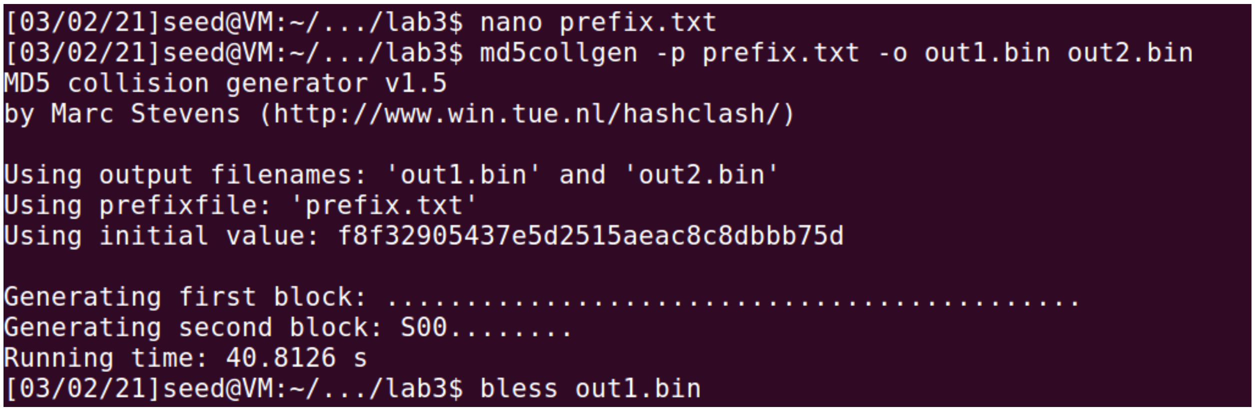
Shane Hagan

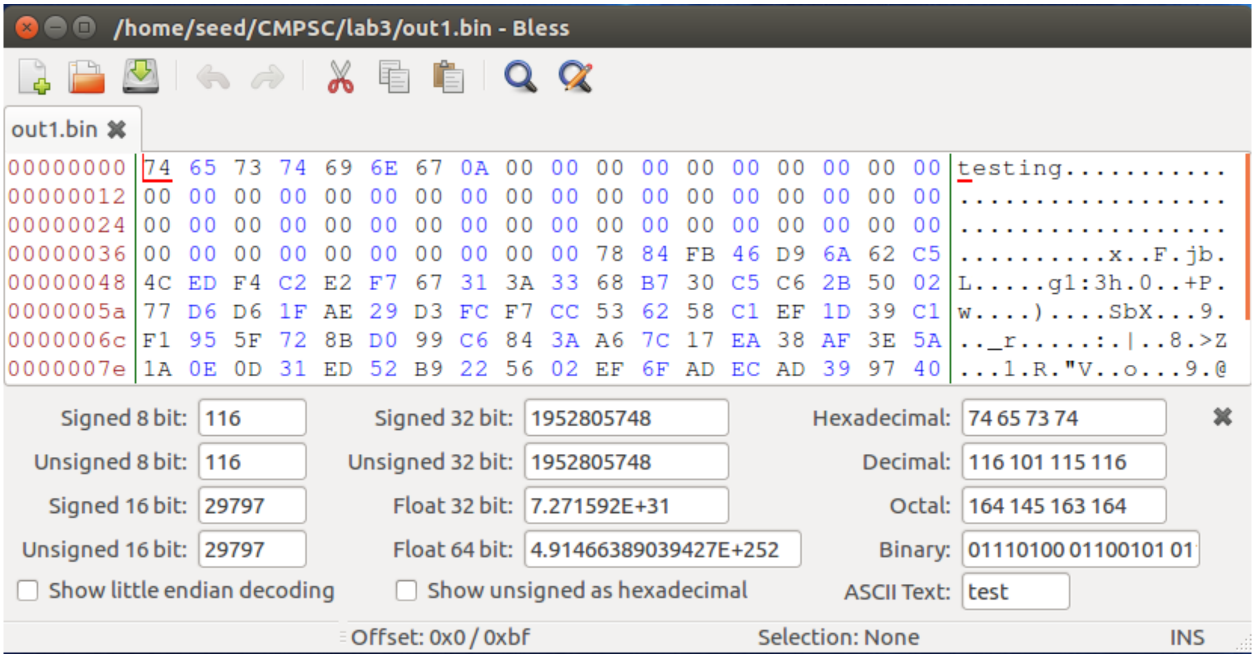
CMPSC 443 – Lab 3

03/12/21

Task 1



```
[03/02/21]seed@VM:~/.../lab3$ nano prefix.txt
[03/02/21]seed@VM:~/.../lab3$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: f8f32905437e5d2515aeac8c8dbbb75d

Generating first block: ......................................
Generating second block: S00........
Running time: 40.8126 s
[03/02/21]seed@VM:~/.../lab3$ bless out1.bin
```



