

Task 1

Question 1

- If the length of your prefix file is not multiple of 64, what is going to happen?
- I set the size of `prefix.txt` at 164 which is not a multiple of 64. The `md5sum` command shows no difference between the two executeable since it's padded with zeroes.

```
> ls -l prefix.txt
-rw-rw-r-- 168 yuri 23 Jan 11:52 prefix.tx
> md5sum out?.bin
5c11e69d1cd6336e3e91d931556a2cfc out1.bin
5c11e69d1cd6336e3e91d931556a2cfc out2.bin
```

Figure 1: No Difference

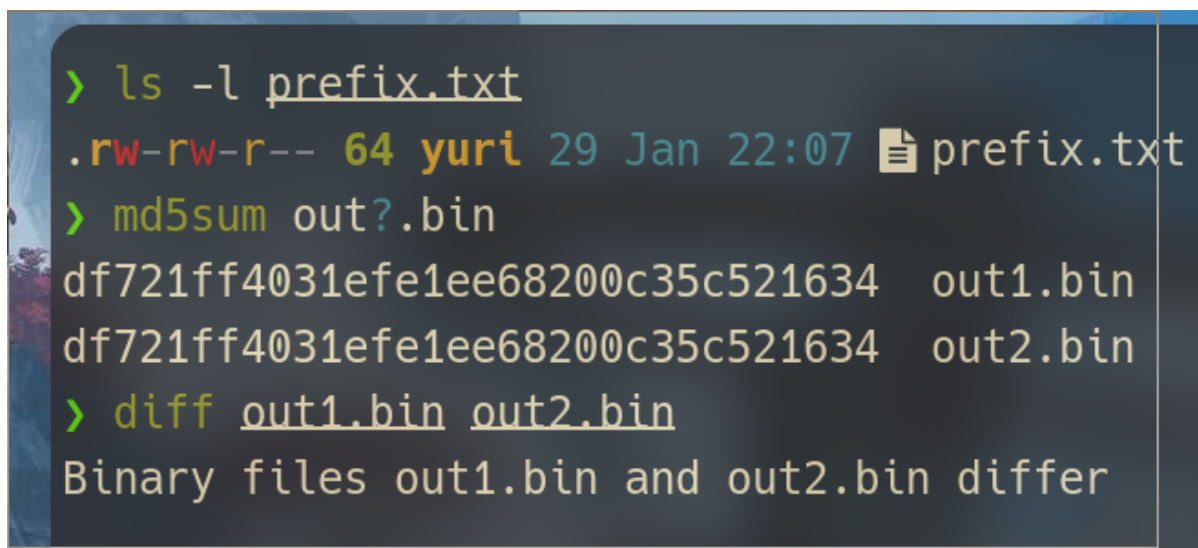
- The `diff` command show difference in binary form.

```
> diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
\ bless out? bin
```

Figure 2: Difference in Binary

Question 2

- Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.



```

> ls -l prefix.txt
.rw-rw-r-- 64 yuri 29 Jan 22:07 prefix.txt
> md5sum out?.bin
df721ff4031efe1ee68200c35c521634 out1.bin
df721ff4031efe1ee68200c35c521634 out2.bin
> diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ

```

Figure 3: 64 Bytes input

- As seen above, the two files have the same MD5 sum. However, they are two different files as showed using the `diff` command.

Question 3

- The two files generated by `md5collgen` have some similarities, especially in the beginning. However, they are still different.



