Prince Lucky Santos ID:921665976
Github: LePuncake321 CSC415 Operating Systems

# Assignment 6 – Device Driver

**Description**:
This assignment is to create a loadable device driver that has some user/application functionality. It should be capable of taking in commands such as open, release, read, write, and at least one ioctl command. It must also be able to be unloaded, and indicate that it has been unloaded from the system. Additionally, it must have some kind of complex functionality that utilizes all the commands above and is able to be used by both a user of the terminal and an application, which in this submission will be a cypher that uses a key to determine what two character emoticons are assigned to each uppercase and lowercase letter in the alphabet.

**Approach**:

*Conceptual Plan (What will the device driver do)*:
The kind of device driver that I want to create is a cypher that is able to encode a string by converting the uppercase and lowercase alphabetical letters into two character emoticons, such as ":3" and "XD," and decode those emoticons back to their original case-sensitive letters.

Before delving into how to make the actual device driver, I want to figure out how to get this cipher working in concept. Currently, I am thinking of storing the first characters, the eyes, and the second characters, the mouths, into two separate arrays, while also storing an integer that determines the offset of the first emoticon being associated with the first ever letter ("a"). Since there are 52 letters in total to encode, I've determined that I need 4 kinds of eyes and 13 kinds of mouths.

```
Array of 4 eyes = [':', ';', 'X', 'B']
Array of 13 mouths = [')', '[', '<', 'P', '(',
                      '/', 'D', 'V', ']', 'O',
                      '|', '>', '3']
```

Additionally, I need to be able to store the mode of the device driver, either encoding the user's inputted alphabet characters into the cipher's emotes or decoding such emotes back into their respective alphabet character. Finally, I need to be able to store the offset from the first alphabetical character possible, 'A', so that the user can use different kinds of ciphers.

Finally, I have to know how to convert the indexes in the eyes and mouths arrays into an offset integer, where it would determine the letter associated with an emote, .
[How to convert from letter to emote then back]

*Implementation Plan and Execution (How to make the kernel driver work)*:

https://tldp.org/LDP/lkmpg/2.4/html/book1.htm
https://lwn.net/Kernel/LDD3/

Now for the implementation of the device driver, I need to figure out how to make the device driver in the first place. I had to learn about the file_operations struct that allows me to connect new implementations of open, release, read, write and ioctl methods. I also needed to learn how to use copy_to_user and copy_from_user, which are the only ways I can read and write data between the kernel and the user space, as well as how to use printk() for the sake of debugging and showing the state of the methods I was using. For the methods mentioned beforehand, I used online resources as well as the lecture recordings to learn about how to implement a fully functional general device driver. Finally, I needed to learn how to build the device driver, as well as install and uninstall the kernel module so that it can be used by user programs. I managed to figure this out by asking around with fellow classmates as well as looking at one of the lectures

As for the specific functionality of the device driver, which is encoding alphabetical letters into two-character emotes and decoding vice versa, I need to plan out how to go through each character in one buffer to convert lower and uppercase alphabets into two character emotes, and vice versa, all while taking into account the device driver's offset. I was able to figure out the calculations I needed to do for converting letters into emotes and emotes in letters:

## For converting a letter into a two-character emote:

Firstly, set the position based on if it is uppercase (start at 0 since they represent the first half of the emotes) lowercase (start at 26 since every position below represents uppercase)

> For uppercase, = `'A' - letter`
> For lowercase, = `'a' - letter + 26`

After getting this new position, find position in both the eyes and mouths arrays by:
1. Adding it with driver's offset and calculate its remainder with the total number of alphabetical character, resulting in the total table index

```
tableIdx = (offset + newPos) % 56
```

2. Finding mouth index by finding the remainder of the total table index with the number
of mouths for the emotes

```
mouthIdx = tableIdx % 13
```

3. Finding eye index by dividing the total table index by the number of eyes for emotes

```
eyesIdx = tableIdx / 13
```

## For converting a two-character emote into a letter:

Firstly, find both the mouth and eye indexes of the current two characters. If a valid emote is found, use those indexes to find the relative position from the first letter/emote

1. Before everything, we need to find what emotes represents 'A' being the first ever alphabet represented by the tables through the offset

```
firstMouthIdx = (offset % 56) / 13

firstEyeIdx = (offset % 56) % 13
```

2. We then need to convert the mouth index of the current emote into the relative mouth index from the first emote []

> If the first mouth index come before the current mouth index, then it is a simple subtraction to find the relative index

```
relMIdx = firstMouthIdx - currMouthIdx
```

> If the first index comes after the current one, then you need to account for the position wrapping around the array, with an additional subtraction to the array's size

```
relMIdx = 13 - (firstMouthIdx - currMouthIdx)
```

3. We afterwards need to do the same for the eye index, using similar operations

> When first comes before current, …

```
relEIdx = firstEyeIdx - currEyeIdx
```

> When first comes after, …

```
relEIdx = 4 - (firstEyeIdx - currEyeIdx)
```

4. From this point, we need to consider the condition where the first mouth index is not 0, in which mouth indexes before the first could possibly increment the eye index unintentionally. We must decrement the relative eye index if ever the current eye index is less than the first eye index, and make sure that it does not go negative.

```
if (currMouthIdx < firstMouthIdx) {
     relEIdx--;
     if (relEIdx < 0) {
          relEIdx = 3
     }
}
```

5. With these relative indexes, we can finally calculate the position relative to the first ever letter (‘A’), adding the characters passed by both eyes and mouth arrays; With each pass in the eyes array, there were 13 emotes that were passed, hence the multiplication by 13 with the eye index

```
letterPos = (relEIdx * 13) + relMIdx
```

4. Finally, we’ve found our alphabetical character, making sure to account for uppercase or lowercase (subtract by 26 since arrays represent them at 26)
    > For uppercase, = `‘A’ + letterPos`
    > For lowercase, = `‘a’ + letterPos - 26`

Finally for the device driver’s functionality, I need to figure out how to use the integers representing the offset from the first emote / letter as well as the mode for the device driver. I concluded that the emote cipher should be able to handle both negative and positive integers, meaning I have to take into account negative numbers within my calculations of letters to emotes and emotes to letters. As for the mode, I’ve decided to make it so that 0 represents encoding mode where the cipher converts alphabetical letters into emotes, while any other non-zero number will put the cipher into decode mode, where it converts emotes into alphabetical letters.

*In terms of the device driver functions, these are what they do*:

1. open() => Open a module file descriptor of the device driver for a program to use
2. release() => Release the module file descriptor
3. ioctl() => After the file descriptor parameter, there are two choices for the second parameter, which takes an unsigned integer:
    a. 0 = Changes the mode of the emote cipher’s mode field, which determines if it will convert letters into emotes, which is essentially encoding, (0) or emotes backs to letters, being decoding, (any non-zero number), to the third parameter, which is an unsigned long.
    b. 1 = Changes the offset of the emote cipher, which can go negative or positive, to the third parameter, which is an unsigned long.
    c. Any other input on the second parameter will do nothing, and return an error
4. write() => Takes in a buffer and writes it onto an original message array, from which it will convert its contents into an translated message (dependent on the mode and offset)

5. read() => Pass in a buffer to read the contents of the translated message (Will not allow the reading of the original message, or the user's buffer, from write())

*Testing Program (How would the user test and utilize the device driver)*:

Now for the testing program, I need to figure out how to utilize the device driver itself, and somehow demonstrate its functionality with all the file_operations methods assigned to it. I managed to learn how they work by looking back at the demonstration for the NULL device driver, where it's shown that device drivers act similarly to files. In fact, most of the operations done on the device driver use functions relating to interacting with files.

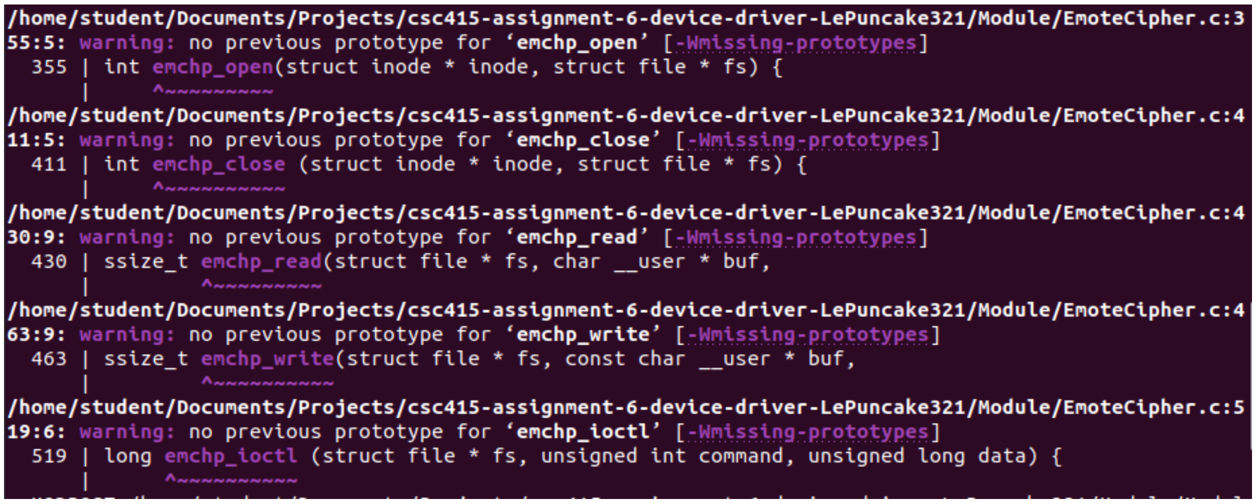
Finally, I need to figure out how to demonstrate the device driver with any kind of case. I concluded that my program will just run on the terminal like any simple input/output program; It will take in two integer numbers, one for setting the mode of the device driver's cipher (encoding letters into emotes or decoding emotes back to letters) as well as for exiting the program, as well as another for setting the offset from the first emote, allowing from the creation of unique ciphers on the fly, all of which will be passed onto the device driver's ioctl. After these two integer inputs, it will then take in a string message that will be passed on to the device driver's write to encode/decode it and get its translated version via read. The program will repeat this request for input until either the user inputs the exit value or they input an invalid type for both integers.

**Issues and Resolutions:**

Initially, I was having trouble wrapping my head around how to implement the device driver, especially with its specific functionality of converting letters into emotes and vice versa.

I decided that I should probably do the conversion/cipher functionality separately from the general device driver implementation, as well as to avoid making the read and write methods seem packed.

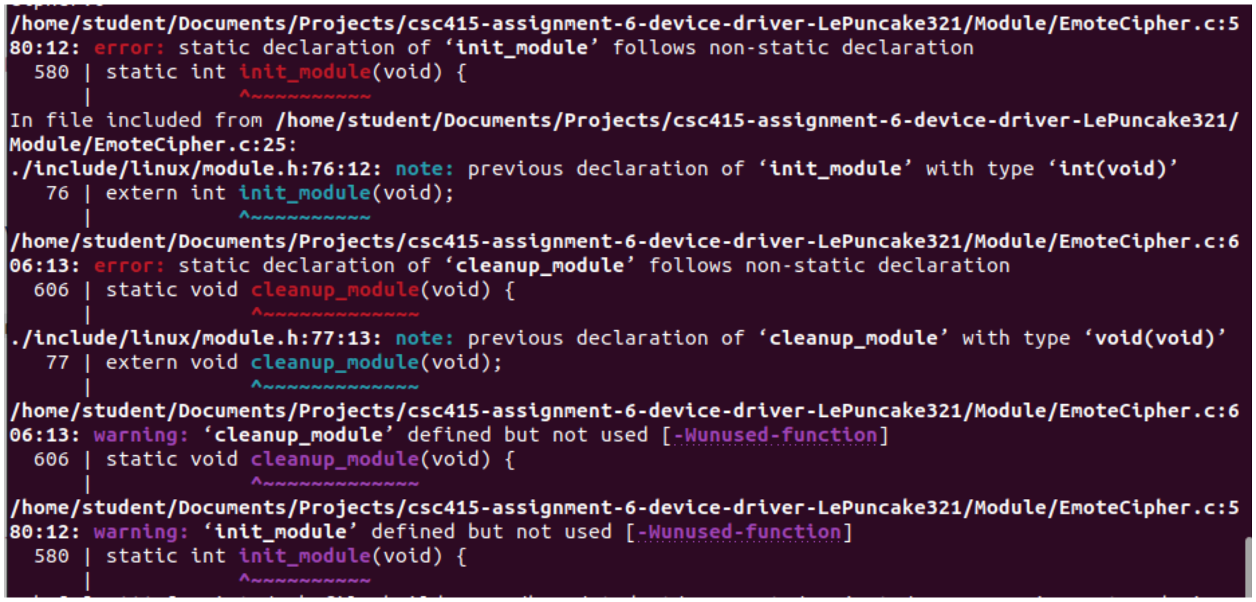
When trying to compile my code, I got various errors regarding some of the file_operations methods. It appears that these methods are not implemented properly by their method definitions.



```
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:3
55:5: warning: no previous prototype for ‘emchp_open’ [-Wmissing-prototypes]
  355 | int emchp_open(struct inode * inode, struct file * fs) {
      |     ^~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:4
11:5: warning: no previous prototype for ‘emchp_close’ [-Wmissing-prototypes]
  411 | int emchp_close (struct inode * inode, struct file * fs) {
      |     ^~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:4
30:9: warning: no previous prototype for ‘emchp_read’ [-Wmissing-prototypes]
  430 | ssize_t emchp_read(struct file * fs, char __user * buf,
      |         ^~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:4
63:9: warning: no previous prototype for ‘emchp_write’ [-Wmissing-prototypes]
  463 | ssize_t emchp_write(struct file * fs, const char __user * buf,
      |         ^~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
19:6: warning: no previous prototype for ‘emchp_ioctl’ [-Wmissing-prototypes]
  519 | long emchp_ioctl (struct file * fs, unsigned int command, unsigned long data) {
      |      ^~~~~~~~~~~
```

It turns out these file_operations methods need to be static methods, so I just ended up adding the static keywords to most of the methods needed for the device driver.

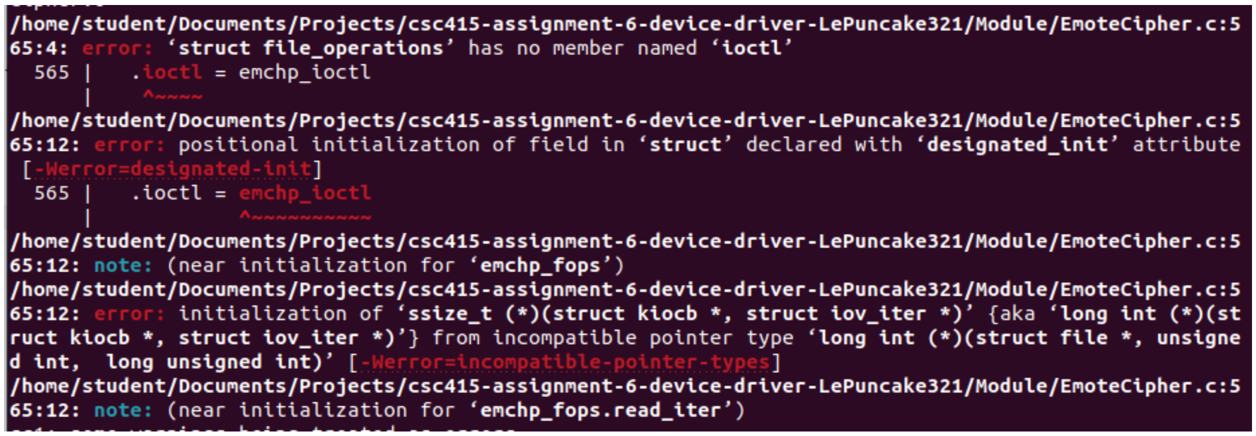
When trying to compile the methods for the device driver again, there are now these errors for the init_modules and cleanup_module, so it appears that the methods are again not implemented right by their definitions.



```
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
80:12: error: static declaration of ‘init_module’ follows non-static declaration
  580 | static int init_module(void) {
      |            ^~~~~~~~~~~
In file included from /home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/
Module/EmoteCipher.c:25:
./include/linux/module.h:76:12: note: previous declaration of ‘init_module’ with type ‘int(void)’
   76 | extern int init_module(void);
      |            ^~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:6
06:13: error: static declaration of ‘cleanup_module’ follows non-static declaration
  606 | static void cleanup_module(void) {
      |             ^~~~~~~~~~~~~~
./include/linux/module.h:77:13: note: previous declaration of ‘cleanup_module’ with type ‘void(void)’
   77 | extern void cleanup_module(void);
      |             ^~~~~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:6
06:13: warning: ‘cleanup_module’ defined but not used [-Wunused-function]
  606 | static void cleanup_module(void) {
      |             ^~~~~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
80:12: warning: ‘init_module’ defined but not used [-Wunused-function]
  580 | static int init_module(void) {
      |            ^~~~~~~~~~~
```

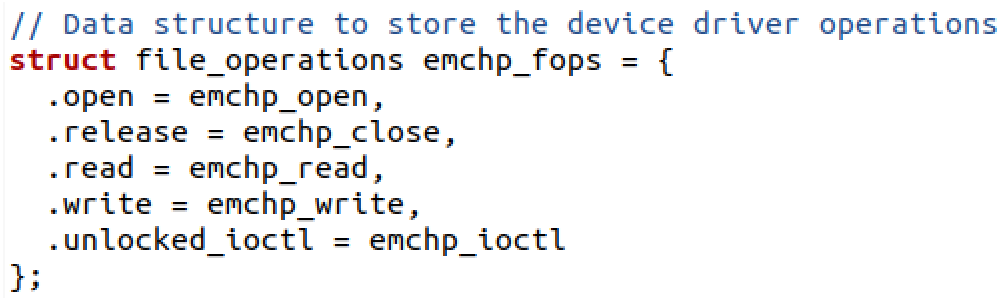
Looking into it, it turns out that these two methods don’t need the static keyword, as shown by the extern definitions in the terminal, so I just remove that keyword and it compiles properly.