/home/seed/CMPSC/lab3/out1.bin - Bless

out1.bin

```
00000000 74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing...........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 00 00 00 00 00 00 00 00 00 00 78 84 FB 46 D9 6A 62 C5 ..........x..F.jb.
00000048 4C ED F4 C2 E2 F7 67 31 3A 33 68 B7 30 C5 C6 2B 50 02 L.....g1:3h.0..+P.
0000005a 77 D6 D6 1F AE 29 D3 FC F7 CC 53 62 58 C1 EF 1D 39 C1 w....)....SbX...9.
0000006c F1 95 5F 72 8B D0 99 C6 84 3A A6 7C 17 EA 38 AF 3E 5A .._r.....:.|..8.>Z
0000007e 1A 0E 0D 31 ED 52 B9 22 56 02 EF 6F AD EC AD 39 97 40 ...1.R."V..o...9.@
```

Signed 8 bit: 116
Unsigned 8 bit: 116
Signed 16 bit: 29797
Unsigned 16 bit: 29797
□ Show little endian decoding

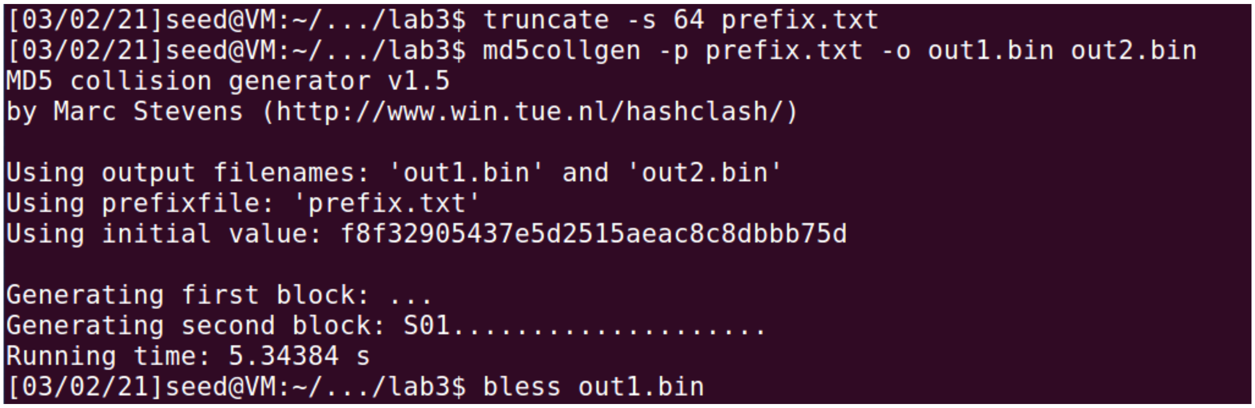
Signed 32 bit: 1952805748
Unsigned 32 bit: 1952805748
Float 32 bit: 7.271592E+31
Float 64 bit: 4.91466389039427E+252
□ Show unsigned as hexadecimal

Hexadecimal: 74 65 73 74
Decimal: 116 101 115 116
Octal: 164 145 163 164
Binary: 01110100 01100101 01
ASCII Text: test