Figure 4: Out1.bin

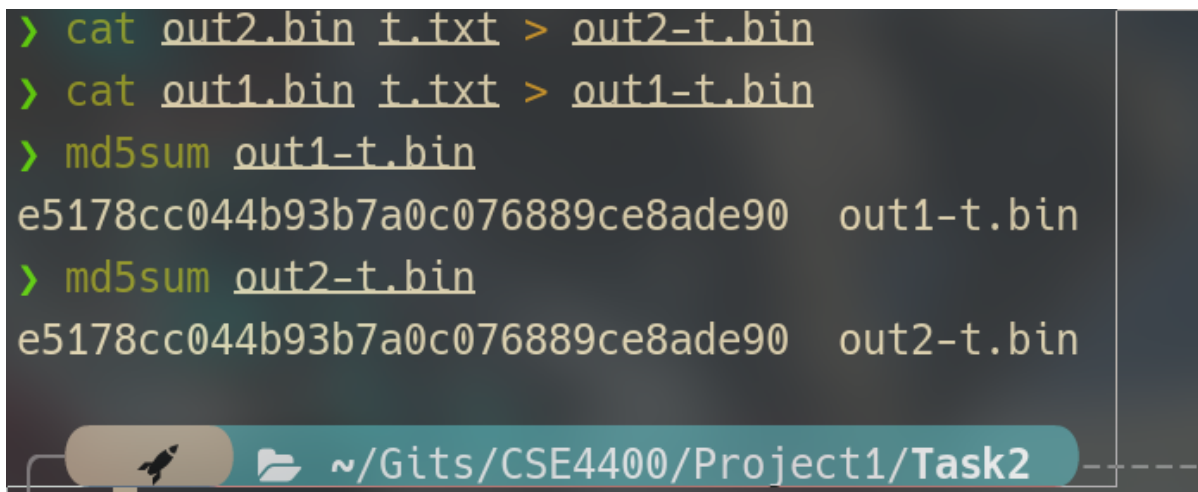


Figure 5: Out2.bin

(Sorry if the screenshot is blurred)

Task 2

- From the previous task, we know that the two files Out1 and Out2 have the same hash. In this case, $\text{MD5}\{\text{Out1}\} = \text{MD5}\{\text{Out2}\}$.
- I created a text file that contains a word. I concatenated this file with Out1 and Out2 respectively and verified that the MD5 hash results of the two files are the same.

A terminal window with a dark background and light green text. The commands and output are as follows:

```
> cat out2.bin t.txt > out2-t.bin
> cat out1.bin t.txt > out1-t.bin
> md5sum out1-t.bin
e5178cc044b93b7a0c076889ce8ade90  out1-t.bin
> md5sum out2-t.bin
e5178cc044b93b7a0c076889ce8ade90  out2-t.bin
```

The terminal has a status bar at the bottom with a folder icon, a file icon, and the path `~/Gits/CSE4400/Project1/Task2`.

```
> cat out2.bin t.txt > out2-t.bin
> cat out1.bin t.txt > out1-t.bin
> md5sum out1-t.bin
e5178cc044b93b7a0c076889ce8ade90  out1-t.bin
> md5sum out2-t.bin
e5178cc044b93b7a0c076889ce8ade90  out2-t.bin
```

Figure 6: Same output

Task 3

- Using the code in the instruction, I filled the array with 200 character 'A' and using the command `xxd -g1 task3` to display the binary values.

```

00003000: 00 00 00 00 00 00 00 00 08 40 00 00 00 00 00 00 .....@.....
00003010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00003020: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003030: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003040: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003050: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003060: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003070: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003080: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
00003090: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
000030a0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
000030b0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
000030c0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
000030d0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAA
000030e0: 41 41 41 41 41 41 41 41 47 43 43 3a 20 28 55 62 AAAAAAGCC: (Ub
000030f0: 75 6e 74 75 20 39 2e 34 2e 30 2d 31 75 62 75 6e untu 9.4.0-1ubun
00003100: 74 75 31 7e 32 30 2e 30 34 2e 32 20 20 30 2e 34 tu1~20 04 2) 9 4

```

Figure 7: A's location

- We can see that the array starts from `3020` in hex which is `12320` in decimal, but this value is not a multiple of 64. The nearest value which is the multiple of 64 is `12352`. This is our prefix.
- I Run the command `head -c 12352 task3 > prefix`. Use this `prefix` to as input for `md5collgen`. As a result, I created 2 binary files which have the same md5 hash.

```
> md5collgen -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: b177f11c26506096668e1900b59f9fa9

Generating first block: ...
Generating second block: S11.....
Running time: 3.03293 s
> md5sum out?.bin
zsh: no matches found: out?.bin
> md5sum out1
c9a7a1e79b4dd01ab45e97a34fe29054  out1
> md5sum out2
c9a7a1e79b4dd01ab45e97a34fe29054  out2
```

Figure 8: Collision

- Now We can create P and Q using out1 and out2 last 128 bytes. To find the start of the suffix region, we add the value where prefix's end with 128. This yields 12480.

```
> tail -c 128 out1 > p
> tail -c 128 out2 > q
> tail -c +12481 task3 > suffix
> cat prefix p suffix > bin1
> cat prefix q suffix > bin2
> diff bin?
Binary files bin1 and bin2 differ
> md5sum bin1
a9f070a44cc34e5d1a4423968f00c949  bin1
> md5sum bin2
a9f070a44cc34e5d1a4423968f00c949  bin2
```