For some reason, I can't assign my current ioctl implementation to .iotcl inside the struct file_operations, despite it being the way in which the demonstration program in class implemented the same method.



```
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
65:4: error: 'struct file_operations' has no member named 'ioctl'
  565 |   .ioctl = emchp_ioctl
      |    ^~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
65:12: error: positional initialization of field in 'struct' declared with 'designated_init' attribute
 [-Werror=designated-init]
  565 |   .ioctl = emchp_ioctl
      |            ^~~~~~~~~~~
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
65:12: note: (near initialization for 'emchp_fops')
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
65:12: error: initialization of 'ssize_t (*)(struct kiocb *, struct iov_iter *)' {aka 'long int (*)(st
ruct kiocb *, struct iov_iter *)'} from incompatible pointer type 'long int (*)(struct file *, unsigne
d int,  long unsigned int)' [-Werror=incompatible-pointer-types]
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:5
65:12: note: (near initialization for 'emchp_fops.read_iter')
```

Looking into the documentation for file_operations, it turns out that there is now two different kinds of ioctl that are in the struct: compat_ioctl and unlocked_ioctl. The second field now replaced the ioctl field of the struct, so I ended up changing the field, as well as the method's parameters since it does not need to take in one of them (I made this change before the screenshot above)

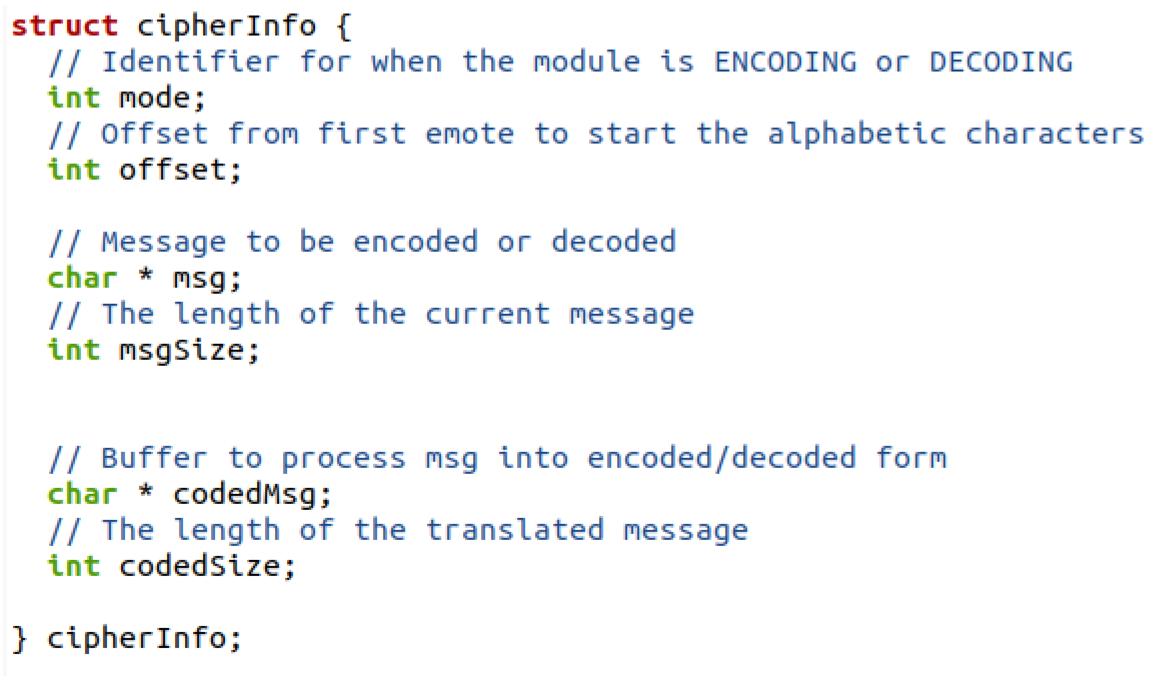


```
// Data structure to store the device driver operations
struct file_operations emchp_fops = {
  .open = emchp_open,
  .release = emchp_close,
  .read = emchp_read,
  .write = emchp_write,
  .unlocked_ioctl = emchp_ioctl
};
```

https://lwn.net/Articles/119652/

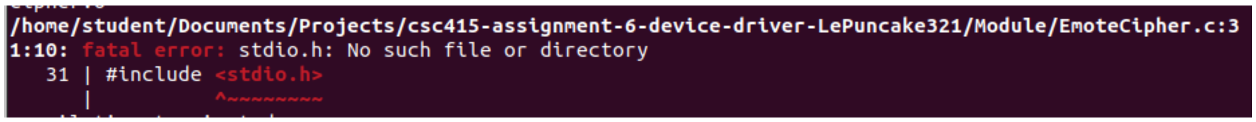
Now, I am trying to figure out how to convert the user's input between encoded and decoded versions; I couldn't figure out if I needed two buffers to separate these two cases or have one buffer for both cases.

Learning that data from kernel can only be directed to the user space through copy_to_user and copy_from_user, it will be difficult to have only one buffer handling the original message and then constantly calling the two methods above when the time comes to let the user read the translated message. So, I determine that I needed another buffer to hold the translated version of the original message written by the user, and because of this necessity, I also figured that the write method should be responsible for doing the encoding/decoding operation right after getting the user's message, while read will simply read the result of that operation.



```c
struct cipherInfo {
  // Identifier for when the module is ENCODING or DECODING
  int mode;
  // Offset from first emote to start the alphabetic characters
  int offset;

  // Message to be encoded or decoded
  char * msg;
  // The length of the current message
  int msgSize;


  // Buffer to process msg into encoded/decoded form
  char * codedMsg;
  // The length of the translated message
  int codedSize;

} cipherInfo;
```

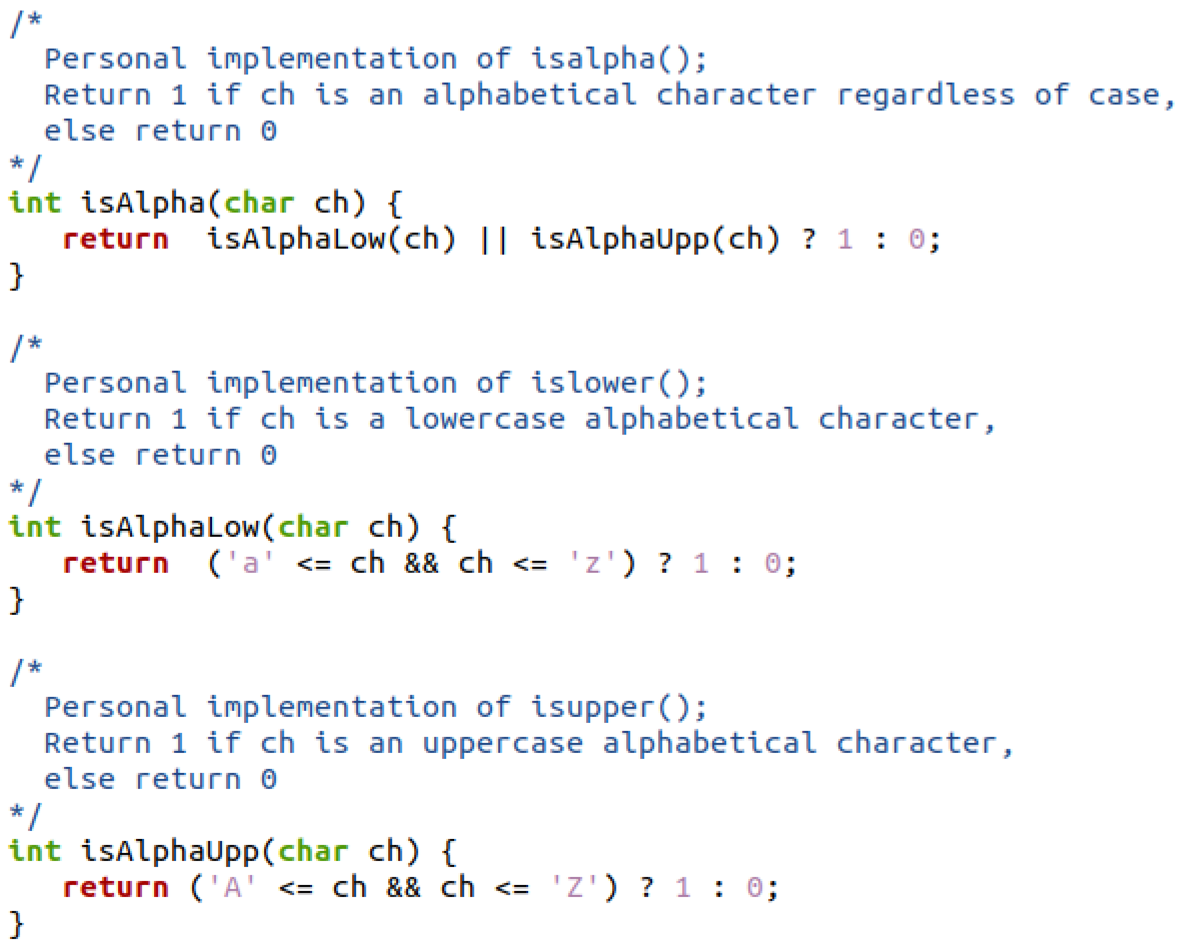
[ => Can’t use libaries in kernels and had to work with implementationt o not need these abs() (measures to prevent any negative numbers in the makeIntoLetter and makeIntoEmotes methods) and isalpha() (Created new method to replicate the function) methods]
For some reason, when trying to make the cipher aspect of the device driver, the stdio.h library, of which I am using abs() to deal with negative numbers and isalpha() , gives an error stating that it does not exist somehow.



```
/home/student/Documents/Projects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:3
1:10: fatal error: stdio.h: No such file or directory
   31 | #include <stdio.h>
      |          ^~~~~~~~~
```

Researching more about how the library was not appearing in a kernel module, it turns out that you can’t use libraries because these are outside of the scope of the kernel, so I need to make my own implementation of abs() and isalpha(). For abs(), I just have conditions that check if a number is lower than 0 and if so, add the number related to the array it represents.

For isalpha(), I ended up creating my own method that checks if a character is within the range of A-Z and a-z. Note that isAlphaLow and isAlphaUpp were implemented much later with a problem that comes later in the writeup, but the contents should somewhat be the same.

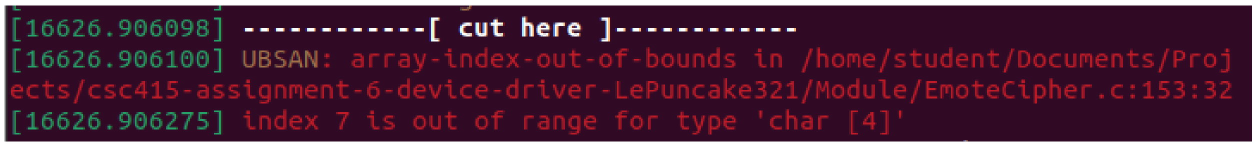


```
/*
  Personal implementation of isalpha();
  Return 1 if ch is an alphabetical character regardless of case,
  else return 0
*/
int isAlpha(char ch) {
   return  isAlphaLow(ch) || isAlphaUpp(ch) ? 1 : 0;
}

/*
  Personal implementation of islower();
  Return 1 if ch is a lowercase alphabetical character,
  else return 0
*/
int isAlphaLow(char ch) {
   return  ('a' <= ch && ch <= 'z') ? 1 : 0;
}

/*
  Personal implementation of isupper();
  Return 1 if ch is an uppercase alphabetical character,
  else return 0
*/
int isAlphaUpp(char ch) {
   return ('A' <= ch && ch <= 'Z') ? 1 : 0;
}
```

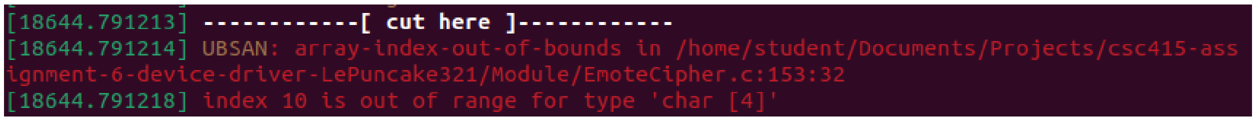
Within the test program, I tried to put an incorrect input but it ended up doing an infinite loop upon itself.

I checked through the code of the test program and made sure that instead of asking for the correct input from the user when an invalid one was given, the program will simply terminate from there and close their access to the device driver the conditions which kept repeating asking for integers until it found the correct input.

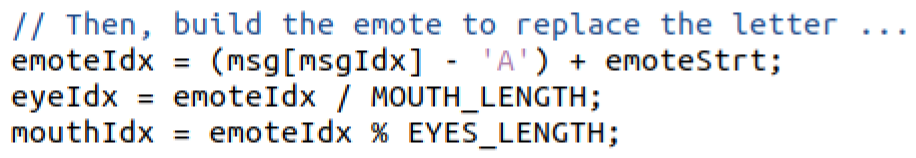
Despite the fixes I made in the Test program, the infinite loop is still happening. This means that there must be something completely wrong with the kernel module, and so I ended up finding some out-of-bounds errors that end up happening in the code relating to makeIntoEmotes.



```
[16626.906098] ------------[ cut here ]------------
[16626.906100] UBSAN: array-index-out-of-bounds in /home/student/Documents/Proj
ects/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:153:32
[16626.906275] index 7 is out of range for type 'char [4]'
```

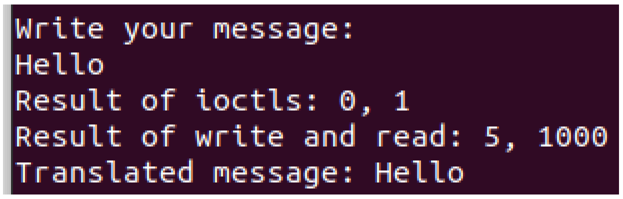


```
[18644.791213] ------------[ cut here ]------------
[18644.791214] UBSAN: array-index-out-of-bounds in /home/student/Documents/Projects/csc415-ass
ignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:153:32
[18644.791218] index 10 is out of range for type 'char [4]'
```

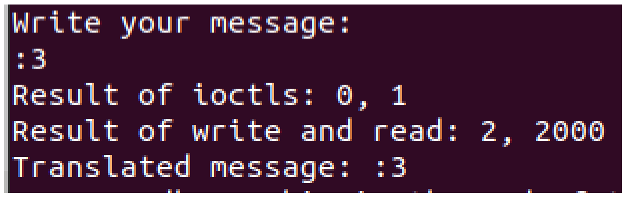


```
// Then, build the emote to replace the letter ...
emoteIdx = (msg[msgIdx] - 'A') + emoteStrt;
eyeIdx = emoteIdx / MOUTH_LENGTH;
mouthIdx = emoteIdx % EYES_LENGTH;
```

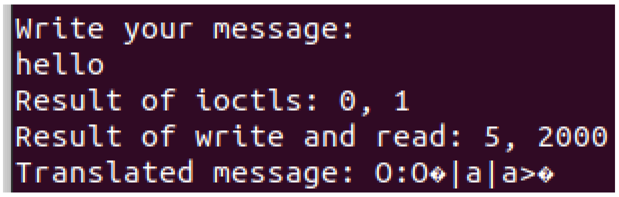
Note that the messages are not being encoded or decoded properly, and alongside that problem, the out-of-bounds errors only come up when trying to decode an input. Also something of not is that the read function always reads max, not the length of the translated message which is meant to be double of the original message.



```
Write your message:
Hello
Result of ioctls: 0, 1
Result of write and read: 5, 1000
Translated message: Hello
```

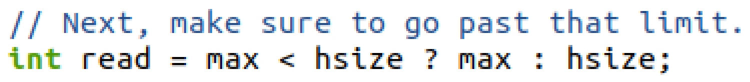


```
Write your message:
:3
Result of ioctls: 0, 1
Result of write and read: 2, 2000
Translated message: :3
```



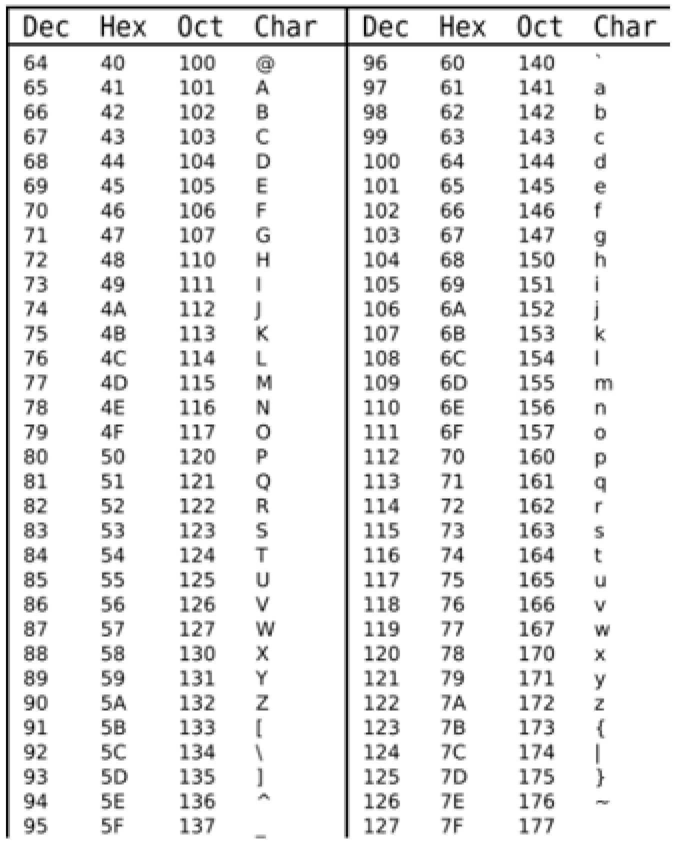
```
Write your message:
hello
Result of ioctls: 0, 1
Result of write and read: 5, 2000
Translated message: O:O�|a|a>�
```

It turns out that for the max size reading problem, I made it so that it read either the requested number of bytes or the max buffer size, so I had to make sure that only the relevant bytes of the translated message were being read. I got this to work by having an additional condition that sets the bytes to read as the size of the translated message. Now the read function does not give max size bytes any longers, but  the out-of-bounds behaviour is still there.



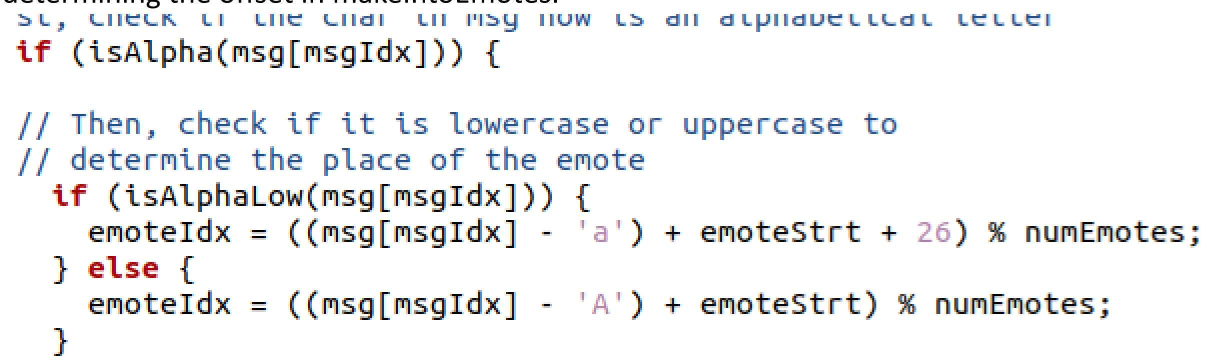
```
// Next, make sure to go past that limit.
int read = max < hsize ? max : hsize;
```

While dealing with the out-of-bounds index problem, I researched into the ASCCI values of the alphabetical letters and got a revelation that could help: The uppercase and lowercase letters are not actually in sequence (i.e. after 'Z', I expected to get 'a'), with various other characters to account for between them.



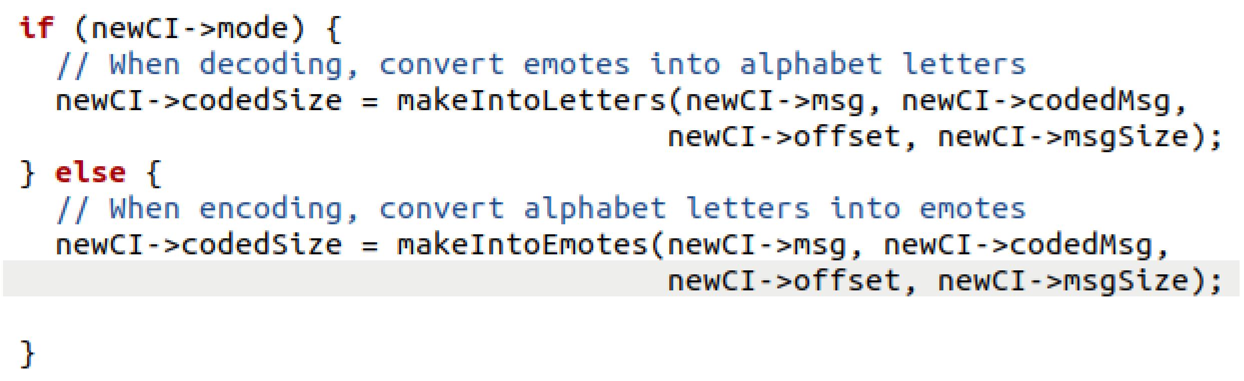
| Dec | Hex | Oct | Char | Dec | Hex | Oct | Char |
|---|---|---|---|---|---|---|---|
| 64 | 40 | 100 | @ | 96 | 60 | 140 | ` |
| 65 | 41 | 101 | A | 97 | 61 | 141 | a |
| 66 | 42 | 102 | B | 98 | 62 | 142 | b |
| 67 | 43 | 103 | C | 99 | 63 | 143 | c |
| 68 | 44 | 104 | D | 100 | 64 | 144 | d |
| 69 | 45 | 105 | E | 101 | 65 | 145 | e |
| 70 | 46 | 106 | F | 102 | 66 | 146 | f |
| 71 | 47 | 107 | G | 103 | 67 | 147 | g |
| 72 | 48 | 110 | H | 104 | 68 | 150 | h |
| 73 | 49 | 111 | I | 105 | 69 | 151 | i |
| 74 | 4A | 112 | J | 106 | 6A | 152 | j |
| 75 | 4B | 113 | K | 107 | 6B | 153 | k |
| 76 | 4C | 114 | L | 108 | 6C | 154 | l |
| 77 | 4D | 115 | M | 109 | 6D | 155 | m |
| 78 | 4E | 116 | N | 110 | 6E | 156 | n |
| 79 | 4F | 117 | O | 111 | 6F | 157 | o |
| 80 | 50 | 120 | P | 112 | 70 | 160 | p |
| 81 | 51 | 121 | Q | 113 | 71 | 161 | q |
| 82 | 52 | 122 | R | 114 | 72 | 162 | r |
| 83 | 53 | 123 | S | 115 | 73 | 163 | s |
| 84 | 54 | 124 | T | 116 | 74 | 164 | t |
| 85 | 55 | 125 | U | 117 | 75 | 165 | u |
| 86 | 56 | 126 | V | 118 | 76 | 166 | v |
| 87 | 57 | 127 | W | 119 | 77 | 167 | w |
| 88 | 58 | 130 | X | 120 | 78 | 170 | x |
| 89 | 59 | 131 | Y | 121 | 79 | 171 | y |
| 90 | 5A | 132 | Z | 122 | 7A | 172 | z |
| 91 | 5B | 133 | [ | 123 | 7B | 173 | { |
| 92 | 5C | 134 | \ | 124 | 7C | 174 | \| |
| 93 | 5D | 135 | ] | 125 | 7D | 175 | } |
| 94 | 5E | 136 | ^ | 126 | 7E | 176 | ~ |
| 95 | 5F | 137 | _ | 127 | 7F | 177 | |

Because of this revelation, I need to rework the way makeIntoLetter and makeIntoEmotes were dealing with alphabetical letter, by creating new methods that check for lowercase and uppercase alphabetical characters. Additionally, I added additional calculations that made sure that the right character were being assigned for lowercase alphabet letters, especially when determining the offset in makeIntoEmotes.