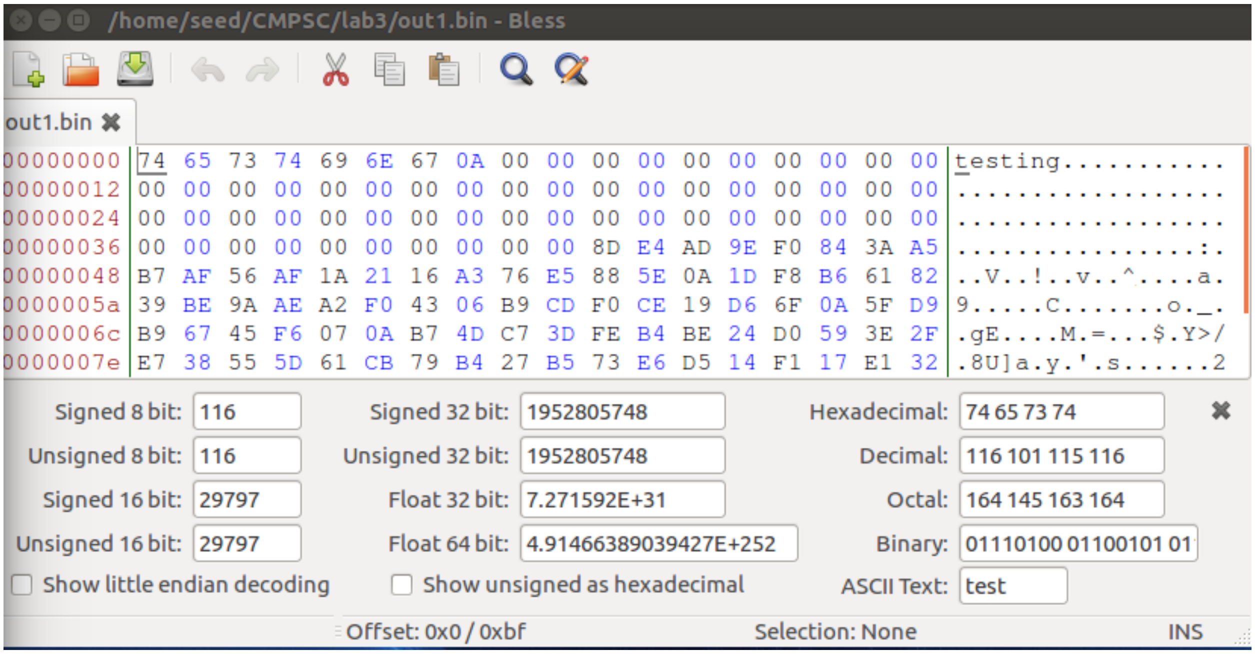
Offset: 0x0 / 0xbf   Selection: None   INS

Question 1: If the length of your prefix file is not multiple of 64, what is going to happen?

Answer: If it is NOT a multiple of 64, then the file will be padded with zeros.



```
[03/02/21]seed@VM:~/.../lab3$ truncate -s 64 prefix.txt
[03/02/21]seed@VM:~/.../lab3$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: f8f32905437e5d2515aeac8c8dbbb75d

Generating first block: ...
Generating second block: S01...................
Running time: 5.34384 s
[03/02/21]seed@VM:~/.../lab3$ bless out1.bin
```



/home/seed/CMPSC/lab3/out1.bin - Bless

```
00000000 74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing...........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 00 00 00 00 00 00 00 00 00 00 8D E4 AD 9E F0 84 3A A5 ................:.
00000048 B7 AF 56 AF 1A 21 16 A3 76 E5 88 5E 0A 1D F8 B6 61 82 ..V..!..v..^....a.
0000005a 39 BE 9A AE A2 F0 43 06 B9 CD F0 CE 19 D6 6F 0A 5F D9 9.....C.......o._.
0000006c B9 67 45 F6 07 0A B7 4D C7 3D FE B4 BE 24 D0 59 3E 2F .gE....M.=...$.Y>/
0000007e E7 38 55 5D 61 CB 79 B4 27 B5 73 E6 D5 14 F1 17 E1 32 .8U]a.y.'.s......2
```

Signed 8 bit: 116
Unsigned 8 bit: 116
Signed 16 bit: 29797
Unsigned 16 bit: 29797
□ Show little endian decoding

Signed 32 bit: 1952805748
Unsigned 32 bit: 1952805748
Float 32 bit: 7.271592E+31
Float 64 bit: 4.91466389039427E+252
□ Show unsigned as hexadecimal

Hexadecimal: 74 65 73 74
Decimal: 116 101 115 116
Octal: 164 145 163 164
Binary: 01110100 01100101 01
ASCII Text: test