Figure 9: Different Binary but same Hash

Task 4

- Like what we did in task 3, now I created 2 arrays that filled with As and find which binary region they resided on.

```

00003070: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003080: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003090: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000030a0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000030b0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000030c0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000030d0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000030e0: 41 41 41 41 41 41 41 41 00 00 00 00 00 00 00 00  AAAAAAA.....
000030f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00003100: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003110: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003120: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003130: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003140: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003150: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003160: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003170: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003180: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00003190: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000031a0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000031b0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000031c0: 41 41 41 41 41 41 41 41 47 43 43 3a 20 28 55 62  AAAAAAGCC: (Ub
000031d0: 75 6e 74 75 20 39 2e 34 2e 30 2d 31 75 62 75 6e  untu 9.4.0-1ubun
000031e0: 74 75 31 7e 32 30 2e 30 34 2e 32 29 20 39 2e 34  tu1~20.04.2) 9.4
000031f0: 2e 30 00 2c 00 00 00 02 00 00 00 00 00 08 00 00  0

```

Figure 10: A's location for 2 arrays

- Notice that the prefix is still the same which is `12352`. Therefore, we have the same prefix as we did in part 3. Creating p and q is like what we did in task 3 which take the last 128 bytes of out1 and out2: `tail -c 128 out1.bin > p`. Same for the Suffix.

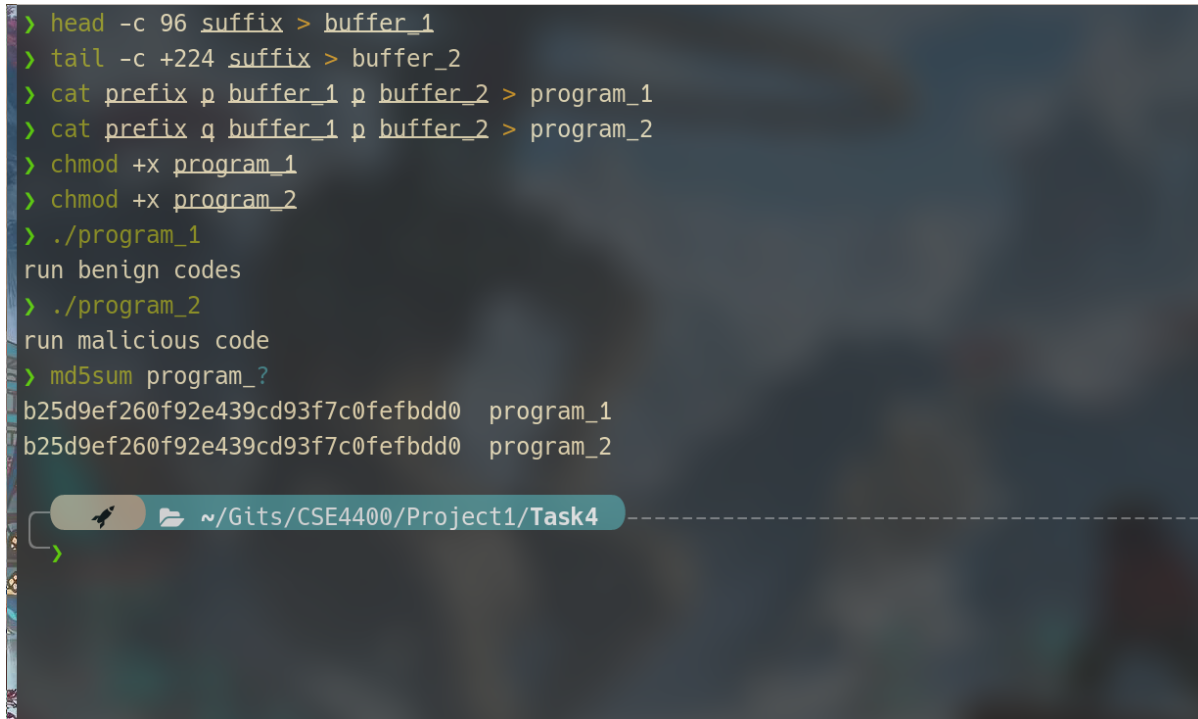

```

> tail -c +12481 task4 > suffix
> xxd -g1 suffix
00000000: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000010: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000020: 41 41 41 41 41 41 41 41 00 00 00 00 00 00 00 00  AAAAAAA.....
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000040: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000050: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000060: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000070: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000080: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000090: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000a0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000b0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000c0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000d0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000e0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000000f0: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
00000100: 41 41 41 41 41 41 41 41 47 43 43 3a 20 28 55 62  AAAAAAAAGCC: (Ub
00000110: 75 6e 74 75 20 39 2e 34 2e 30 2d 31 75 62 75 6e  untu 9.4.0-1ubun
00000120: 74 75 31 7e 32 30 2e 30 34 2e 32 29 20 39 2e 34  tu1~20.04.2) 9.4
00000130: 2e 30 00 2c 00 00 00 02 00 00 00 00 00 08 00 00  .0.....

