000000000000000000075711218c56f3b6afcae7745f5d0b1d5632ee7c4b473
Difficulty	1 873 105 475 221 61		
Nonce	3915896580		

Note the block hash, which has many leading zeros, and the "nonce", which, when added into the block header, has been discovered by the publisher of the block (the miner) to yield the low hash.