Offset: 0x0 / 0xbf   Selection: None   INS

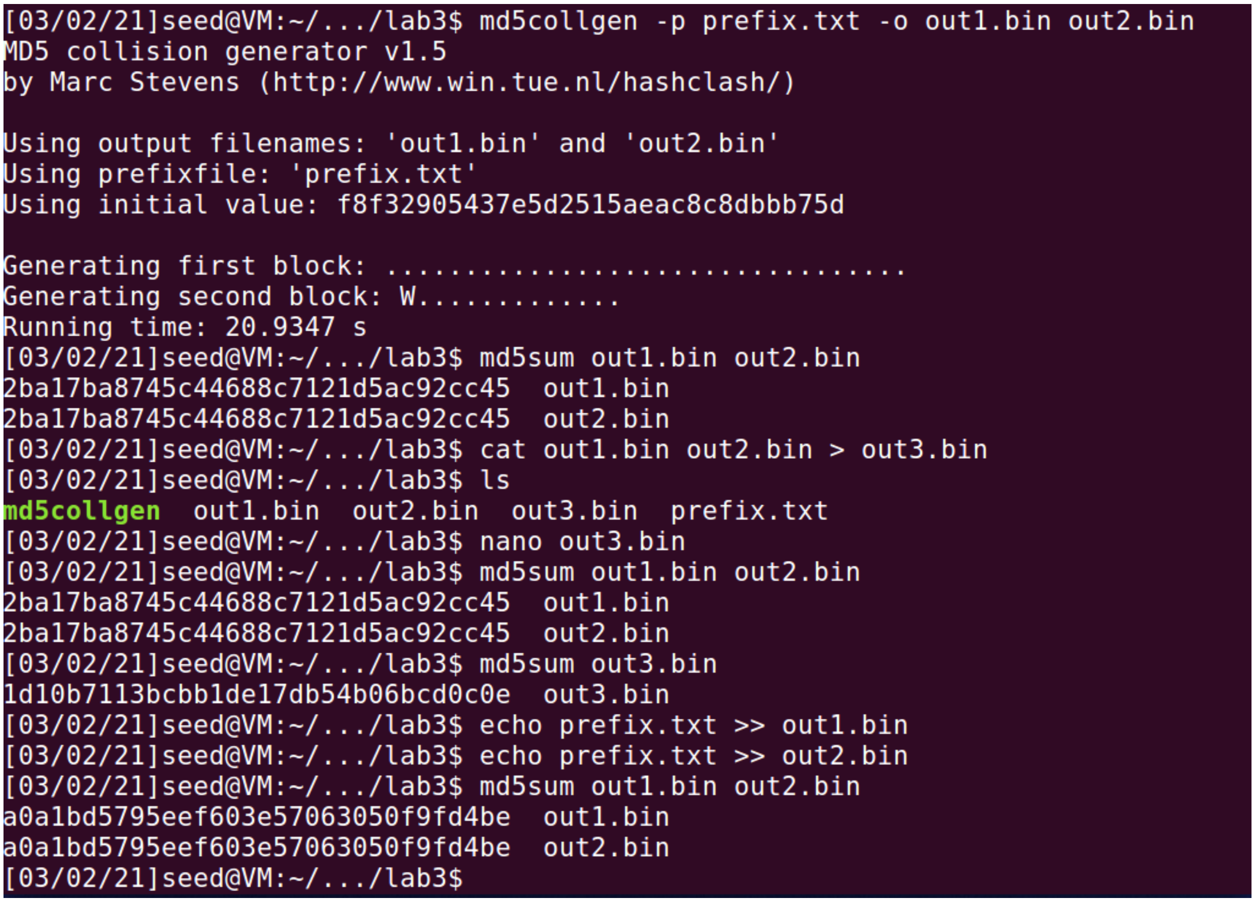
<u>Question 2</u>: Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

Answer: It will be the opposite of the first case, there will be no zero padding on the file.

<u>Question 3</u>: Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

Answer: Only some of the bytes differ, some remain the same.