```

Figure 11: Second array's Location

- Notice from the picture above, we can see the second array starts at 0x0040 which is 64 in decimal. We need to add another 32 A's until we reach the start of the second P. Name this region `buffer_1` which has length of 96
- P's region take 128 bytes. Therefore, we add 128 to `buffer_1`'s region to obtain 224 bytes which is the rest of the region. Name it `buffer_2`.
- We put everything together using the `Cat` command as shown in the picture below:

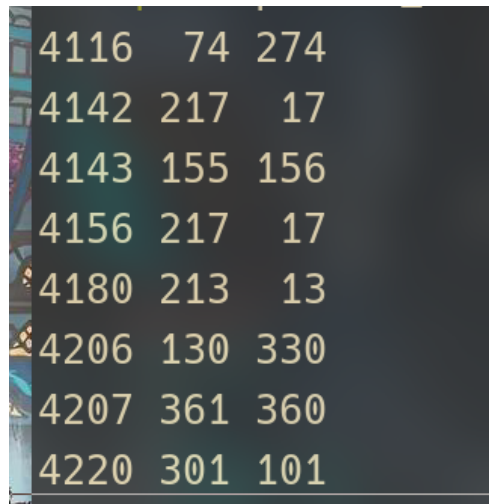
A terminal window with a dark background and light-colored text. The user enters a series of commands to create two programs, program_1 and program_2, from a file named suffix. The commands use head and tail to extract specific parts of the suffix file and cat to concatenate them with different prefixes. Both programs are then made executable and run. The output of program_1 is 'run benign codes' and the output of program_2 is 'run malicious code'. Finally, the user runs md5sum on both programs, and the terminal shows that both have the same MD5 hash: b25d9ef260f92e439cd93f7c0fefbdd0. A file manager icon and the path ~/Gits/CSE4400/Project1/Task4 are visible at the bottom of the terminal window.

```
> head -c 96 suffix > buffer_1
> tail -c +224 suffix > buffer_2
> cat prefix p buffer_1 p buffer_2 > program_1
> cat prefix q buffer_1 p buffer_2 > program_2
> chmod +x program_1
> chmod +x program_2
> ./program_1
run benign codes
> ./program_2
run malicious code
> md5sum program_?
b25d9ef260f92e439cd93f7c0fefbdd0  program_1
b25d9ef260f92e439cd93f7c0fefbdd0  program_2
```

Figure 12: Two different programs with same hashes

Task 5

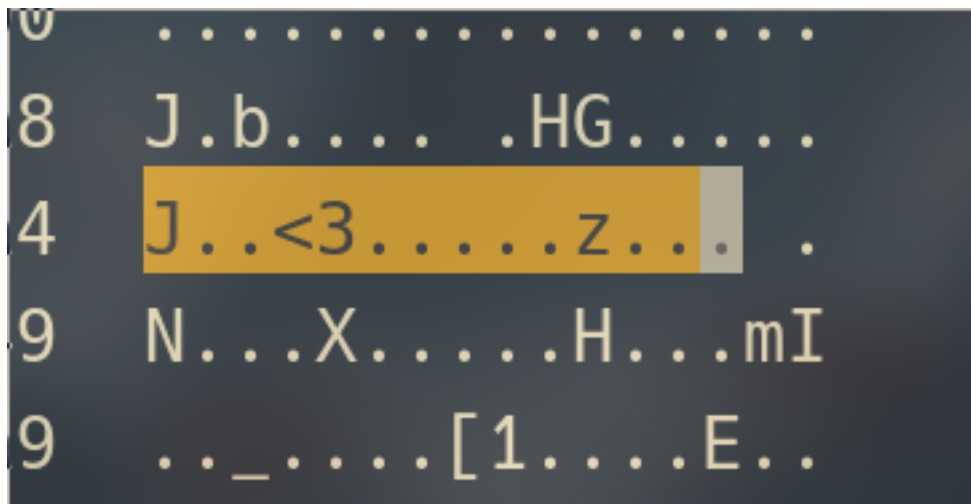
- I moved the data section and the text section.
- I identify the starting point of my array at offset 4096 which lucky enough to be a multiple of 64. The command to get prefix's region is the same as before: `head -c 4096 task5 > prefix`.
- P and Q are obtained just like we did before: `tail -c 128 out1 > p`
- I compared P and Q and see the difference at



4116	74	274
4142	217	17
4143	155	156
4156	217	17
4180	213	13
4206	130	330
4207	361	360
4220	301	101

Figure 13: P and Q

- To show some reference, here is the different in ASCII text:



```

0 . . . . .
8 J . b . . . . . H G . . . . .
4 J . . < 3 . . . . . z . . . . .
9 N . . . X . . . . . H . . . m I
9 . . _ . . . . [ 1 . . . . . E . . . . .