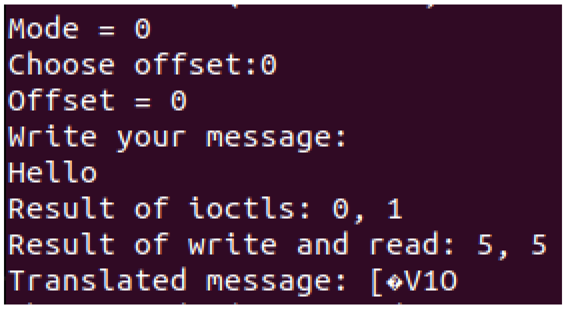
```
st, check if the char in msg now is an alphabetical letter
if (isAlpha(msg[msgIdx])) {

// Then, check if it is lowercase or uppercase to
// determine the place of the emote
  if (isAlphaLow(msg[msgIdx])) {
    emoteIdx = ((msg[msgIdx] - 'a') + emoteStrt + 26) % numEmotes;
  } else {
    emoteIdx = ((msg[msgIdx] - 'A') + emoteStrt) % numEmotes;
  }
```

Additionally, while looking at the problem with encoding mode not encoding, I realized that the condition inside write that chooses between makeIntoLetters and makeIntoEmotes, because converting letters into emotes is meant to happen when mode is 0 and vice versa when it is any non-zero number. Now, the unusual behaviour starts to make sense because of how emotes are only converted in decode, thus normal letters stay the same then.

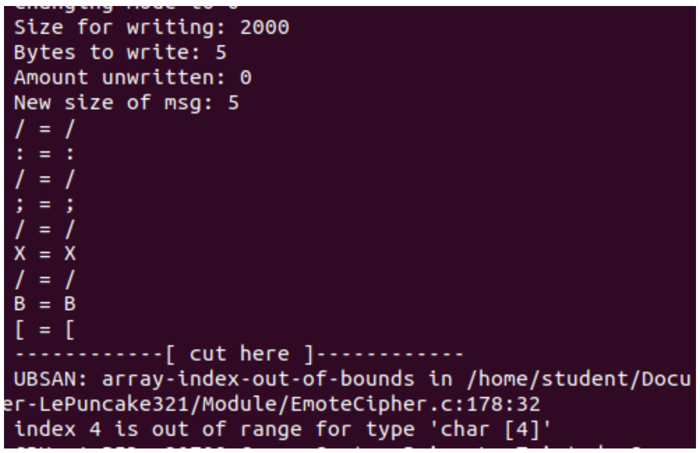


```
if (newCI->mode) {
  // When decoding, convert emotes into alphabet letters
  newCI->codedSize = makeIntoLetters(newCI->msg, newCI->codedMsg,
                                     newCI->offset, newCI->msgSize);
} else {
  // When encoding, convert alphabet letters into emotes
  newCI->codedSize = makeIntoEmotes(newCI->msg, newCI->codedMsg,
                                     newCI->offset, newCI->msgSize);

}
```

Now, looking at the code here still with the out-of-bounds error, I realize that the encoded message does not output the proper message, and the size of it is not doubled since all of the characters inputted are meant to be converted into emotes. I found out all of this by adding printk() statements inside makeIntoEmotes, but as for the emotes themselves, it appears that everything inside the translated message is correct.



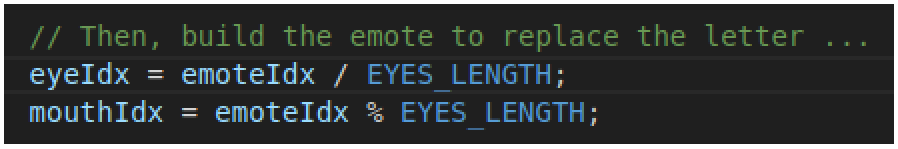
```
Mode = 0
Choose offset:0
Offset = 0
Write your message:
Hello
Result of ioctls: 0, 1
Result of write and read: 5, 5
Translated message: [�V1O
```



```
Size for writing: 2000
Bytes to write: 5
Amount unwritten: 0
New size of msg: 5
/ = /
: = :
/ = /
; = ;
/ = /
X = X
/ = /
B = B
[ = [
------------[ cut here ]------------
UBSAN: array-index-out-of-bounds in /home/student/Docu
er-LePuncake321/Module/EmoteCipher.c:178:32
index 4 is out of range for type 'char [4]'
```

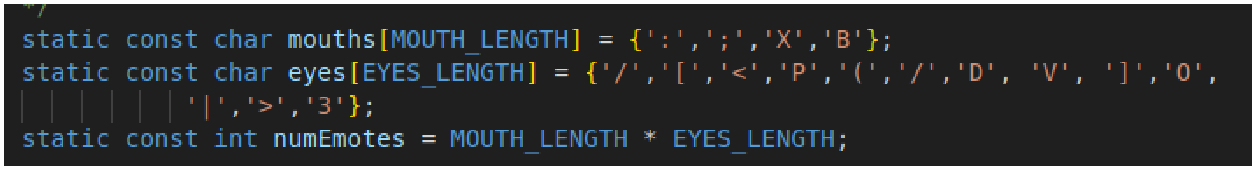
When checking the makeIntoEmotes method again, I realized that the index for the eye index was dividing by the length of the mouth, which would end up returning numbers larger than 3, which is the max index for accessing the eyes array.

What I ended up doing to solve this was replacing the division of MOUTH_LENGTH to EYES_LENGTH to prevent the result being greater than 3 and providing the proper index as well.



```
// Then, build the emote to replace the letter ...
eyeIdx = emoteIdx / EYES_LENGTH;
mouthIdx = emoteIdx % EYES_LENGTH;
```

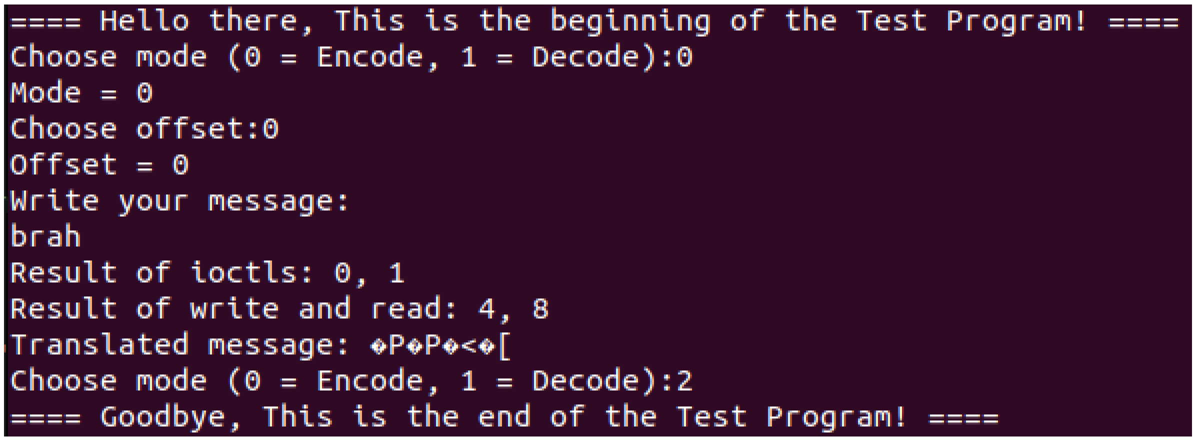
When all else failed, I ended up finally looking at the arrays I set for the mouths and eyes and it turned out that the array names as well as the array sizes were swapped accidentally. This was a core cause of the out-of-bounds errors.



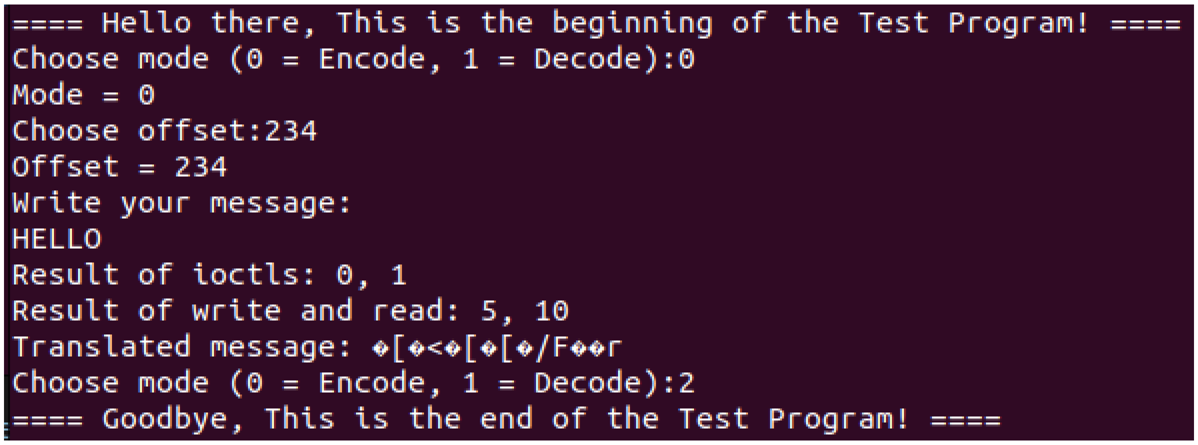
```
static const char mouths[MOUTH_LENGTH] = {':',';','X','B'};
static const char eyes[EYES_LENGTH] = {'/','[','<','P','(','/','D', 'V', ']','O',
                '|','>','3'};
static const int numEmotes = MOUTH_LENGTH * EYES_LENGTH;
```

And so, I fixed the errors by making sure the variable name eyes was associated with the size 4 array and the variable name mouths was associated with the size 13 array.

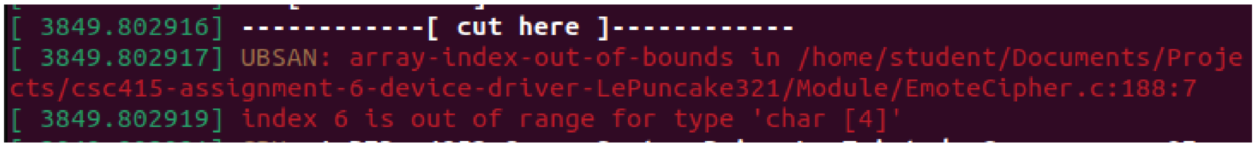
Finally, after solving the out-of-bounds errors, I could look into conversion methods. Currently, the encoding of lowercase letters is giving unintended results, as well as when the offset is large. In addition to this, there were still some out-of-bounds errors popping up, but this time they came from a different locations in the code.



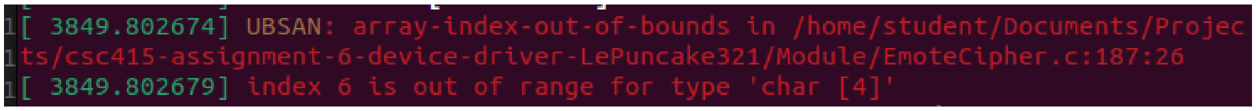
```
==== Hello there, This is the beginning of the Test Program! ====
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:0
Offset = 0
Write your message:
brah
Result of ioctls: 0, 1
Result of write and read: 4, 8
Translated message: �P�P�<�[
Choose mode (0 = Encode, 1 = Decode):2
==== Goodbye, This is the end of the Test Program! ====
```



```
==== Hello there, This is the beginning of the Test Program! ====
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:234
Offset = 234
Write your message:
HELLO
Result of ioctls: 0, 1
Result of write and read: 5, 10
Translated message: �[�<�[�[�/F��r
Choose mode (0 = Encode, 1 = Decode):2
==== Goodbye, This is the end of the Test Program! ====
```



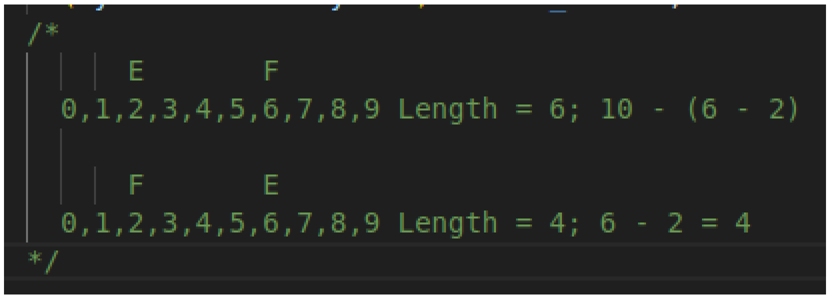
```
[ 3849.802916] ------------[ cut here ]------------
[ 3849.802917] UBSAN: array-index-out-of-bounds in /home/student/Documents/Proje
cts/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:188:7
[ 3849.802919] index 6 is out of range for type 'char [4]'
```



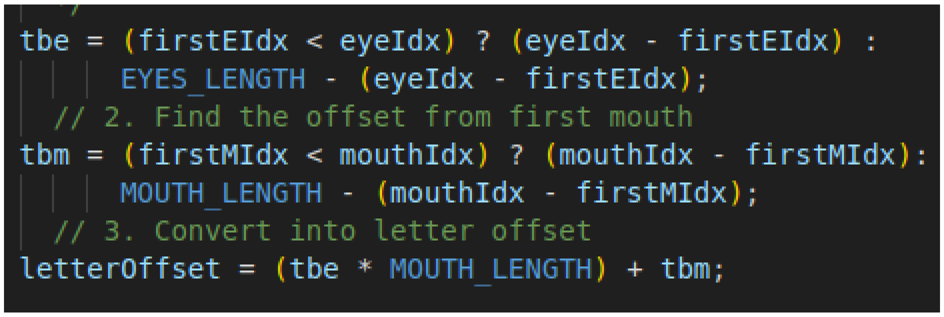
```
[ 3849.802674] UBSAN: array-index-out-of-bounds in /home/student/Documents/Projec
ts/csc415-assignment-6-device-driver-LePuncake321/Module/EmoteCipher.c:187:26
[ 3849.802679] index 6 is out of range for type 'char [4]'
```

It turns out that I just forgot to save and make the changes that I ended up doing beforehand, and these problems were gone by then.

While reworking the makeIntoLetters method, I was having difficulty figuring out how to convert the indexes of the current emote into the associated letters, especially with the addition of the offset into this calculation.



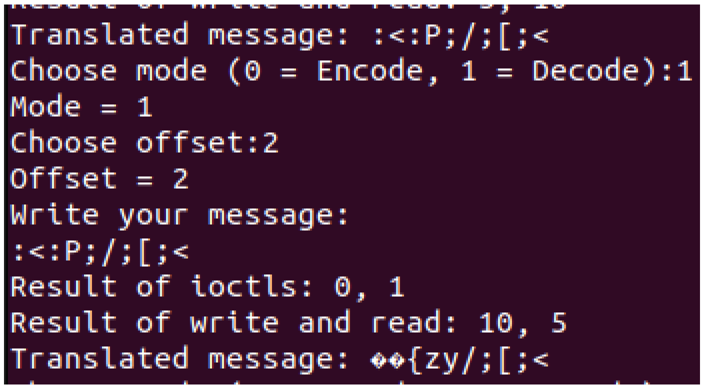
```
/*
      E       F
  0,1,2,3,4,5,6,7,8,9 Length = 6; 10 - (6 - 2)

      F       E
  0,1,2,3,4,5,6,7,8,9 Length = 4; 6 - 2 = 4
*/
```



```
tbe = (firstEIdx < eyeIdx) ? (eyeIdx - firstEIdx) :
        EYES_LENGTH - (eyeIdx - firstEIdx);
  // 2. Find the offset from first mouth
tbm = (firstMIdx < mouthIdx) ? (mouthIdx - firstMIdx):
        MOUTH_LENGTH - (mouthIdx - firstMIdx);
  // 3. Convert into letter offset
letterOffset = (tbe * MOUTH_LENGTH) + tbm;
```

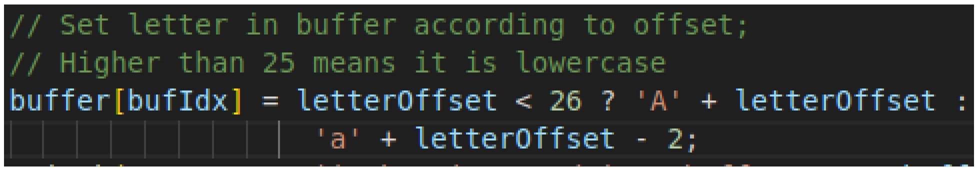
I realized that I could simplify the process by converting the offset integer into two indexes representing the emote associated with 'A'. Additionally, I had to put up my plan somewhere to visualize the sub-problem, and it came to me then the full solution. I could use the first indexes and the current emote's indexes to calculate the relative position of the current emote to the first emote. I also had to account for when either the current or first index was behind the other index. Finding out the relative index allowed me to finally find the proper letter to an emote, even with the offset

Now that the makeIntoLetter was implemented, I checked the test program again and there are finally no more out-of-bounds errors, but for some reason, the decode mode of the device driver was returning the wrong values.



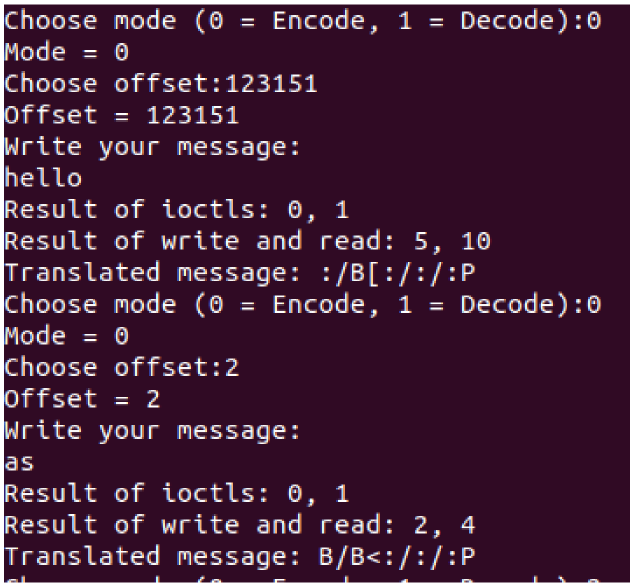
```
Translated message: :<:P;/;[;<
Choose mode (0 = Encode, 1 = Decode):1
Mode = 1
Choose offset:2
Offset = 2
Write your message:
:<:P;/;[;<
Result of ioctls: 0, 1
Result of write and read: 10, 5
Translated message: ��{zy/;[;<
```

It turns out that the condition to handle lowercase letters was formatted wrong, as the calculation below was meant to use 26, not 2. I rectified this pretty quickly.