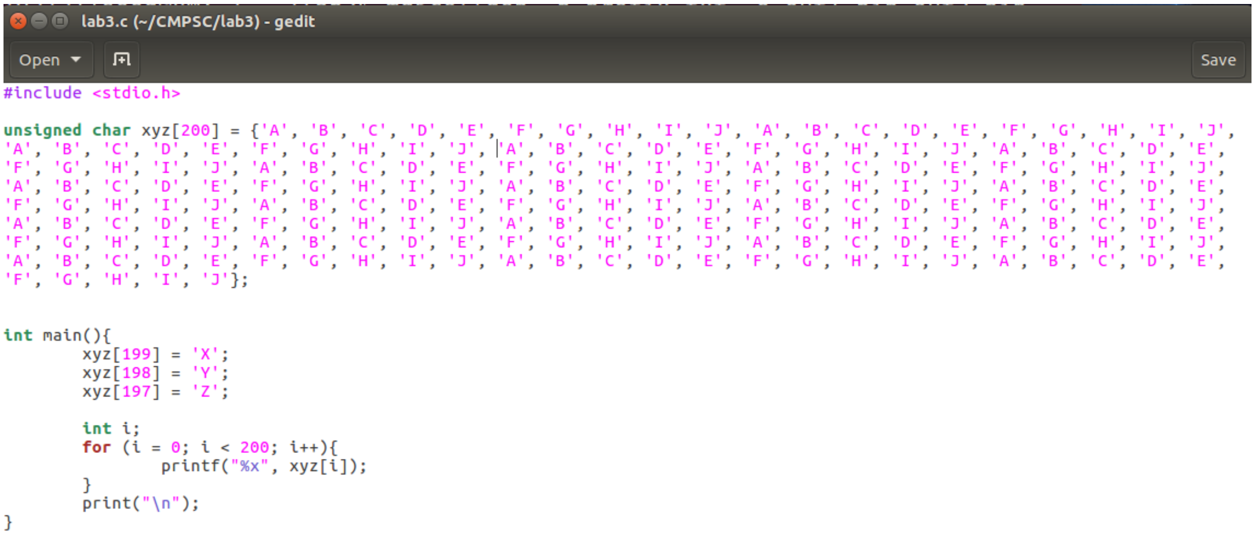
Task 2



```
[03/02/21]seed@VM:~/.../lab3$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: f8f32905437e5d2515aeac8c8dbbb75d

Generating first block: ...............................
Generating second block: W............
Running time: 20.9347 s
[03/02/21]seed@VM:~/.../lab3$ md5sum out1.bin out2.bin
2ba17ba8745c44688c7121d5ac92cc45  out1.bin
2ba17ba8745c44688c7121d5ac92cc45  out2.bin
[03/02/21]seed@VM:~/.../lab3$ cat out1.bin out2.bin > out3.bin
[03/02/21]seed@VM:~/.../lab3$ ls
md5collgen  out1.bin  out2.bin  out3.bin  prefix.txt
[03/02/21]seed@VM:~/.../lab3$ nano out3.bin
[03/02/21]seed@VM:~/.../lab3$ md5sum out1.bin out2.bin
2ba17ba8745c44688c7121d5ac92cc45  out1.bin
2ba17ba8745c44688c7121d5ac92cc45  out2.bin
[03/02/21]seed@VM:~/.../lab3$ md5sum out3.bin
1d10b7113bcbb1de17db54b06bcd0c0e  out3.bin
[03/02/21]seed@VM:~/.../lab3$ echo prefix.txt >> out1.bin
[03/02/21]seed@VM:~/.../lab3$ echo prefix.txt >> out2.bin
[03/02/21]seed@VM:~/.../lab3$ md5sum out1.bin out2.bin
a0a1bd5795eef603e57063050f9fd4be  out1.bin
a0a1bd5795eef603e57063050f9fd4be  out2.bin
[03/02/21]seed@VM:~/.../lab3$
```

