ENEL206: Computing and Modelling
Laboratory 011
Bitwise Manipulation

Andrew Bainbridge-Smith, A404

May 27, 2008

1 Objective

In this laboratory session you will write C code functions to decode encrypted text messages. The purpose of the exercise is familiarise yourself with the bitwise manipulation operations and with character codes.

2 Housekeeping

Download the zip file for the laboratory (lab3.zip). This file contains 3 encoded messages and 3 source files: *main.c, crypt.c, crypt.h*.

You are only required to edit the file *crypt.c* for this laboratory.

The program is compiled using:

```
gcc -Wall -I. -lm -o crypt main.c crypt.c
```

When you run the program 3 or 4 command-line parameters are required; the encoder/decoder method name, an optional key, the input message filename, and the output message filename.

```
./crypt caesar --charkey e input_message output_message
./crypt decode input_message output_message
```

You are required to decode the 3 messages in order. Instructions for decoding the first message are given below. Instructions for decoding the next message are given in this encrypted message.

3 Caesar Encryption

The first message is encrypted using *Caesar's* method. Developed in the time of Julius Caesar a message is encoded by substituting each letter in the alphabet with another. Decoding the message

required knowing the substitution. In this case the substitution process is a simple shift process along the alphabet, consider the following example of a 3 position shift:

$$a \rightarrow d$$

$$b \rightarrow e$$

$$c \rightarrow f$$

$$d \rightarrow g$$

$$\ldots$$

$$w \rightarrow z$$

$$x \rightarrow a$$

$$y \rightarrow b$$

$$z \rightarrow c$$

So in Caesar encryption the only information required to decode the message is the shift. This key is supplied by giving the letter that corresponds to 'a', in the example above this is the letter 'd'.

In this laboratory you will be required to write decoding functions that follow the function prototype form of:

```
void functionname (buffer_t* inmessage, buffer_t* outmessage, void* key);
```

The datatype for *buffer_t* is declared in *crypt.h*. This structure type contains members that detail how long the message is (*length*), an array containing the message (*buffer*) and the size of the buffer in bytes (*buffer_size*). Note that usually *buffer_size* >> *length* and it shouldn't be necessary to manipulate or use this member of the structure. The function *new_buffer(message_length)* is used to create an instance of the *buffer_t* datatype. Have a look at *main.c* to see how it is used, but it shouldn't be necessary to call it yourself.

The key is provided as a pointer to void. This is done so that any sort of datatype can be passed to our function. In the case of Caesar encrypt and decrypt we would like to provide a single character as the key. This must be done providing the address of the character when calling the function and the Caesar encoding/decoding function must first 'cast' the void pointer into a character pointer before using it.

Have a look at the Caesar encryption routine given in *crypt.c* to see how an encoding/decoding function is written using the required prototype, the *buffer_t* datatype and use of the key. You should note the encrypting routine can encode the entire ASCII table of 256 characters.

Now write a function called 'decaesar' to perform decoding of a Caesar encoded message. Edit the function *get_encoder()* to associate your function with a command-line name for the function. You will see this has already been done for both the Caesar encoder and the decoder (just uncomment the lines for the decoder).

When writing the decoders for the other two messages you will need to follow the encoder prototype as above and edit the function *get_encoder()* to make the appropriate associations.

Once you have successfully written your Caesar decoding function (it is logically correct and everything compiles ok) it will decode the instructions for completing the next step of the laboratory exercise. The key for decrypting the message1.code file is 'e'. Continue with this next step.