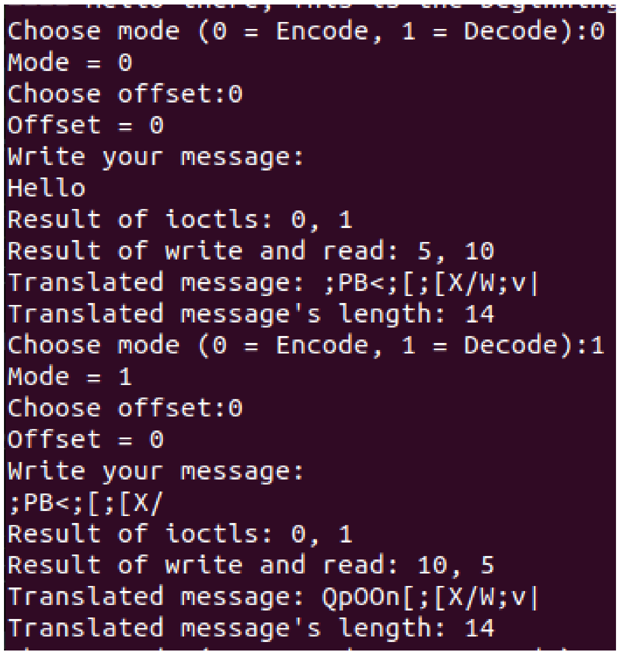
```
// Set letter in buffer according to offset;
// Higher than 25 means it is lowercase
buffer[bufIdx] = letterOffset < 26 ? 'A' + letterOffset :
                 'a' + letterOffset - 2;
```

Now, when the device driver's read and write are properly implemented (hopefully), I checked the Test Program again, and it appears that the output is not consistent with every loop of the program. There is a clue based on how previous emotes persist if size of next message is smaller

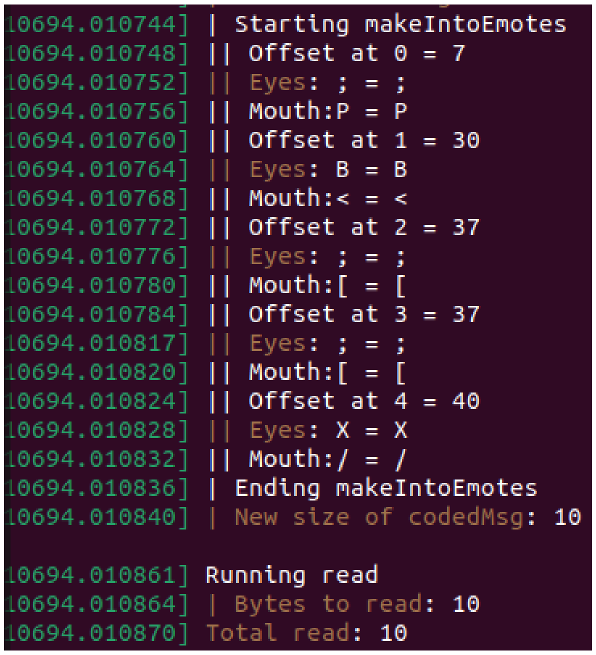


```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:123151
Offset = 123151
Write your message:
hello
Result of ioctls: 0, 1
Result of write and read: 5, 10
Translated message: :/B[:/:/:P
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:2
Offset = 2
Write your message:
as
Result of ioctls: 0, 1
Result of write and read: 2, 4
Translated message: B/B<:/:/:P
```

Additionally, the test program is also returning some excess data that was never inputted in the first place. It seems so strange because it only goes away after running make clean and make again.

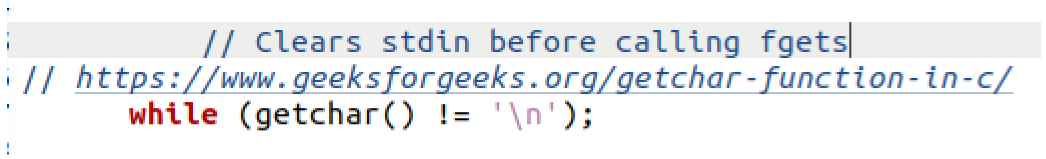


```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:0
Offset = 0
Write your message:
Hello
Result of ioctls: 0, 1
Result of write and read: 5, 10
Translated message: ;PB<;[;[X/W;v|
Translated message's length: 14
Choose mode (0 = Encode, 1 = Decode):1
Mode = 1
Choose offset:0
Offset = 0
Write your message:
;PB<;[;[X/
Result of ioctls: 0, 1
Result of write and read: 10, 5
Translated message: QpOOn[;[X/W;v|
Translated message's length: 14
```



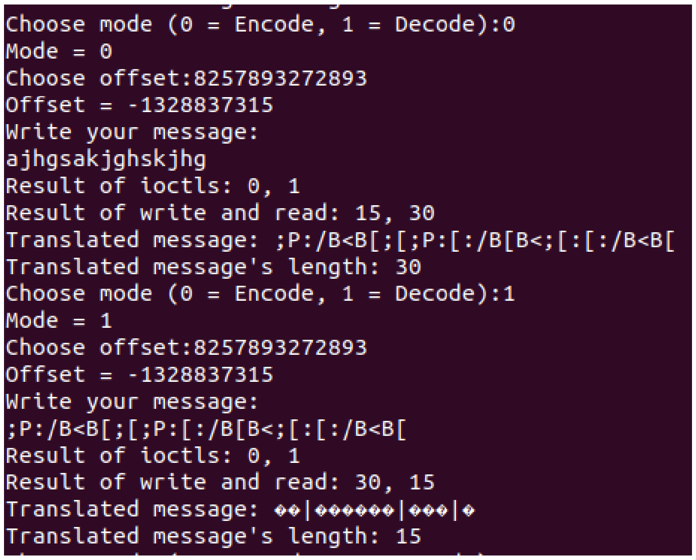
```
10694.010744] | Starting makeIntoEmotes
10694.010748] || Offset at 0 = 7
10694.010752] || Eyes: ; = ;
10694.010756] || Mouth:P = P
10694.010760] || Offset at 1 = 30
10694.010764] || Eyes: B = B
10694.010768] || Mouth:< = <
10694.010772] || Offset at 2 = 37
10694.010776] || Eyes: ; = ;
10694.010780] || Mouth:[ = [
10694.010784] || Offset at 3 = 37
10694.010817] || Eyes: ; = ;
10694.010820] || Mouth:[ = [
10694.010824] || Offset at 4 = 40
10694.010828] || Eyes: X = X
10694.010832] || Mouth:/ = /
10694.010836] | Ending makeIntoEmotes
10694.010840] | New size of codedMsg: 10

10694.010861] Running read
10694.010864] | Bytes to read: 10
10694.010870] Total read: 10
```

The solution to both these problems is to clear the standard input buffer before any input. TO be honest, I have no idea what causes the random bytes of data getting into the input without any clear cause, but otherwise the original problem has been solved



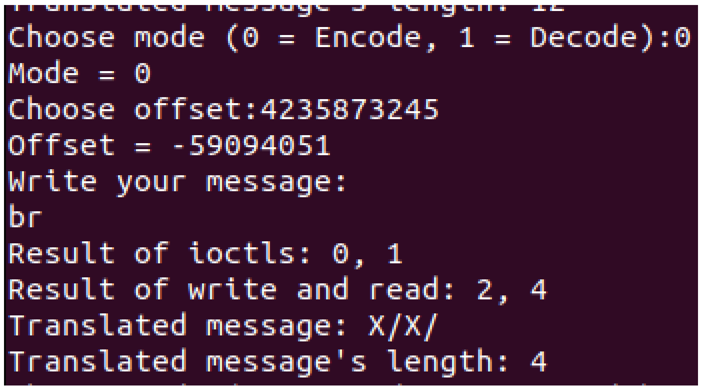
```
            // Clears stdin before calling fgets
// https://www.geeksforgeeks.org/getchar-function-in-c/
        while (getchar() != '\n');
```

Now, with that out of the way, it appears that decoding is still not working properly as while trying to decode with a large negative offset, it produces strange results.



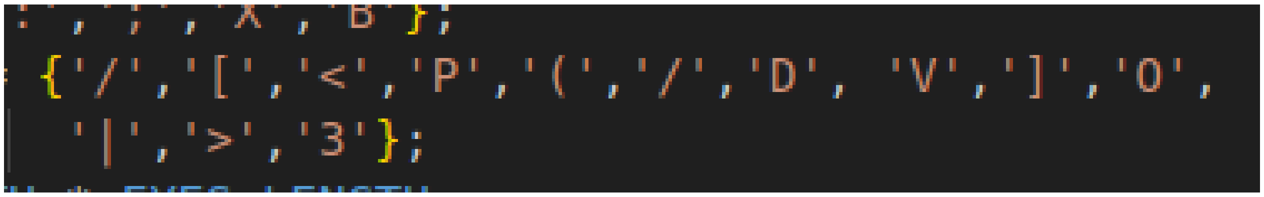
```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:8257893272893
Offset = -1328837315
Write your message:
ajhgsakjghskjhg
Result of ioctls: 0, 1
Result of write and read: 15, 30
Translated message: ;P:/B<B[;[;P:[:/B[B<;[:[:/B<B[
Translated message's length: 30
Choose mode (0 = Encode, 1 = Decode):1
Mode = 1
Choose offset:8257893272893
Offset = -1328837315
Write your message:
;P:/B<B[;[;P:[:/B[B<;[:[:/B<B[
Result of ioctls: 0, 1
Result of write and read: 30, 15
Translated message: ��|������|���|�
Translated message's length: 15
```

I also inadvertently found out during the search for the solution to the previous problem that two characters can end up being associated with the face.

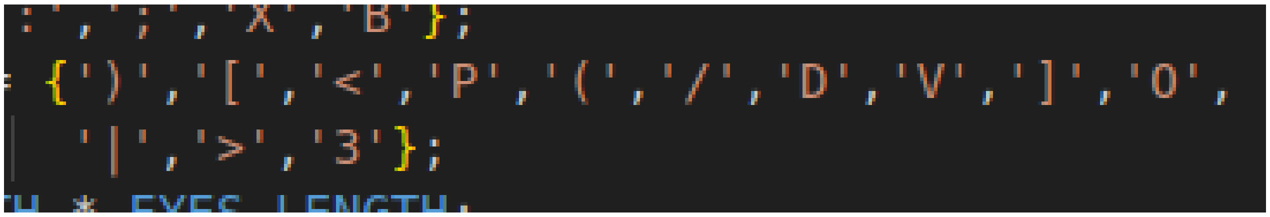


```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:4235873245
Offset = -59094051
Write your message:
br
Result of ioctls: 0, 1
Result of write and read: 2, 4
Translated message: X/X/
Translated message's length: 4
```

It turns out that in the mouths arrays, there was a duplicate Turns out that there is a duplicate '/' character, so I ended up replacing it with ')', which was missing from the array.

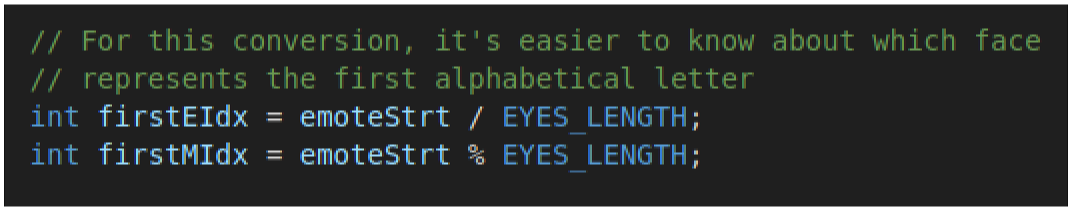


```
{'/','[','<','P','(','/','D', 'V',']','O',
 '|','>','3'};
```

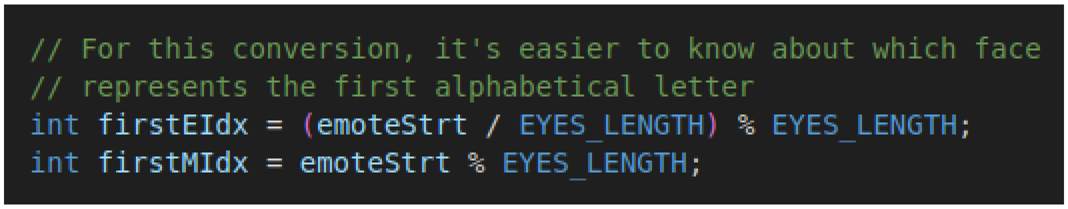


```
{')','[','<','P','(','/','D','V',']','O',
 '|','>','3'};
```

Continuing onto the larger problem, I found out that I made the same mistake as before when it came to calculating the mouth and eyes index of the first emote with the offset. I needed to put a modulus to prevent the first eye index from going past 3, preventing any out-of-bounds error.

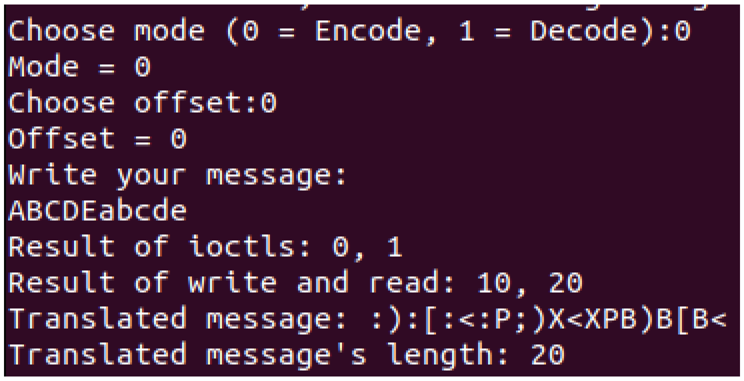


```
// For this conversion, it's easier to know about which face
// represents the first alphabetical letter
int firstEIdx = emoteStrt / EYES_LENGTH;
int firstMIdx = emoteStrt % EYES_LENGTH;
```

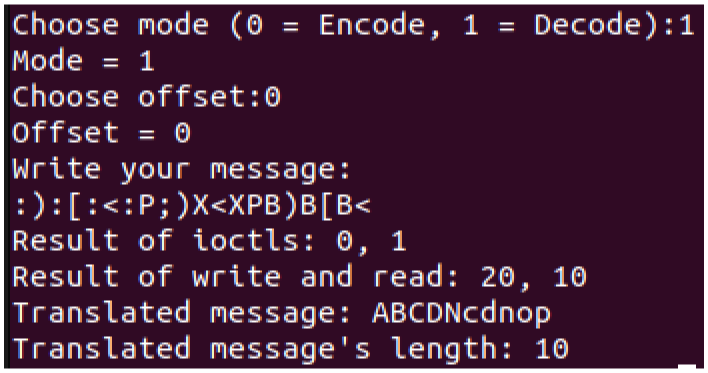


```
// For this conversion, it's easier to know about which face
// represents the first alphabetical letter
int firstEIdx = (emoteStrt / EYES_LENGTH) % EYES_LENGTH;
int firstMIdx = emoteStrt % EYES_LENGTH;
```

Looks like whatever problem I had is still not fixed with the inconsistency of decoding. However, note that strange behavior has some consistency now: It happens after the first four letters and always does a sequence of letters after four letters, meaning that there was something wrong with the calculation of emote being built.

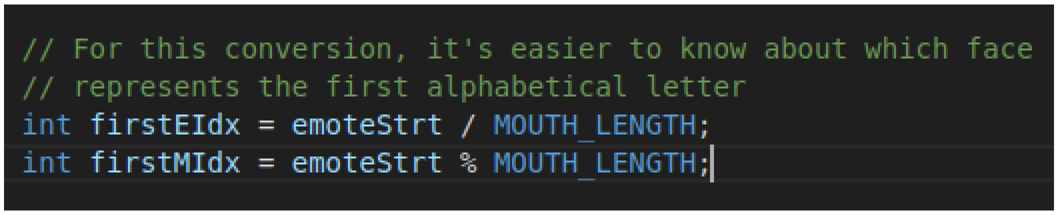


```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:0
Offset = 0
Write your message:
ABCDEabcde
Result of ioctls: 0, 1
Result of write and read: 10, 20
Translated message: :):[:<:P;)X<XPB)B[B<
Translated message's length: 20
```



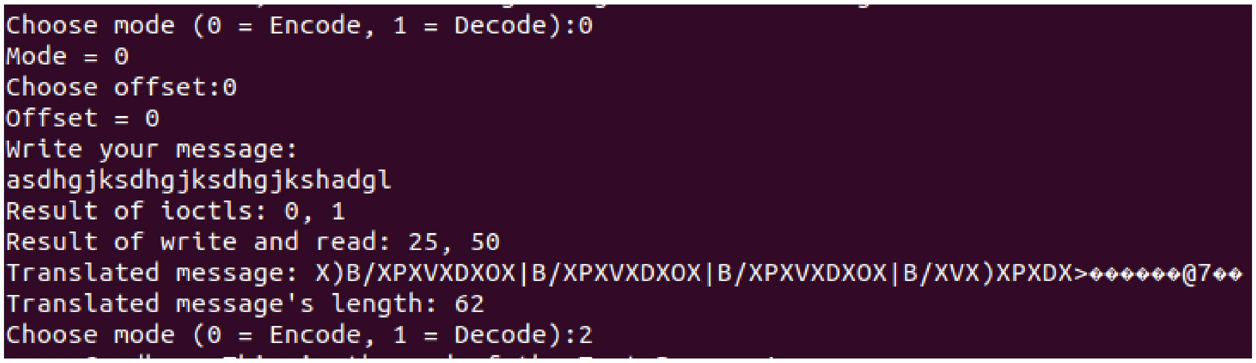
```
Choose mode (0 = Encode, 1 = Decode):1
Mode = 1
Choose offset:0
Offset = 0
Write your message:
:):[:<:P;)X<XPB)B[B<
Result of ioctls: 0, 1
Result of write and read: 20, 10
Translated message: ABCDNcdnop
Translated message's length: 10
```

I hadn't realized that the calculation for the first eye index is not the same as makeIntoEmotes manner of calculating the eye index for the emote being built in it. What I am meant to be doing was dividing the offset by the length of mouth (13). Additionally, the first mouth index calculator had the same problem of being the wrong calculation to EYES_LENGTH instead of MOUTH_LENGTH.



```
// For this conversion, it's easier to know about which face
// represents the first alphabetical letter
int firstEIdx = emoteStrt / MOUTH_LENGTH;
int firstMIdx = emoteStrt % MOUTH_LENGTH;
```

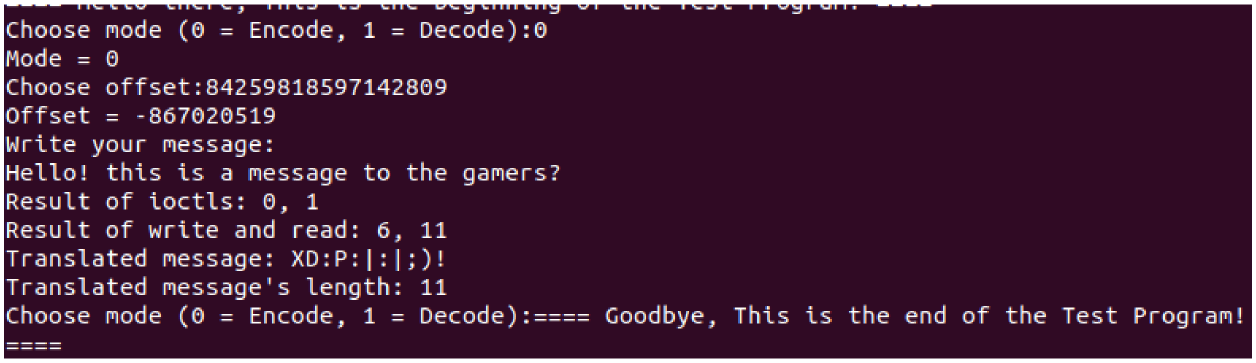
While testing again, I keep getting the same strange behaviour of additional character in output despite all of device drivers working properly.



```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:0
Offset = 0
Write your message:
asdhgjksdhgjksdhgjkshadgl
Result of ioctls: 0, 1
Result of write and read: 25, 50
Translated message: X)B/XPXVXDXOX|B/XPXVXDXOX|B/XPXVXDXOX|B/XVX)XPXDX>������@7��
Translated message's length: 62
Choose mode (0 = Encode, 1 = Decode):2
```

I just moved the flushing operation for the output at the beginning of every loop through the program so that it is guaranteed to not be filled from the start. Again, I still have no idea why it is happening in the first place.