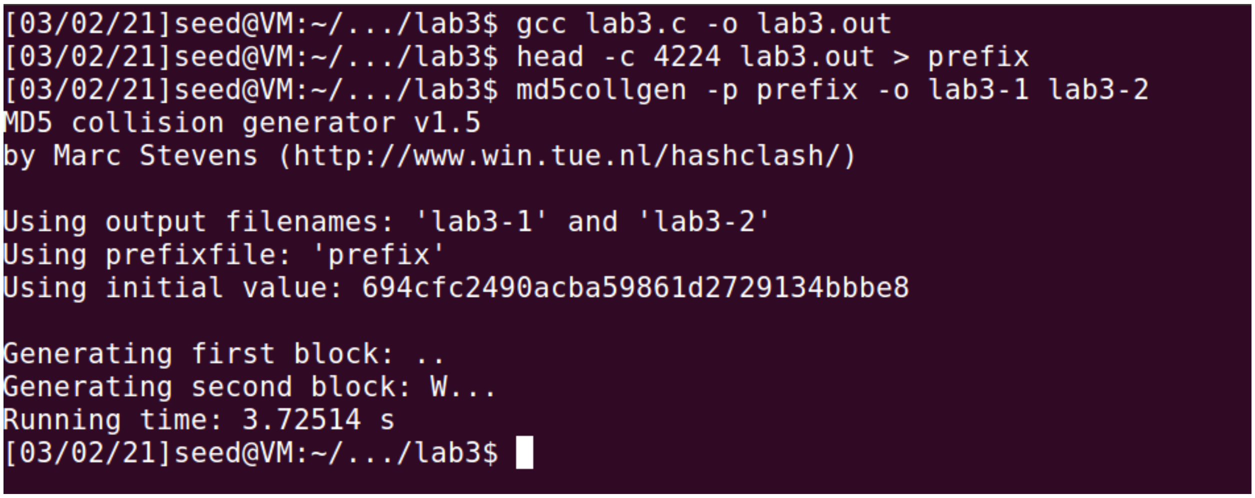
Here, I begin by once again creating the files. From there, I use the md5sum function passing in the two .bin files I created, this checks to see the hashes of each file. As shown, they are the same, from there, I was not very sure how to use cat function, so I improvised and just used echo to append a string to the end of each file. I then checked the MD5 hashes again, and they ended up being the same again, so this could be a successful method.

Task 3



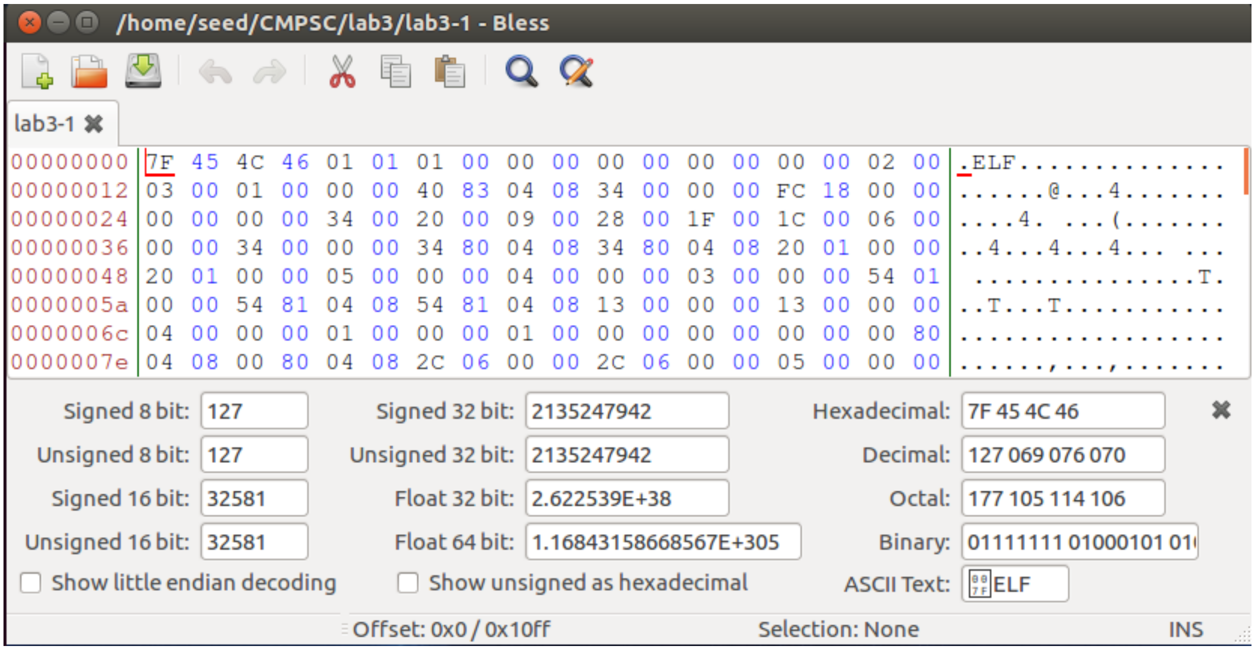
```
#include <stdio.h>

unsigned char xyz[200] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J'};


int main(){
        xyz[199] = 'X';
        xyz[198] = 'Y';
        xyz[197] = 'Z';

        int i;
        for (i = 0; i < 200; i++){
                printf("%x", xyz[i]);
        }
        print("\n");
}
```