```

Figure 14: P

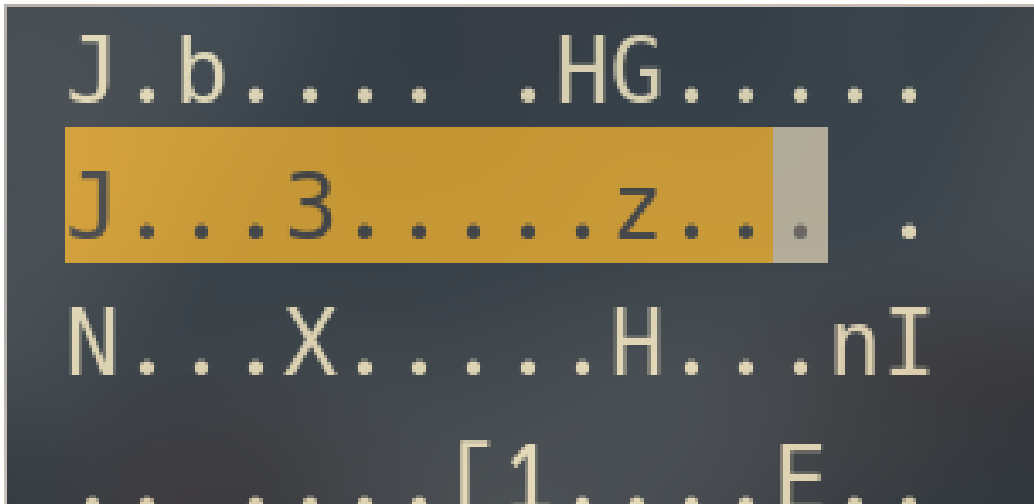


Figure 15: Q

- I marked the different and change the code accordingly. Then, I recompiled the code and find the suffix again.
- I add 128 (P's region) bytes with 4096 and get 4224 as the region for the suffix.
- After I put everything together, we have two different programs.

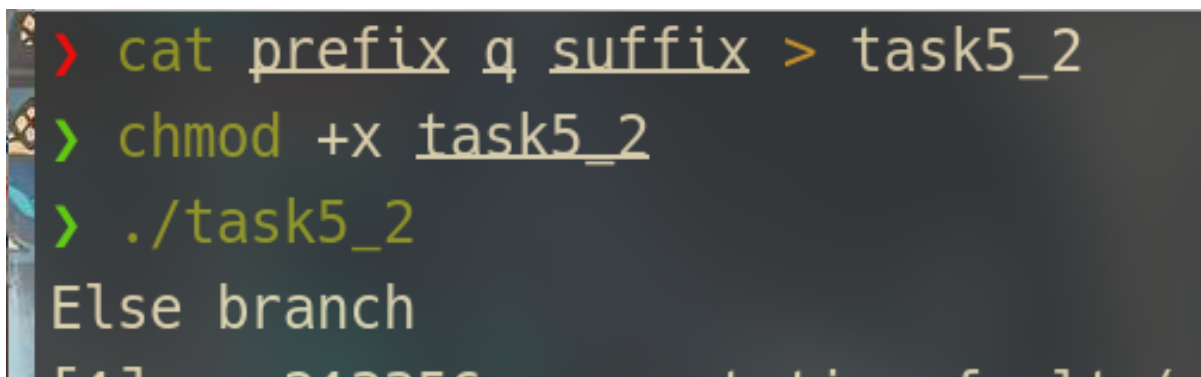


Figure 16: Two difference programs

- However they have the same hashes.

```
> cat prefix p suffix > task5_1  
> chmod +x task5_1  
> ./task5_1  
If branch
```

Figure 17: Two difference programs

```
> md5sum task5_  
c4ff786bcd1fe580a31ca02c2a0087a task5_1  
c4ff786bcd1fe580a31ca02c2a0087a task5_2
```

Figure 18: Same Hash