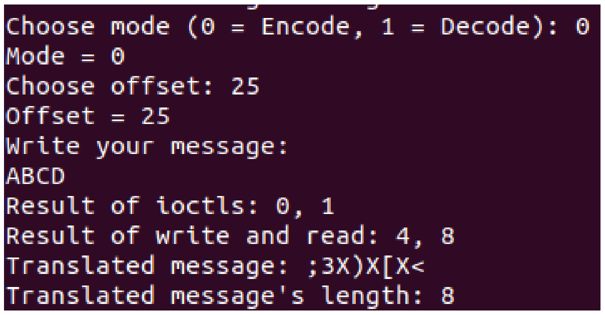
Further checking the test program, I realized that the program is unable to take into strings with spaces in between. This is based on the limitations of scanf() when it takes in char * types.



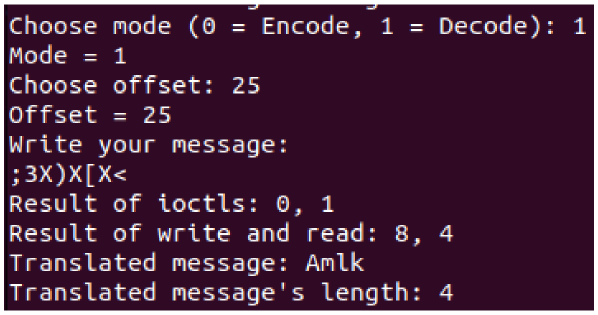
```
Choose mode (0 = Encode, 1 = Decode):0
Mode = 0
Choose offset:84259818597142809
Offset = -867020519
Write your message:
Hello! this is a message to the gamers?
Result of ioctls: 0, 1
Result of write and read: 6, 11
Translated message: XD:P:|:|;)!
Translated message's length: 11
Choose mode (0 = Encode, 1 = Decode):==== Goodbye, This is the end of the Test Program!
====
```

What I needed to use instead of scanf() was the fgets() method so that the program can take in a line of characters without stopping at a space character. With the use of fgets(), I now needed to rework the program to make it work properly: the scanf() before the fgets() skips the string input so I needed to use a getchar() method to prevent this, and I needed to use strcspn() to make sure that the nextline character from fgets() does not get passed into the write() method for the device driver.

Finally, while testing a sizable offset, there seems to be a problem with translating again where some emotes don't translate properly into the correct letter. What stands out is how the first letter is translated properly, but not the other ones. This means that the problem lies in either the conversion of emotes to letters, since the conversion of letters to emotes is fine.

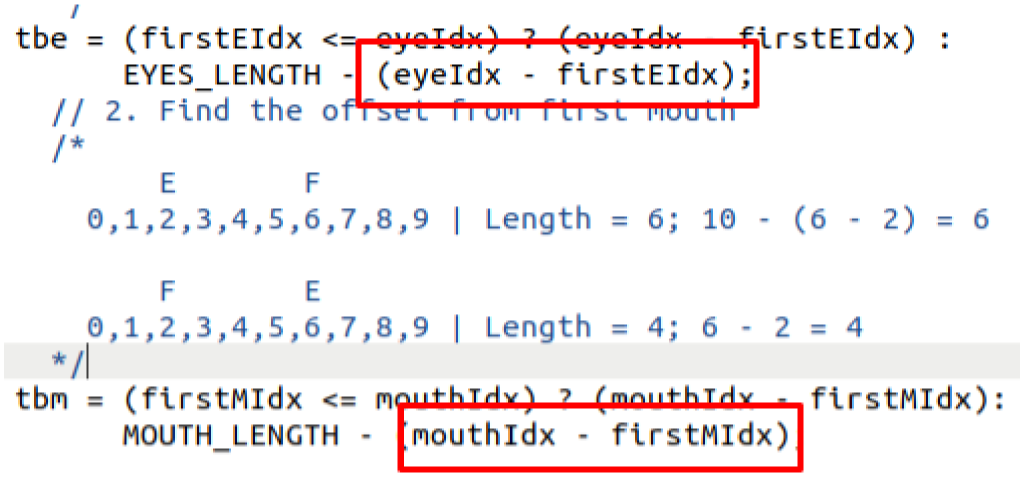


```
Choose mode (0 = Encode, 1 = Decode): 0
Mode = 0
Choose offset: 25
Offset = 25
Write your message:
ABCD
Result of ioctls: 0, 1
Result of write and read: 4, 8
Translated message: ;3X)X[X<
Translated message's length: 8
```



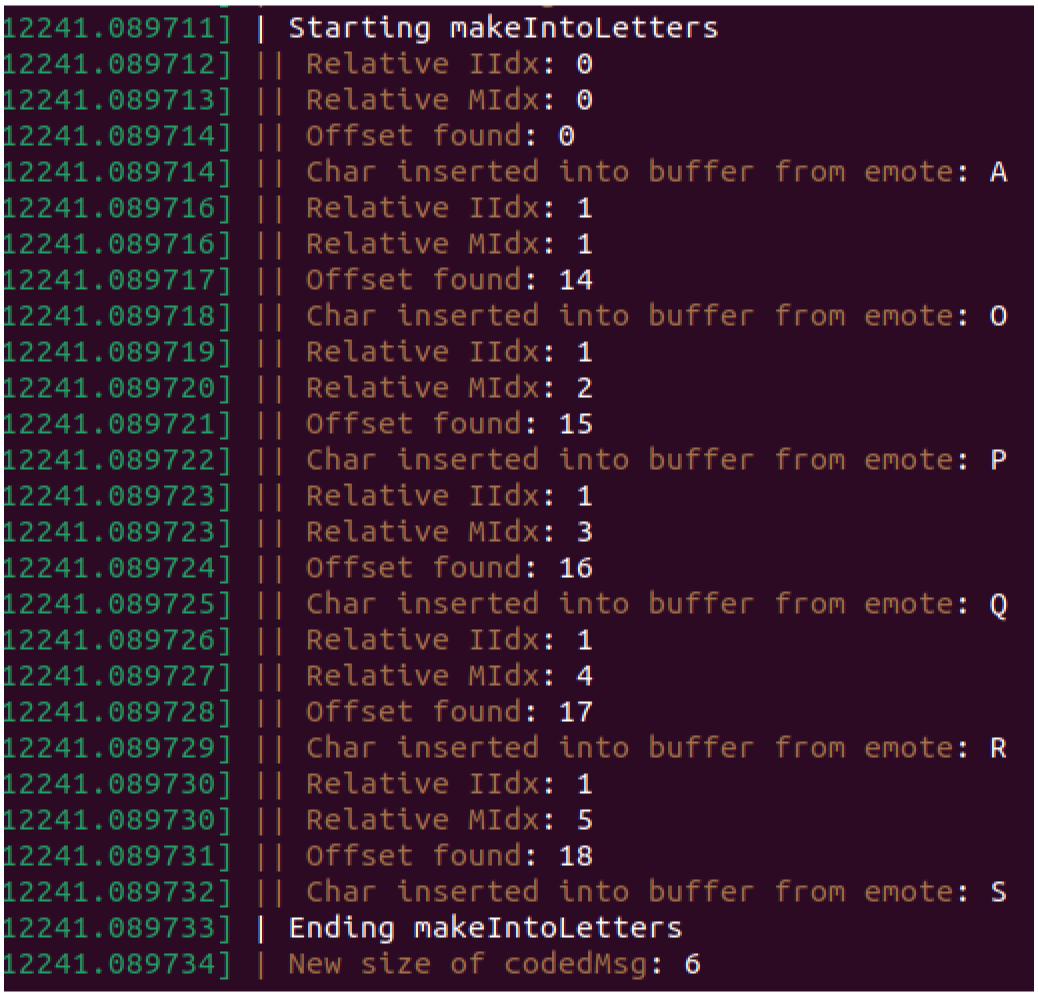
```
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
;3X)X[X<
Result of ioctls: 0, 1
Result of write and read: 8, 4
Translated message: Amlk
Translated message's length: 4
```

it turns out I made a mistake on calculating the index when the current index of the eyes/mouth is less than the first one, based on the confusion of the examples I had, just had to swap first and curr index in that event. I just had to swap the index being calculated in the subtractions relating to the case of the first index being greater than the current index.



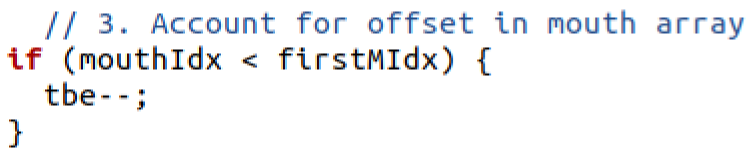
```
tbe = (firstEIdx <= eyeIdx) ? (eyeIdx - firstEIdx) :
        EYES_LENGTH - (eyeIdx - firstEIdx);
   // 2. Find the offset from first mouth
   /*
          E       F
     0,1,2,3,4,5,6,7,8,9 | Length = 6; 10 - (6 - 2) = 6

          F       E
     0,1,2,3,4,5,6,7,8,9 | Length = 4; 6 - 2 = 4
   */
tbm = (firstMIdx <= mouthIdx) ? (mouthIdx - firstMIdx):
        MOUTH_LENGTH - (mouthIdx - firstMIdx)
```

This was not enough since the strange behavior was still happening, so I ended up using printk() statements to check the internal calculations. It turns out that the calculation of the relative eye index was inadvertently skipping some emotes if the first mouth index with the offset was not at the first mouth of the entire array. Essentially, when starting at an emote other than the first mouth, 13 - (first mouth index) emotes will be advanced in eye index by 1 without something to account for that.



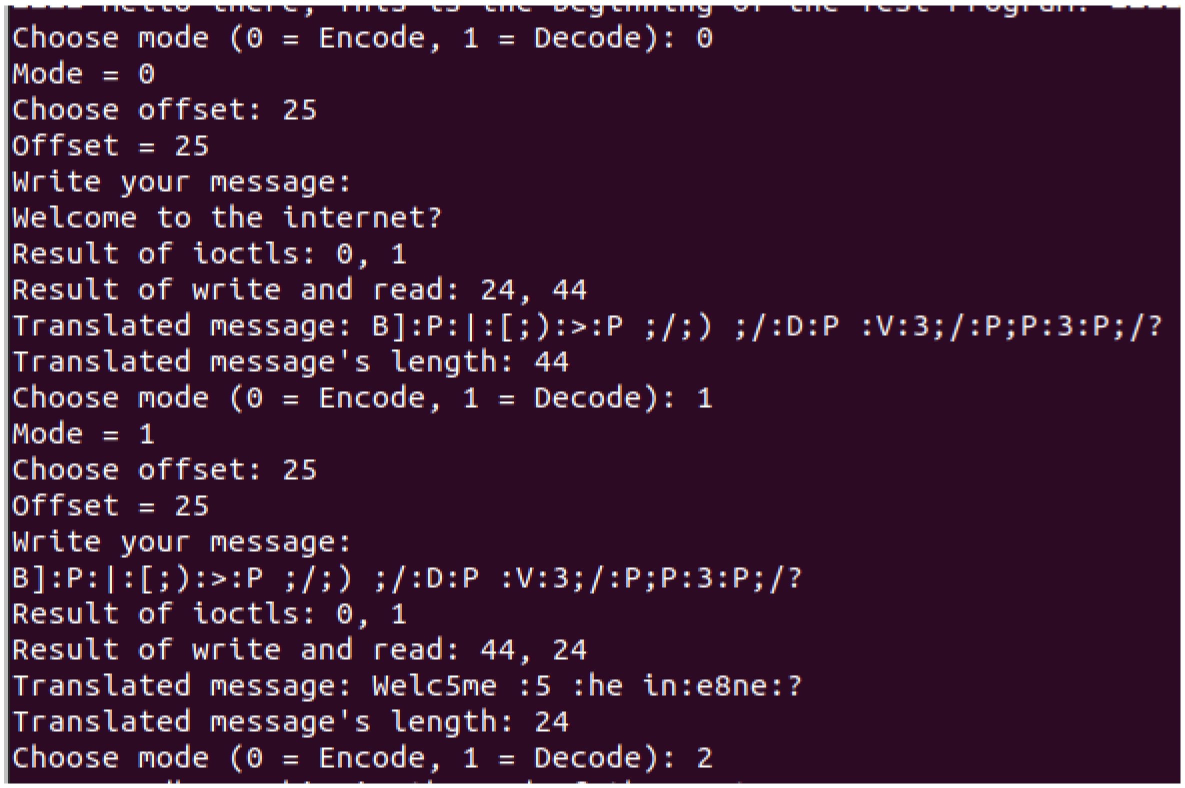
```
12241.089711] | Starting makeIntoLetters
12241.089712] || Relative IIdx: 0
12241.089713] || Relative MIdx: 0
12241.089714] || Offset found: 0
12241.089714] || Char inserted into buffer from emote: A
12241.089716] || Relative IIdx: 1
12241.089716] || Relative MIdx: 1
12241.089717] || Offset found: 14
12241.089718] || Char inserted into buffer from emote: O
12241.089719] || Relative IIdx: 1
12241.089720] || Relative MIdx: 2
12241.089721] || Offset found: 15
12241.089722] || Char inserted into buffer from emote: P
12241.089723] || Relative IIdx: 1
12241.089723] || Relative MIdx: 3
12241.089724] || Offset found: 16
12241.089725] || Char inserted into buffer from emote: Q
12241.089726] || Relative IIdx: 1
12241.089727] || Relative MIdx: 4
12241.089728] || Offset found: 17
12241.089729] || Char inserted into buffer from emote: R
12241.089730] || Relative IIdx: 1
12241.089730] || Relative MIdx: 5
12241.089731] || Offset found: 18
12241.089732] || Char inserted into buffer from emote: S
12241.089733] | Ending makeIntoLetters
12241.089734] | New size of codedMsg: 6
```

To fix this, I added an additional condition that decrements the relative eye index when the current mouth index of the emote being looked at is lower than the first mouth index, before calculating the letter to be added.

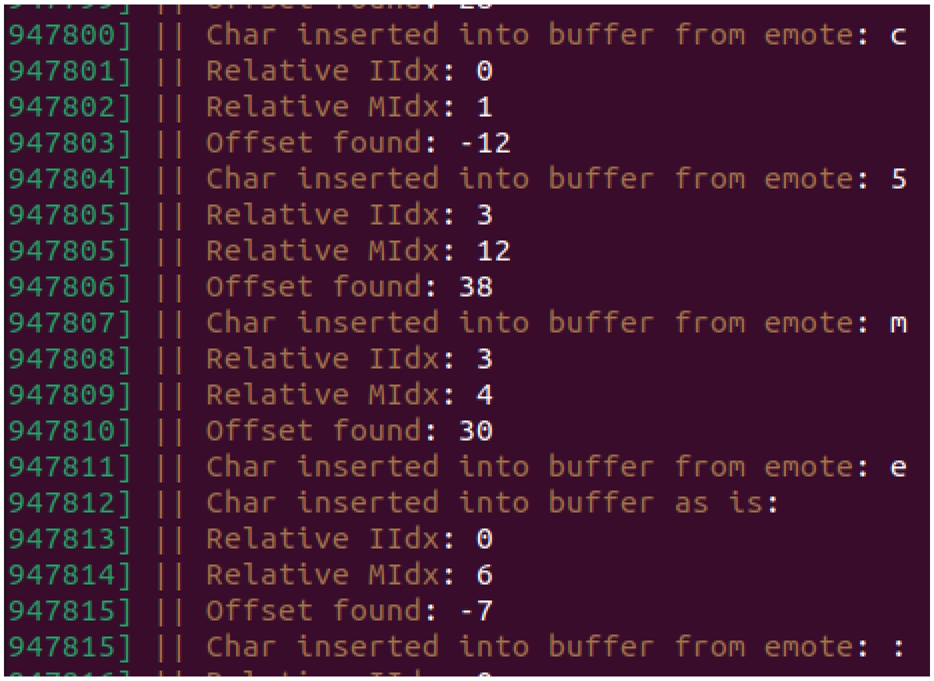


```
    // 3. Account for offset in mouth array
  if (mouthIdx < firstMIdx) {
    tbe--;
  }
```

After the fix above, there is something left wrong where most letters are being translated properly, but few are very wrong. It seems that some offsets are being set as negative, meaning that my solution needs a bit more tweaking. I noticed that most of these negative offsets only appear when the relative eye index is 0.

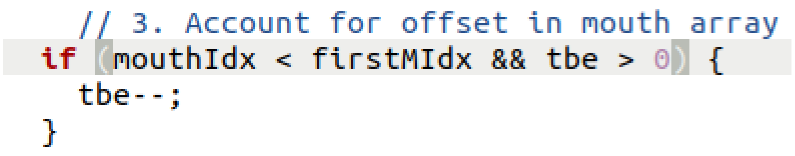


```
Choose mode (0 = Encode, 1 = Decode): 0
Mode = 0
Choose offset: 25
Offset = 25
Write your message:
Welcome to the internet?
Result of ioctls: 0, 1
Result of write and read: 24, 44
Translated message: B]:P:|:[;):>:P ;/;) ;/:D:P :V:3;/:P;P:3:P;/?
Translated message's length: 44
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
B]:P:|:[;):>:P ;/;) ;/:D:P :V:3;/:P;P:3:P;/?
Result of ioctls: 0, 1
Result of write and read: 44, 24
Translated message: Welc5me :5 :he in:e8ne:?
Translated message's length: 24
Choose mode (0 = Encode, 1 = Decode): 2
```



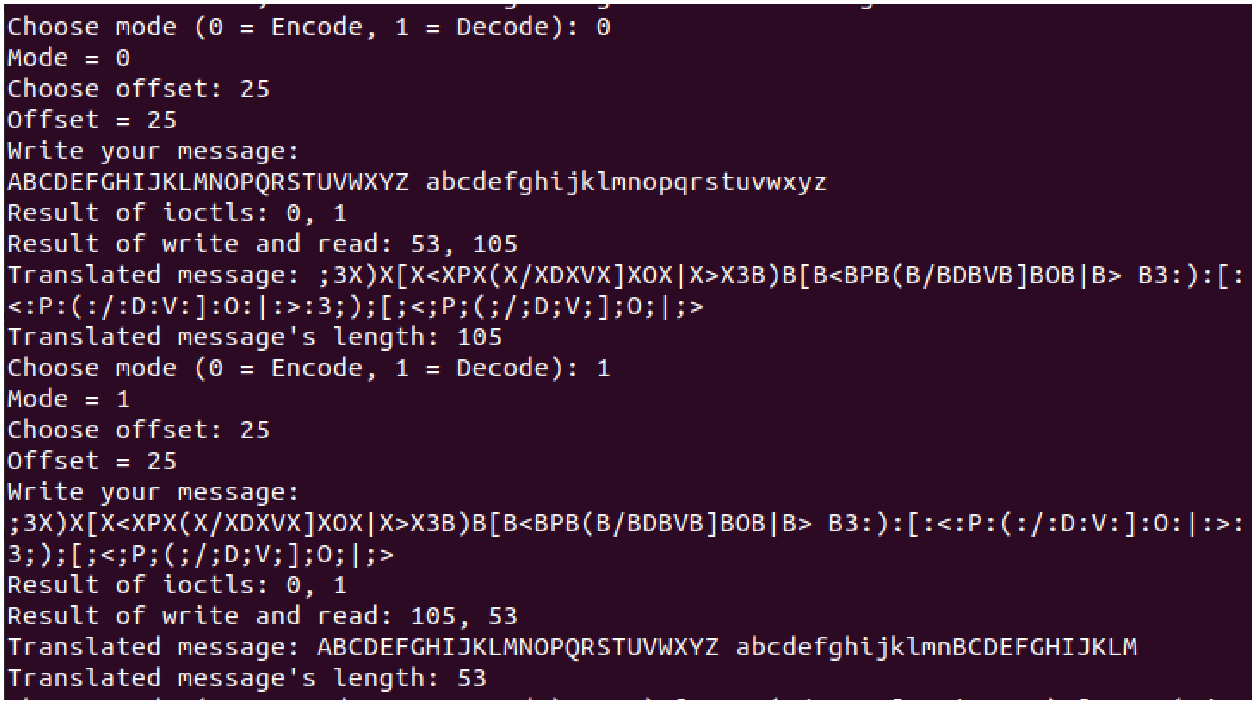
```
947800] || Char inserted into buffer from emote: c
947801] || Relative IIdx: 0
947802] || Relative MIdx: 1
947803] || Offset found: -12
947804] || Char inserted into buffer from emote: 5
947805] || Relative IIdx: 3
947805] || Relative MIdx: 12
947806] || Offset found: 38
947807] || Char inserted into buffer from emote: m
947808] || Relative IIdx: 3
947809] || Relative MIdx: 4
947810] || Offset found: 30
947811] || Char inserted into buffer from emote: e
947812] || Char inserted into buffer as is:
947813] || Relative IIdx: 0
947814] || Relative MIdx: 6
947815] || Offset found: -7
947815] || Char inserted into buffer from emote: :
```