Here is the code used for this lab. Just slightly modified the given one to us, also filled the array with the first 10 letters of the alphabet, 20 times to give us 200.



```
[03/02/21]seed@VM:~/.../lab3$ gcc lab3.c -o lab3.out
[03/02/21]seed@VM:~/.../lab3$ head -c 4224 lab3.out > prefix
[03/02/21]seed@VM:~/.../lab3$ md5collgen -p prefix -o lab3-1 lab3-2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'lab3-1' and 'lab3-2'
Using prefixfile: 'prefix'
Using initial value: 694cfc2490acba59861d2729134bbbe8

Generating first block: ..
Generating second block: W...
Running time: 3.72514 s
[03/02/21]seed@VM:~/.../lab3$ 
```

Generating two files with the same MD5 hash, but with different suffixes, just as described in the lab guidelines.

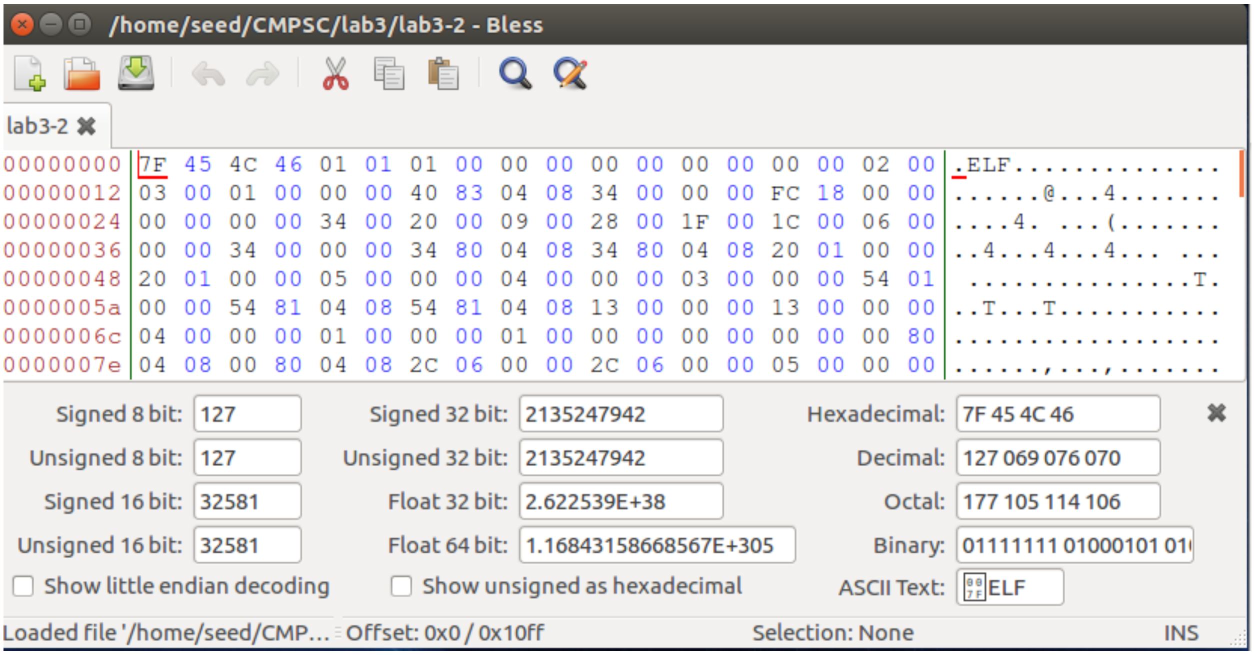


```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 .ELF..............
00000012 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 ......@...4.......
00000024 00 00 00 00 34 00 20 00 09 00 28 00 1F 00 1C 00 06 00 ....4. ...(.......
00000036 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 ..4...4...4... ...
00000048 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 54 01  ..............T.
0000005a 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 ..T...T...........
0000006c 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 ..................
0000007e 04 08 00 80 04 08 2C 06 00 00 2C 06 00 00 05 00 00 00 ......,...,.......
```

Signed 8 bit: 127 | Signed 32 bit: 2135247942 | Hexadecimal: 7F 45 4C 46
Unsigned 8 bit: 127 | Unsigned 32 bit: 2135247942 | Decimal: 127 069 076 070
Signed 16 bit: 32581 | Float 32 bit: 2.622539E+38 | Octal: 177 105 114 106
Unsigned 16 bit: 32581 | Float 64 bit: 1.16843158668567E+305 | Binary: 01111111 01000101 01
□ Show little endian decoding □ Show unsigned as hexadecimal ASCII Text: ELF