To prevent the inclusion of negative numbers in calculating the offset, I added another condition where if the relative eye index was already 0, don't decrement at all.



```
    // 3. Account for offset in mouth array
  if (mouthIdx < firstMIdx && tbe > 0) {
    tbe--;
  }
```

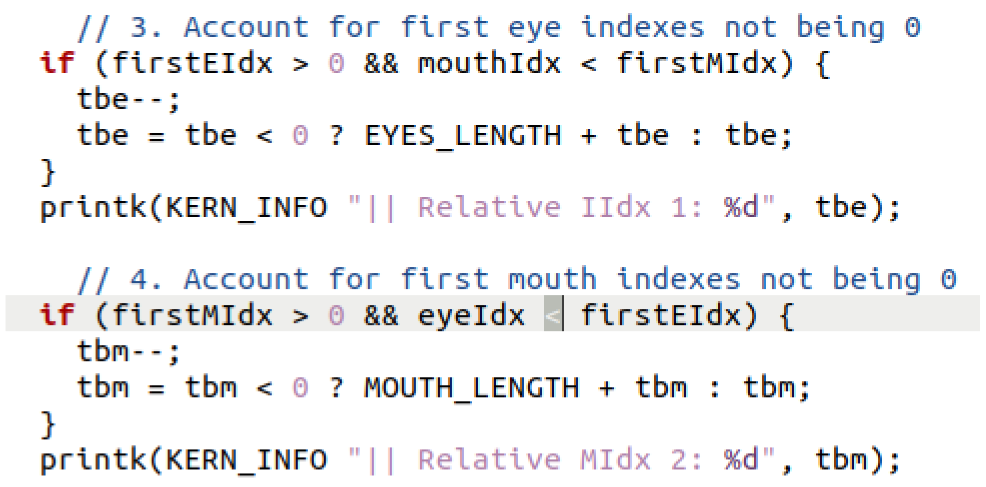
This didn't actually solve the problem.



```
Choose mode (0 = Encode, 1 = Decode): 0
Mode = 0
Choose offset: 25
Offset = 25
Write your message:
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Result of ioctls: 0, 1
Result of write and read: 53, 105
Translated message: ;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|B> B3:):[:
<:P:(:/:D:V:]:O:|:>:3;);[;<;P;(;/;D;V;];O;|;>
Translated message's length: 105
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|B> B3:):[:<:P:(:/:D:V:]:O:|:>:
3;);[;<;P;(;/;D;V;];O;|;>
Result of ioctls: 0, 1
Result of write and read: 105, 53
Translated message: ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnBCDEFGHIJKLM
Translated message's length: 53
```

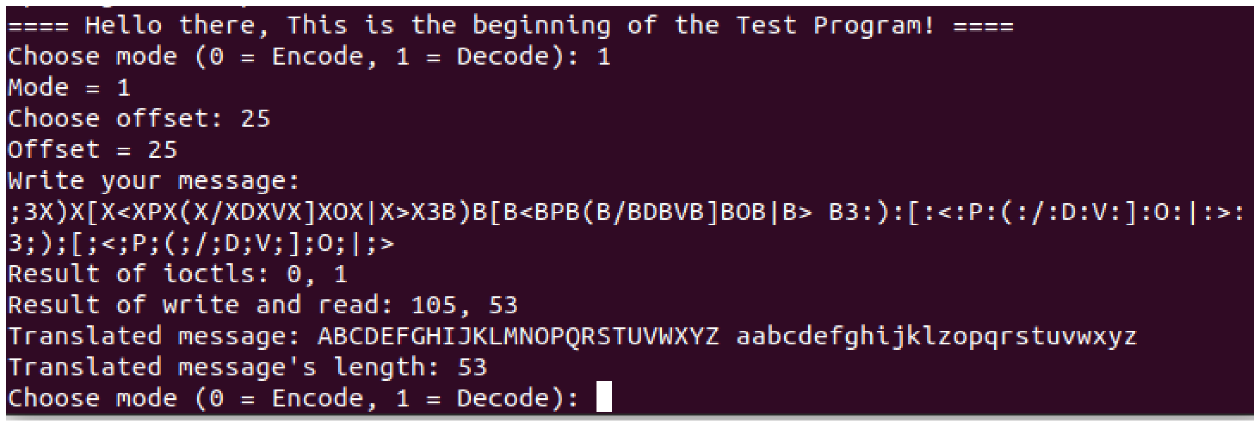
It turns out there is something weird going on with how the offset is being calculated.

I ended up figuring out a better solution, because I did not account for when the eye index is also having the same issue. Additionally, I realized that this problem only appears when the first eyes and mouth index are not equal to 0. So I managed to put down some conditions that follow this logic where any indexes before the first indexes will subtract their respective indexes, like so below



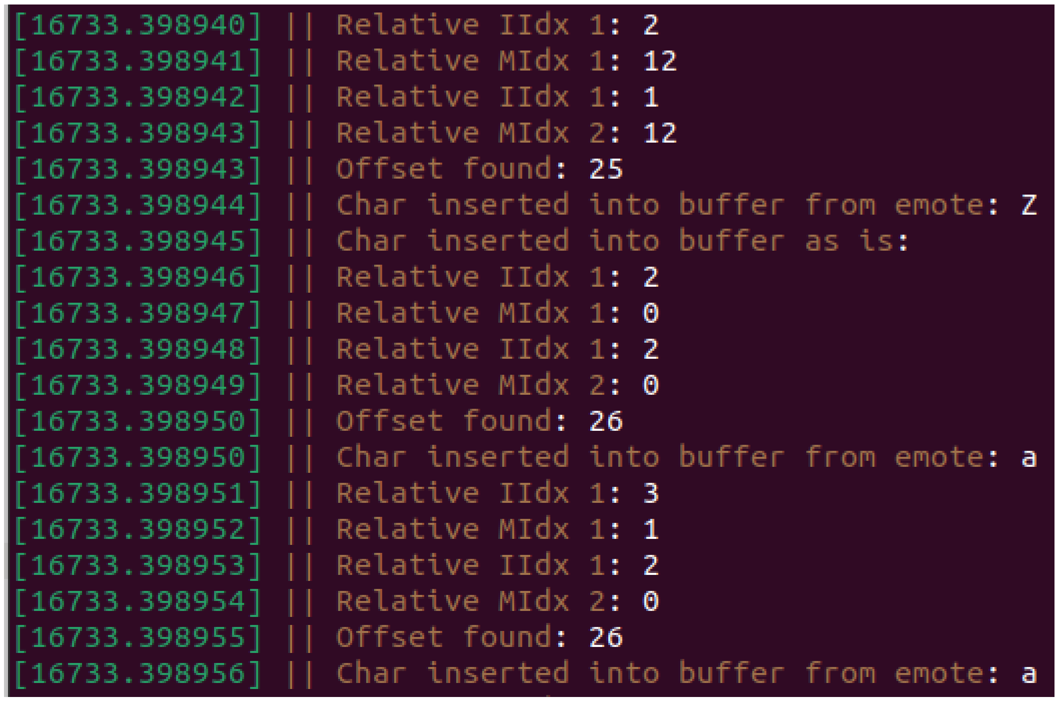
```
  // 3. Account for first eye indexes not being 0
if (firstEIdx > 0 && mouthIdx < firstMIdx) {
  tbe--;
  tbe = tbe < 0 ? EYES_LENGTH + tbe : tbe;
}
printk(KERN_INFO "|| Relative IIdx 1: %d", tbe);

  // 4. Account for first mouth indexes not being 0
if (firstMIdx > 0 && eyeIdx < firstEIdx) {
  tbm--;
  tbm = tbm < 0 ? MOUTH_LENGTH + tbm : tbm;
}
printk(KERN_INFO "|| Relative MIdx 2: %d", tbm);
```

The decoding aspect of the program is almost fixed, it appears that there are only two characters that are off.



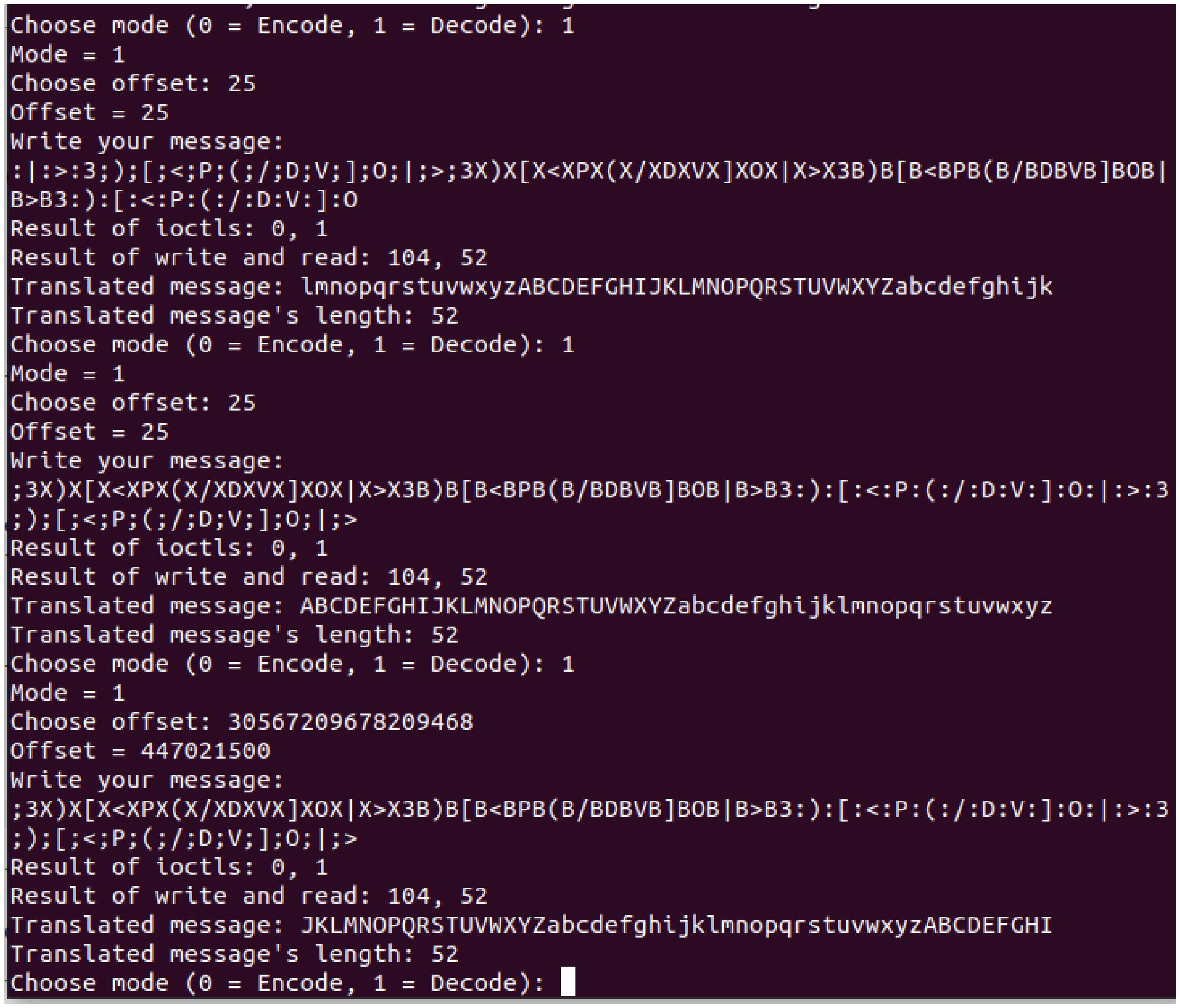
```
==== Hello there, This is the beginning of the Test Program! ====
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|B> B3:):[:<:P:(:/:D:V:]:O:|:>:
3;);[;<;P;(;/;D;V;];O;|;>
Result of ioctls: 0, 1
Result of write and read: 105, 53
Translated message: ABCDEFGHIJKLMNOPQRSTUVWXYZ aabcdefghijklzopqrstuvwxyz
Translated message's length: 53
Choose mode (0 = Encode, 1 = Decode): 
```

On a character that did not follow the proper order, it appears that there is some point in which the calculations go wrong with both mouth.



```
[16733.398940] || Relative IIdx 1: 2
[16733.398941] || Relative MIdx 1: 12
[16733.398942] || Relative IIdx 1: 1
[16733.398943] || Relative MIdx 2: 12
[16733.398943] || Offset found: 25
[16733.398944] || Char inserted into buffer from emote: Z
[16733.398945] || Char inserted into buffer as is:
[16733.398946] || Relative IIdx 1: 2
[16733.398947] || Relative MIdx 1: 0
[16733.398948] || Relative IIdx 1: 2
[16733.398949] || Relative MIdx 2: 0
[16733.398950] || Offset found: 26
[16733.398950] || Char inserted into buffer from emote: a
[16733.398951] || Relative IIdx 1: 3
[16733.398952] || Relative MIdx 1: 1
[16733.398953] || Relative IIdx 1: 2
[16733.398954] || Relative MIdx 2: 0
[16733.398955] || Offset found: 26
[16733.398956] || Char inserted into buffer from emote: a
```

Taking a break and then thinking about how the concept of how indexes before the first mouth would be a representation of the end of the alphabet, as well as understanding the logic behind decrementing the eye index (which is none at all), I just ended up removing the condition that changes the relative eye index. AND IT WORKS.



```
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
:|:>:3;);[;<;P;(;/;D;V;];O;|;>;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|
B>B3:):[:<:P:(:/:D:V:]:O
Result of ioctls: 0, 1
Result of write and read: 104, 52
Translated message: lmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk
Translated message's length: 52
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 25
Offset = 25
Write your message:
;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|B>B3:):[:<:P:(:/:D:V:]:O:|:>:3
;);[;<;P;(;/;D;V;];O;|;>
Result of ioctls: 0, 1
Result of write and read: 104, 52
Translated message: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Translated message's length: 52
Choose mode (0 = Encode, 1 = Decode): 1
Mode = 1
Choose offset: 30567209678209468
Offset = 447021500
Write your message:
;3X)X[X<XPX(X/XDXVX]XOX|X>X3B)B[B<BPB(B/BDBVB]BOB|B>B3:):[:<:P:(:/:D:V:]:O:|:>:3
;);[;<;P;(;/;D;V;];O;|;>
Result of ioctls: 0, 1
Result of write and read: 104, 52
Translated message: JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHI
Translated message's length: 52
Choose mode (0 = Encode, 1 = Decode):
```

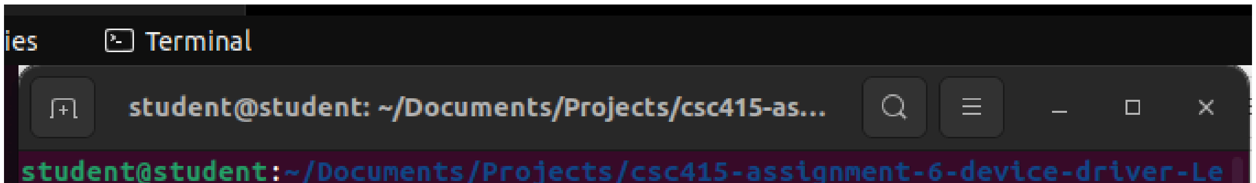
**Screen shot of compilation (aka building the kernel module and test program, as well as installing kernel module):**

## *KERNEL MODULE BUILDING AND CLEANUP*

### Build

Starting with the kernel module, where you have to be at the Module directory of the project before doing anything relating to it, …

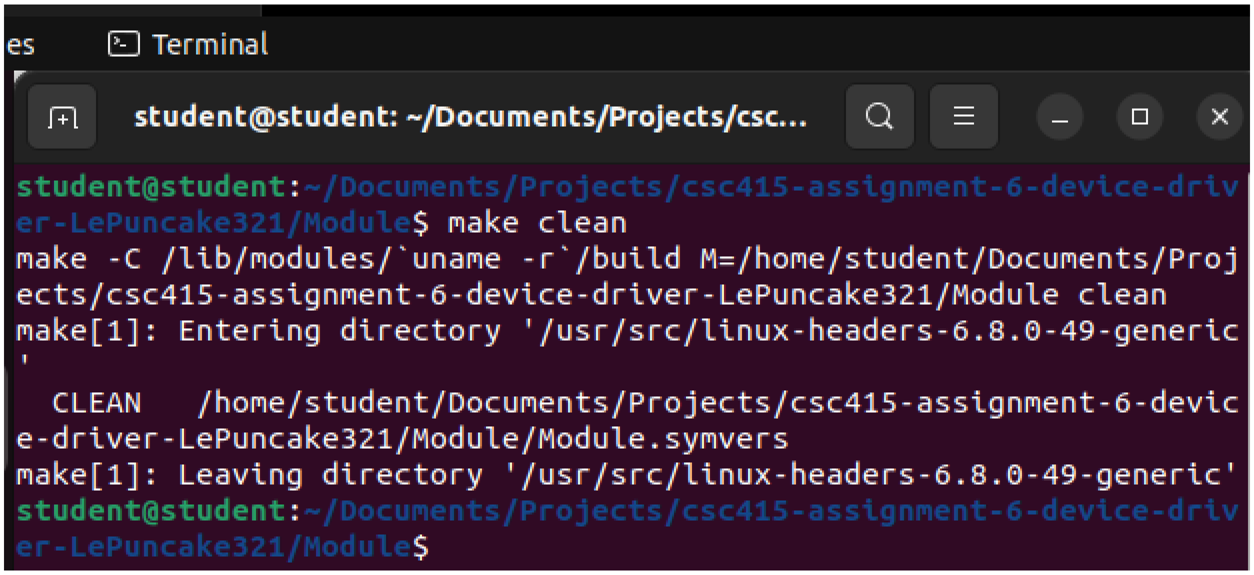
To build the kernel module, you first need to run:
`make clean`, and then
`make`



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-Le
```

### Clean Up

And finally run this command to cleanup the build of the kernel module
`make clean`



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ make clean
make -C /lib/modules/`uname -r`/build M=/home/student/Documents/Proj
ects/csc415-assignment-6-device-driver-LePuncake321/Module clean
make[1]: Entering directory '/usr/src/linux-headers-6.8.0-49-generic
'
  CLEAN   /home/student/Documents/Projects/csc415-assignment-6-devic
e-driver-LePuncake321/Module/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-49-generic'
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$
```

Note that it is best to clean up after you uninstalled the kernel module, since much of the functionality comes from the installed module.

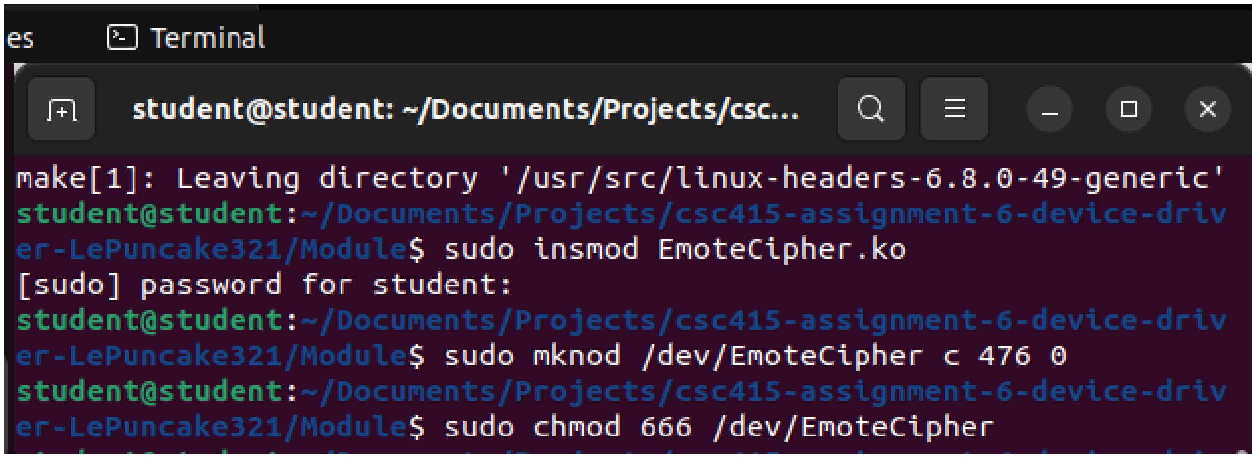
==================================================================

## *KERNEL MODULE INSTALLING AND UNINSTALLING*

### Installation

To now install the kernel module for use, after you built it, you can either …

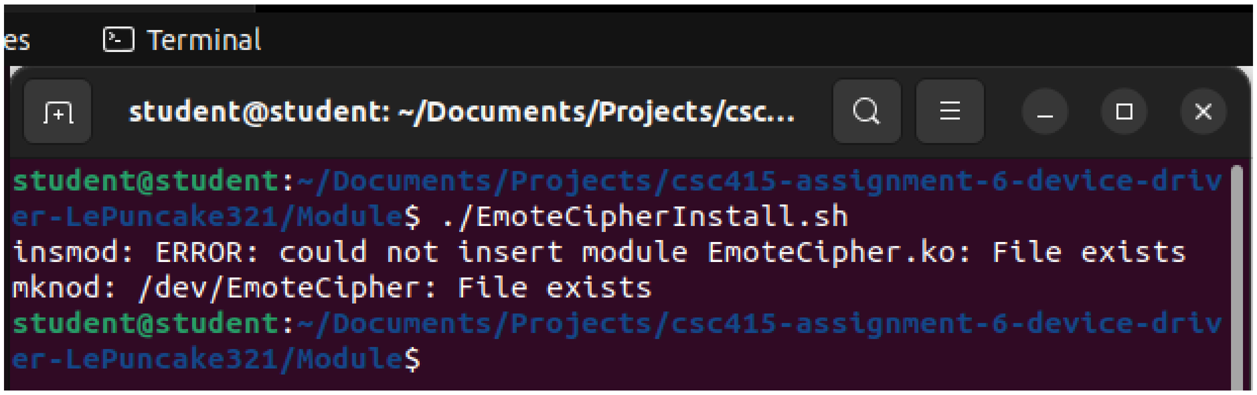
Run these command manually
```
sudo insmod EmoteCipher.ko
sudo mknod /dev/EmoteCipher c 476 0
sudo chmod 666 /dev/EmoteCipher
```



```
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-49-generic'
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ sudo insmod EmoteCipher.ko
[sudo] password for student:
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ sudo mknod /dev/EmoteCipher c 476 0
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ sudo chmod 666 /dev/EmoteCipher
```

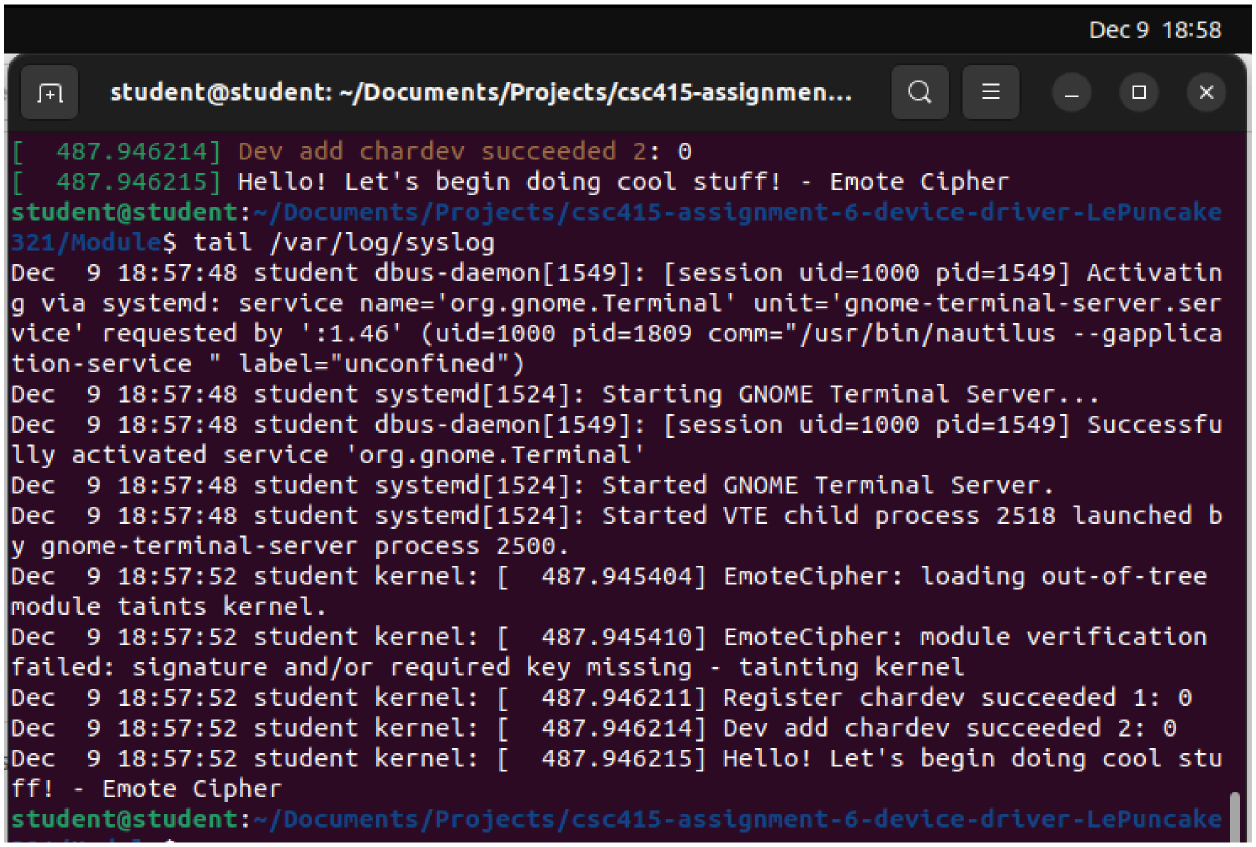
OR

Run this .sh file which does the three command above automatically
`./EmoteCipherInstall.sh`



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ ./EmoteCipherInstall.sh
insmod: ERROR: could not insert module EmoteCipher.ko: File exists
mknod: /dev/EmoteCipher: File exists
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$
```

Note that these two error on this screenshot only happen if you already ran either the three commands or .sh file to build the kernel module


Dec 9 18:58

student@student: ~/Documents/Projects/csc415-assignmen...

```
[  487.946214] Dev add chardev succeeded 2: 0
[  487.946215] Hello! Let's begin doing cool stuff! - Emote Cipher
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Module$ tail /var/log/syslog
Dec  9 18:57:48 student dbus-daemon[1549]: [session uid=1000 pid=1549] Activatin
g via systemd: service name='org.gnome.Terminal' unit='gnome-terminal-server.ser
vice' requested by ':1.46' (uid=1000 pid=1809 comm="/usr/bin/nautilus --gapplica
tion-service " label="unconfined")
Dec  9 18:57:48 student systemd[1524]: Starting GNOME Terminal Server...
Dec  9 18:57:48 student dbus-daemon[1549]: [session uid=1000 pid=1549] Successfu
lly activated service 'org.gnome.Terminal'
Dec  9 18:57:48 student systemd[1524]: Started GNOME Terminal Server.
Dec  9 18:57:48 student systemd[1524]: Started VTE child process 2518 launched b
y gnome-terminal-server process 2500.
Dec  9 18:57:52 student kernel: [  487.945404] EmoteCipher: loading out-of-tree
module taints kernel.
Dec  9 18:57:52 student kernel: [  487.945410] EmoteCipher: module verification
failed: signature and/or required key missing - tainting kernel
Dec  9 18:57:52 student kernel: [  487.946211] Register chardev succeeded 1: 0
Dec  9 18:57:52 student kernel: [  487.946214] Dev add chardev succeeded 2: 0
Dec  9 18:57:52 student kernel: [  487.946215] Hello! Let's begin doing cool stu
ff! - Emote Cipher
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
```

Another thing of note is that when you run these commands to install the kernel module, there will be three messages written onto/var/log/syslog to indicate that it was installed properly onto the system.

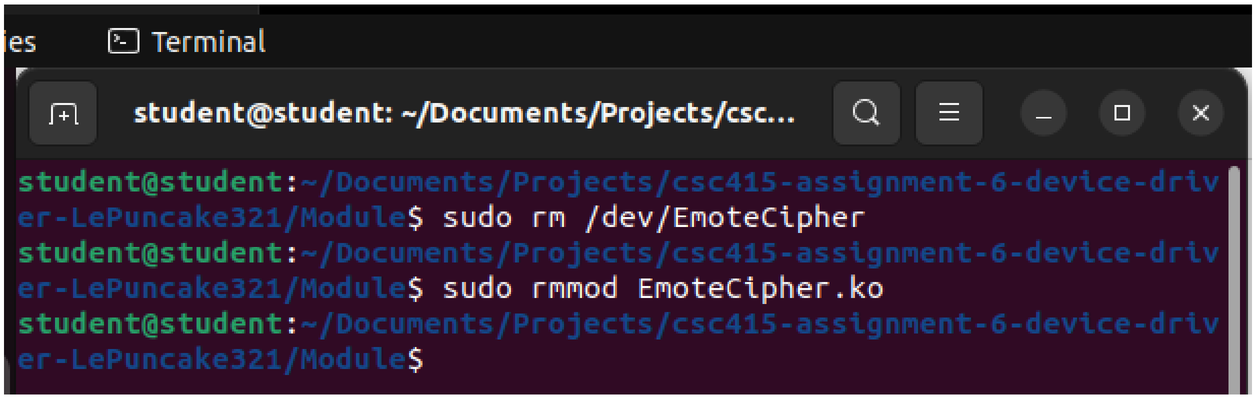
```
Register chardev succeeded 1: 0
Dev add chardev succeeded 2: 0
Hello! Let's begin doing cool stuff! - Emote Cipher
```

## Uninstallation

To uninstall the kernel module, you can either …

Run these commands manually

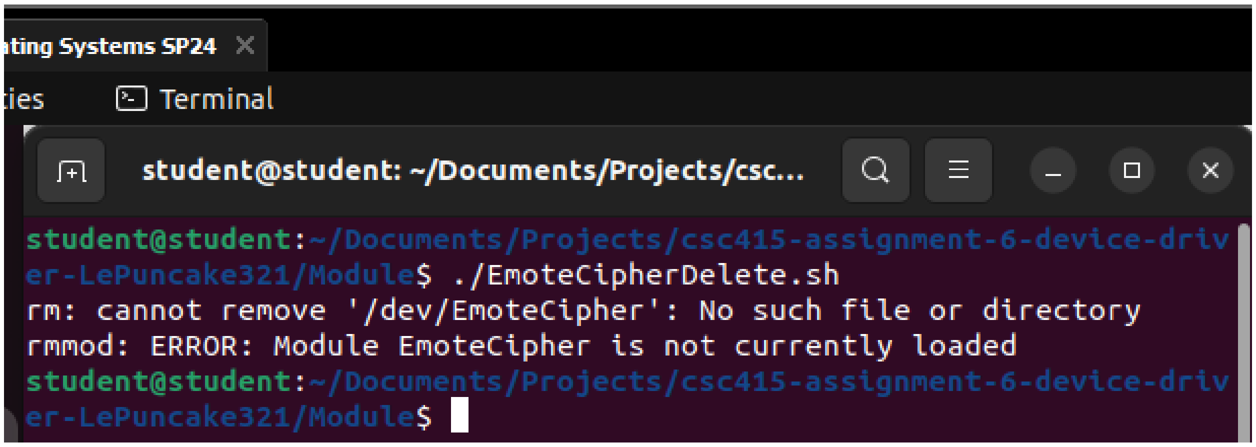
```
sudo rm /dev/EmoteCipher
sudo rmmod EmoteCipher.ko
```



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ sudo rm /dev/EmoteCipher
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ sudo rmmod EmoteCipher.ko
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$
```

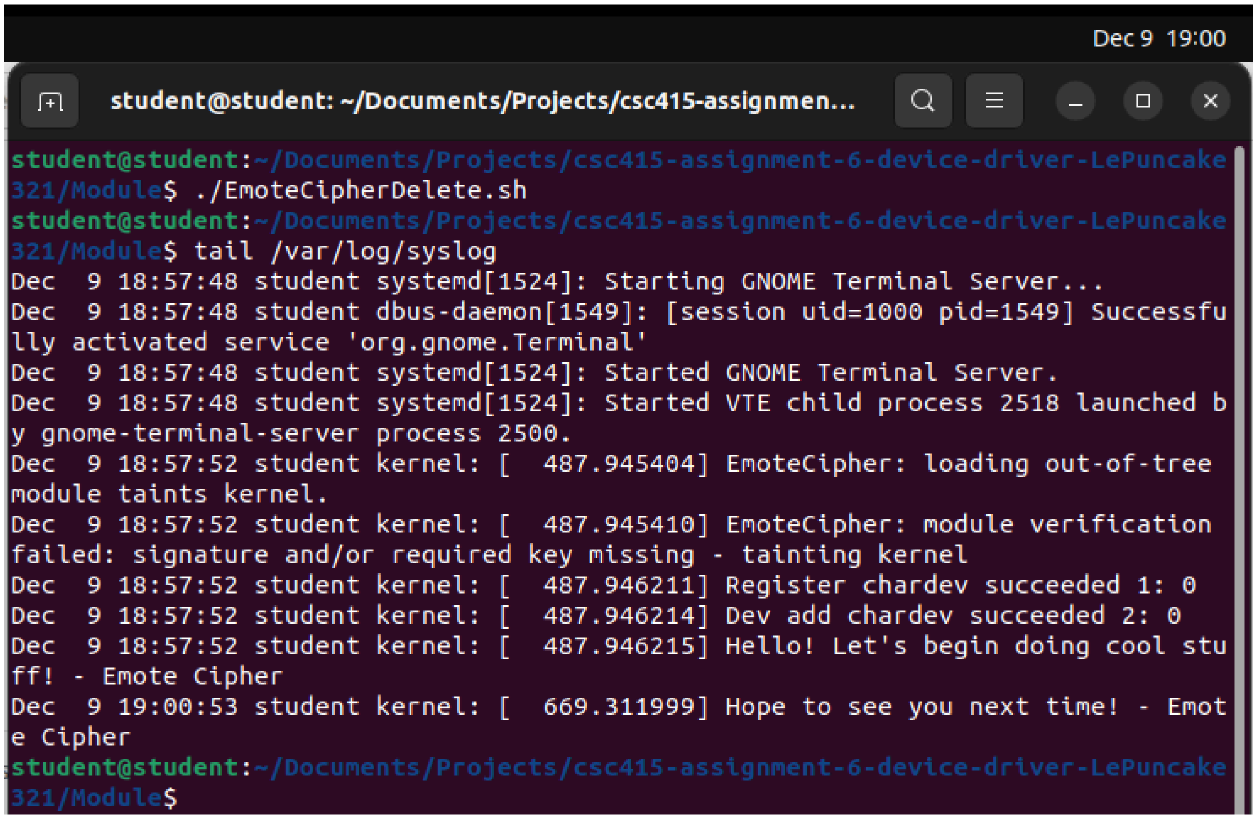
OR

Run this .sh file which runs the same command above automatically
`./EmoteCipherDelete.sh`



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$ ./EmoteCipherDelete.sh
rm: cannot remove '/dev/EmoteCipher': No such file or directory
rmmod: ERROR: Module EmoteCipher is not currently loaded
student@student:~/Documents/Projects/csc415-assignment-6-device-driv
er-LePuncake321/Module$
```

Note again that these two errors only appear if you already uninstalled the kernel module with the .sh file or the two commands.



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Module$ ./EmoteCipherDelete.sh
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Module$ tail /var/log/syslog
Dec  9 18:57:48 student systemd[1524]: Starting GNOME Terminal Server...
Dec  9 18:57:48 student dbus-daemon[1549]: [session uid=1000 pid=1549] Successfu
lly activated service 'org.gnome.Terminal'
Dec  9 18:57:48 student systemd[1524]: Started GNOME Terminal Server.
Dec  9 18:57:48 student systemd[1524]: Started VTE child process 2518 launched b
y gnome-terminal-server process 2500.
Dec  9 18:57:52 student kernel: [  487.945404] EmoteCipher: loading out-of-tree
module taints kernel.
Dec  9 18:57:52 student kernel: [  487.945410] EmoteCipher: module verification
failed: signature and/or required key missing - tainting kernel
Dec  9 18:57:52 student kernel: [  487.946211] Register chardev succeeded 1: 0
Dec  9 18:57:52 student kernel: [  487.946214] Dev add chardev succeeded 2: 0
Dec  9 18:57:52 student kernel: [  487.946215] Hello! Let's begin doing cool stu
ff! - Emote Cipher
Dec  9 19:00:53 student kernel: [  669.311999] Hope to see you next time! - Emot
e Cipher
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Module$
```

Another thing of note is that when you run these commands to uninstall the kernel module, there will be one message written onto/var/log/syslog to indicate that it was uninstalled properly from the system.

```
Hope to see you next time! - Emote Cipher
```

=================================================================================

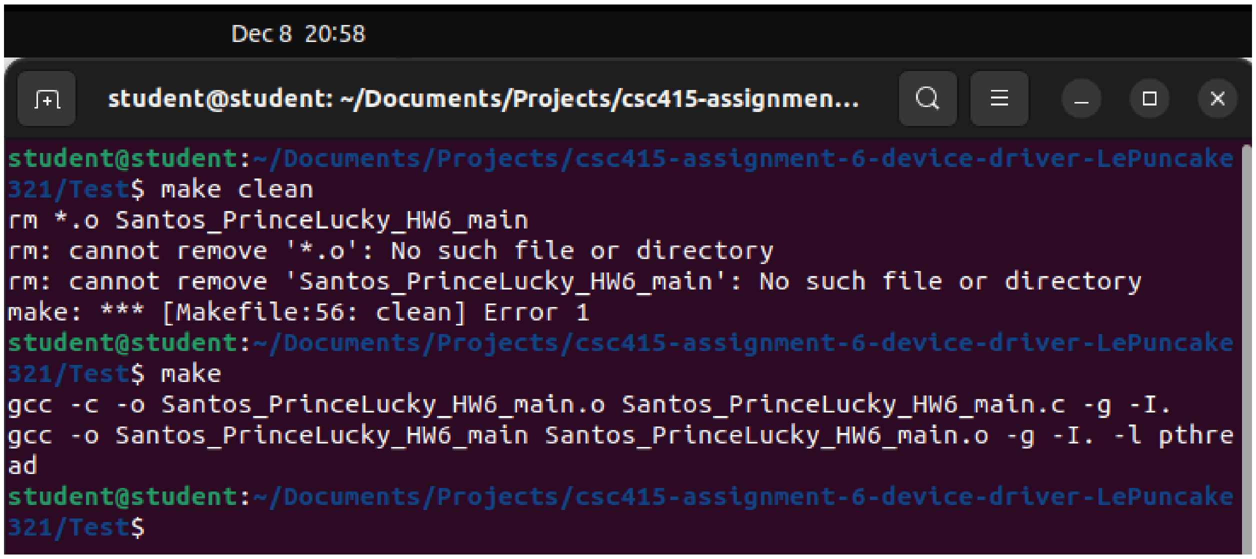
## *TEST PROGRAM BUILDING AND CLEANUP*

### Build

As for building and cleaning the test program, which requires you to be at the Test directory of the project, …

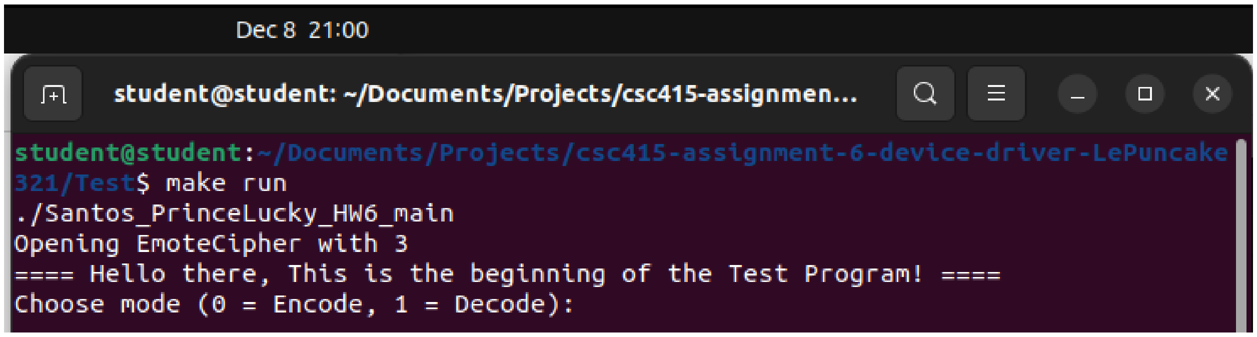
Run these two commands to build the test program for use

```
make clean
make
```

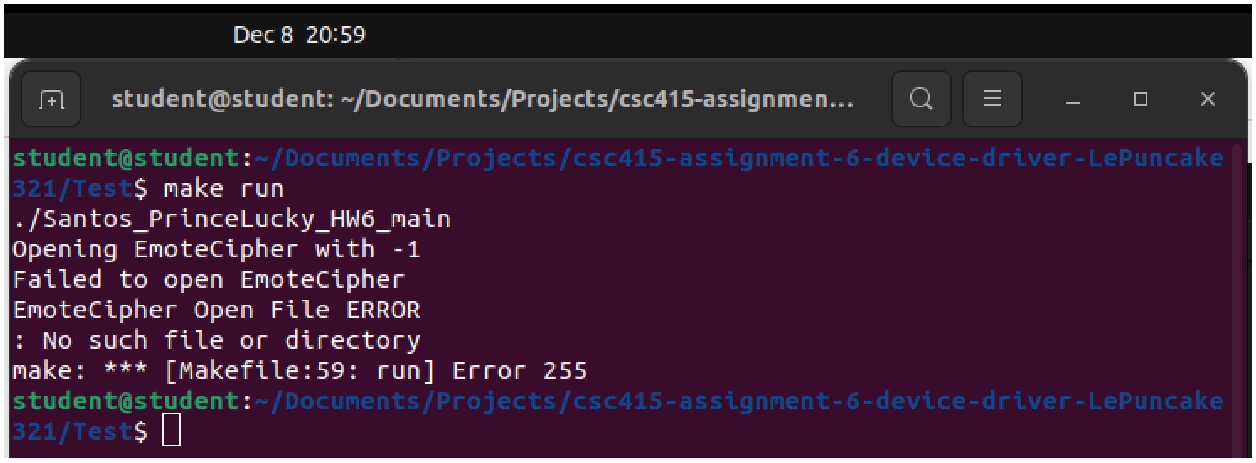

```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ make clean
rm *.o Santos_PrinceLucky_HW6_main
rm: cannot remove '*.o': No such file or directory
rm: cannot remove 'Santos_PrinceLucky_HW6_main': No such file or directory
make: *** [Makefile:56: clean] Error 1
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ make
gcc -c -o Santos_PrinceLucky_HW6_main.o Santos_PrinceLucky_HW6_main.c -g -I.
gcc -o Santos_PrinceLucky_HW6_main Santos_PrinceLucky_HW6_main.o -g -I. -l pthre
ad
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$
```

Run this command to run the program on the terminal:

```
make run
```

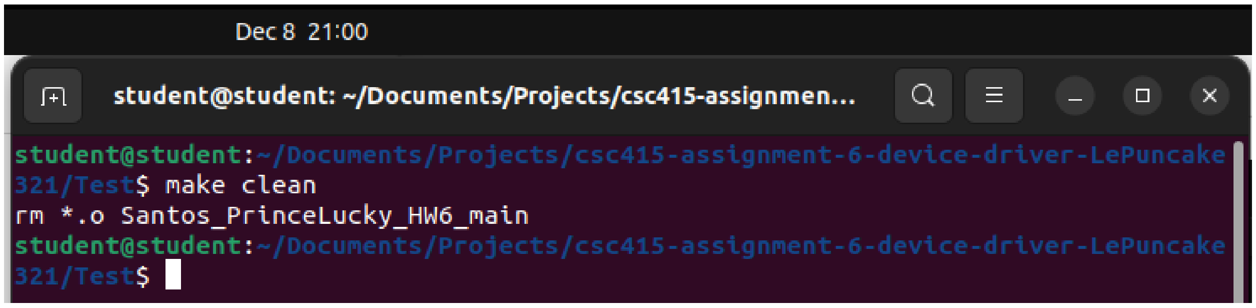

```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ make run
./Santos_PrinceLucky_HW6_main
Opening EmoteCipher with 3
==== Hello there, This is the beginning of the Test Program! ====
Choose mode (0 = Encode, 1 = Decode):
```

Be sure that you've already built and installed the kernel module before building and running the test program, or else you will be unable to use it and get these error messages


```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ make run
./Santos_PrinceLucky_HW6_main
Opening EmoteCipher with -1
Failed to open EmoteCipher
EmoteCipher Open File ERROR
: No such file or directory
make: *** [Makefile:59: run] Error 255
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$
```

## Clean Up

Finally, run this command to clean up the test program
`make clean`



```
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ make clean
rm *.o Santos_PrinceLucky_HW6_main
student@student:~/Documents/Projects/csc415-assignment-6-device-driver-LePuncake
321/Test$ 
```

**Screen shot(s) of the execution of the program (aka using the test program):**

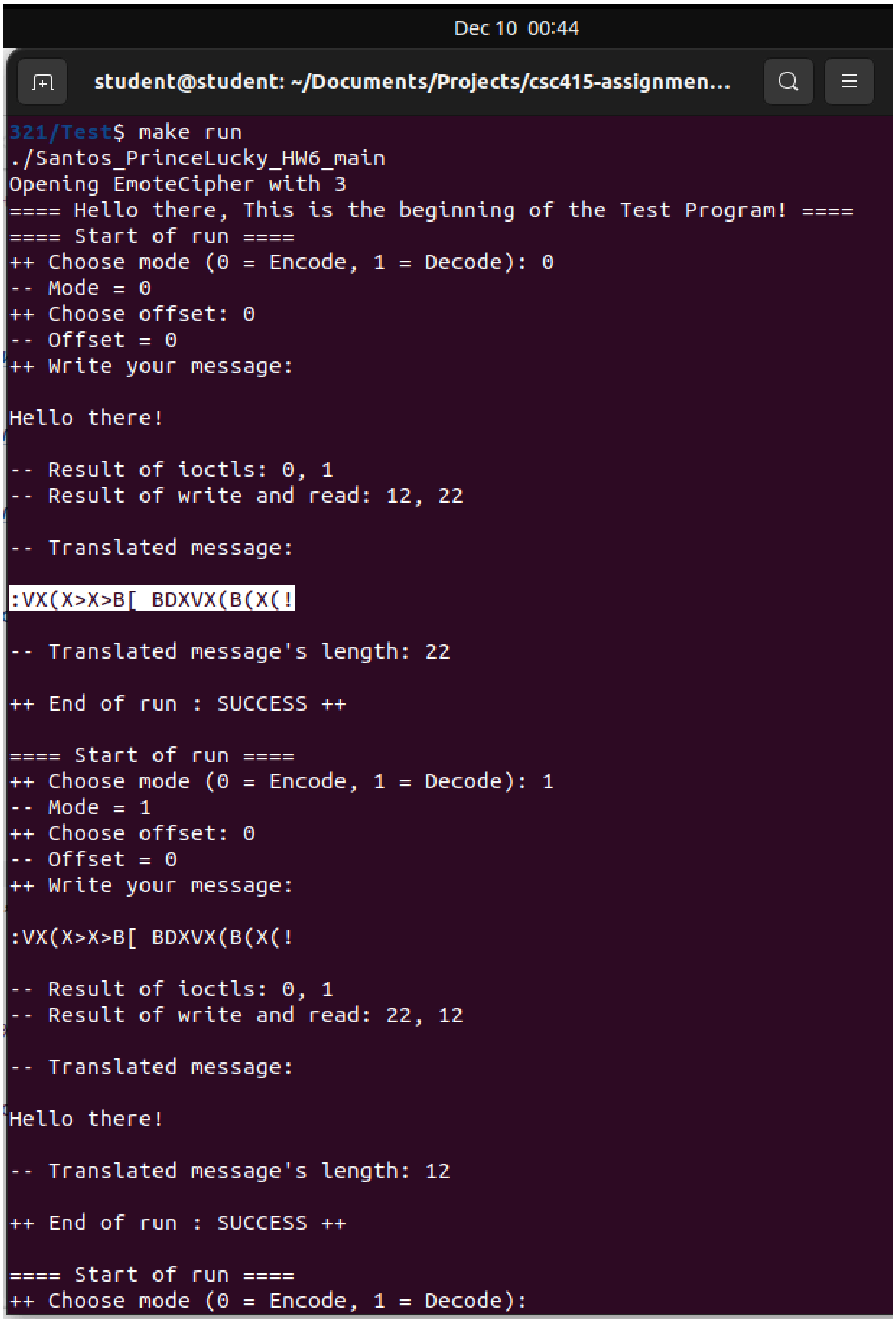
When you `make run` the test program, the user will be prompted to input an integer for the device driver's mode: 0 to encode alphabetical letters into emotes, and 1 to decode emotes into alphabetical letters. This is also where the program can exit with either any other integer number than 0 or 1, or an invalid input to exit the program.

Next, the program will ask for an integer for the offset. This number can be any kind of whole number, negative or positive, but if an invalid type was given, then the program will also exit here.

Now, the user can input any kind of message they want to encode/decode, and once it is entered, the user will see how many characters were written into the device driver as well as how many characters were read. After that, the program prints out the output from the read function by the device driver.

After that output from read, the program asks the user again for the mode, offset and a message, until the user exits the program intentionally or unintentionally.

Example input: “Hello there!” with both encode and decode mode, and offset of 0

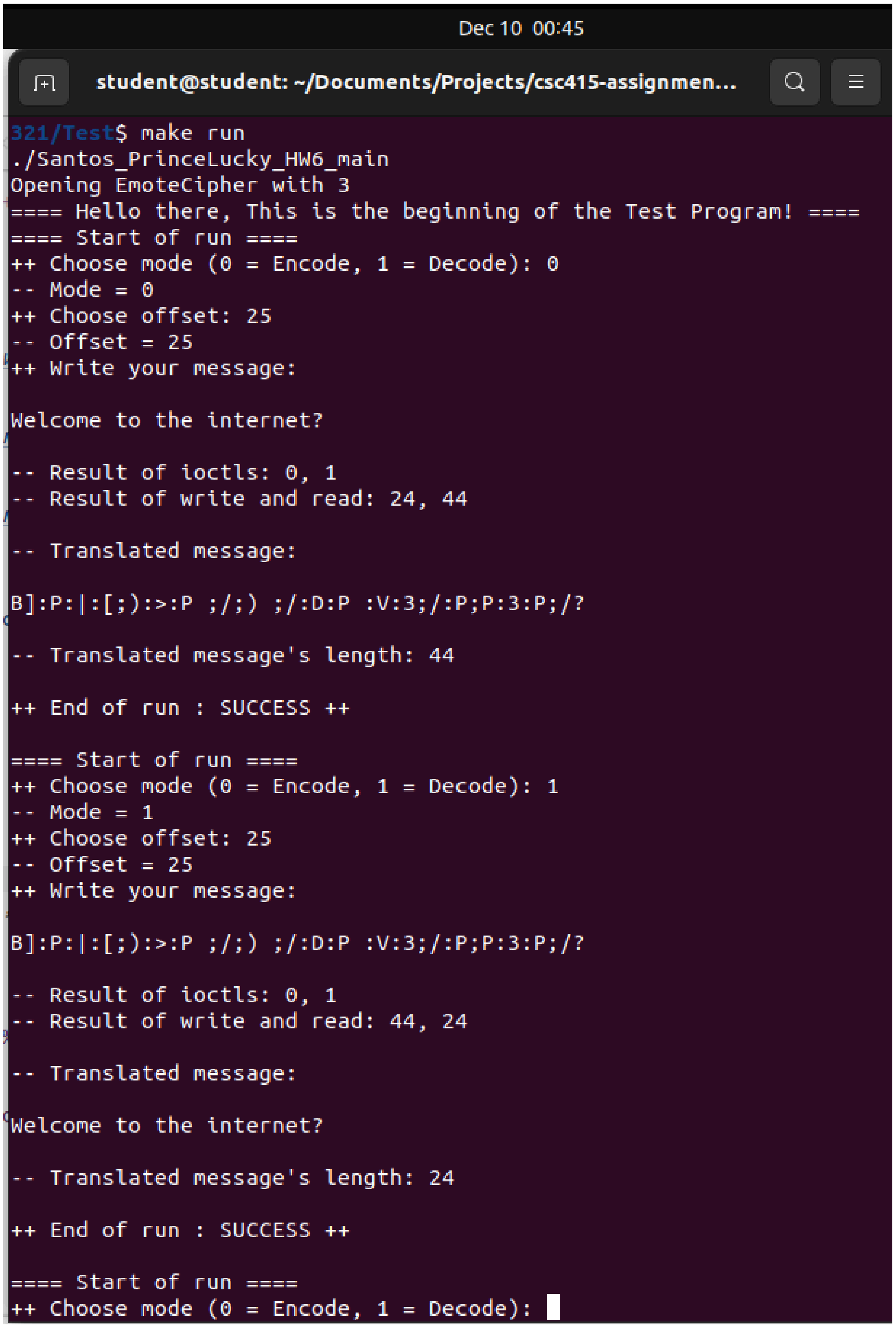


Dec 10 00:44

student@student: ~/Documents/Projects/csc415-assignmen...

```
321/Test$ make run
./Santos_PrinceLucky_HW6_main
Opening EmoteCipher with 3
==== Hello there, This is the beginning of the Test Program! ====
==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 0
-- Mode = 0
++ Choose offset: 0
-- Offset = 0
++ Write your message:

Hello there!

-- Result of ioctls: 0, 1
-- Result of write and read: 12, 22

-- Translated message:

:VX(X>X>B[ BDXVX(B(X(!

-- Translated message's length: 22

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 1
-- Mode = 1
++ Choose offset: 0
-- Offset = 0
++ Write your message:

:VX(X>X>B[ BDXVX(B(X(!

-- Result of ioctls: 0, 1
-- Result of write and read: 22, 12

-- Translated message:

Hello there!

-- Translated message's length: 12

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode):
```

Example input: “Welcome to the internet?” with offset set to 25, both encode and decode

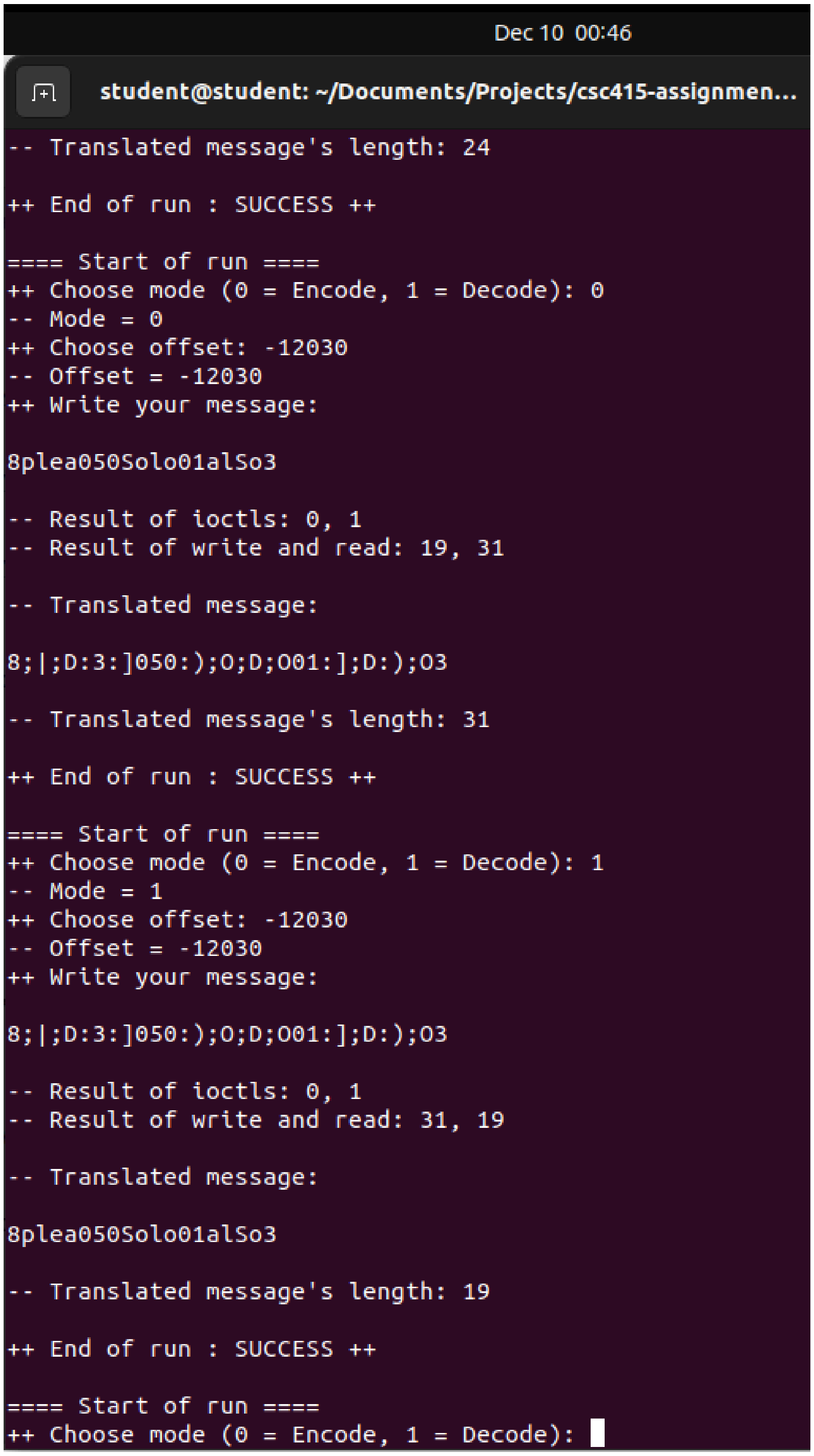


Dec 10 00:45

student@student: ~/Documents/Projects/csc415-assignmen...

```
321/Test$ make run
./Santos_PrinceLucky_HW6_main
Opening EmoteCipher with 3
==== Hello there, This is the beginning of the Test Program! ====
==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 0
-- Mode = 0
++ Choose offset: 25
-- Offset = 25
++ Write your message:

Welcome to the internet?

-- Result of ioctls: 0, 1
-- Result of write and read: 24, 44

-- Translated message:

B]:P:|:[;):>:P ;/;) ;/:D:P :V:3;/:P;P:3:P;/?

-- Translated message's length: 44

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 1
-- Mode = 1
++ Choose offset: 25
-- Offset = 25
++ Write your message:

B]:P:|:[;):>:P ;/;) ;/:D:P :V:3;/:P;P:3:P;/?

-- Result of ioctls: 0, 1
-- Result of write and read: 44, 24

-- Translated message:

Welcome to the internet?

-- Translated message's length: 24

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode):
```

Example input: “8plea050Solo01alSo3” with offset -12030, both encode and decode



```
Dec 10 00:46
student@student: ~/Documents/Projects/csc415-assignmen...
-- Translated message's length: 24

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 0
-- Mode = 0
++ Choose offset: -12030
-- Offset = -12030
++ Write your message:

8plea050Solo01alSo3

-- Result of ioctls: 0, 1
-- Result of write and read: 19, 31

-- Translated message:

8;|;D:3:]050:);O;D;O01:];D:);O3

-- Translated message's length: 31

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode): 1
-- Mode = 1
++ Choose offset: -12030
-- Offset = -12030
++ Write your message:

8;|;D:3:]050:);O;D;O01:];D:);O3

-- Result of ioctls: 0, 1
-- Result of write and read: 31, 19

-- Translated message:

8plea050Solo01alSo3

-- Translated message's length: 19

++ End of run : SUCCESS ++

==== Start of run ====
++ Choose mode (0 = Encode, 1 = Decode):
```