Offset: 0x0 / 0x10ff Selection: None INS

Here is the first file (lab3-1) using bless

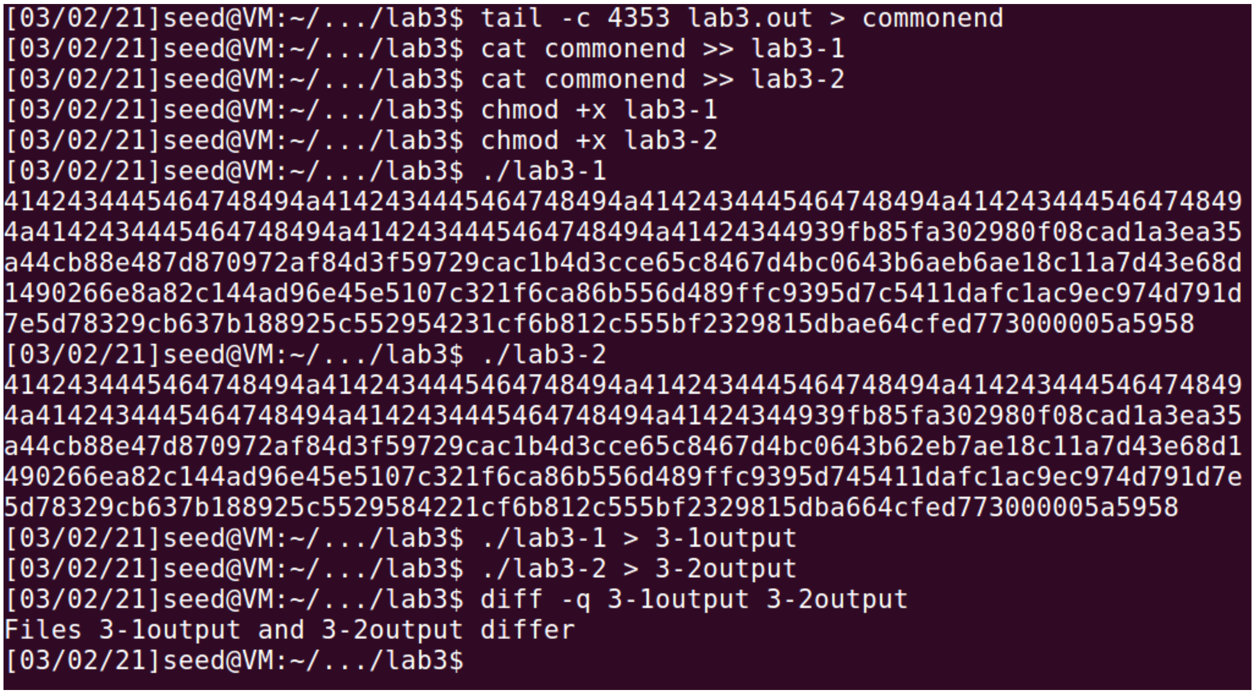


```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 .ELF..............
00000012 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 ......@...4.......
00000024 00 00 00 00 34 00 20 00 09 00 28 00 1F 00 1C 00 06 00 ....4. ...(.......
00000036 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 ..4...4...4... ...
00000048 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 54 01  ..............T.
0000005a 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 ..T...T...........
0000006c 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 ..................
0000007e 04 08 00 80 04 08 2C 06 00 00 2C 06 00 00 05 00 00 00 ......,...,.......
```

Signed 8 bit: 127 | Signed 32 bit: 2135247942 | Hexadecimal: 7F 45 4C 46
Unsigned 8 bit: 127 | Unsigned 32 bit: 2135247942 | Decimal: 127 069 076 070
Signed 16 bit: 32581 | Float 32 bit: 2.622539E+38 | Octal: 177 105 114 106
Unsigned 16 bit: 32581 | Float 64 bit: 1.16843158668567E+305 | Binary: 01111111 01000101 01
□ Show little endian decoding □ Show unsigned as hexadecimal ASCII Text: ELF

Loaded file '/home/seed/CMP... Offset: 0x0 / 0x10ff Selection: None INS

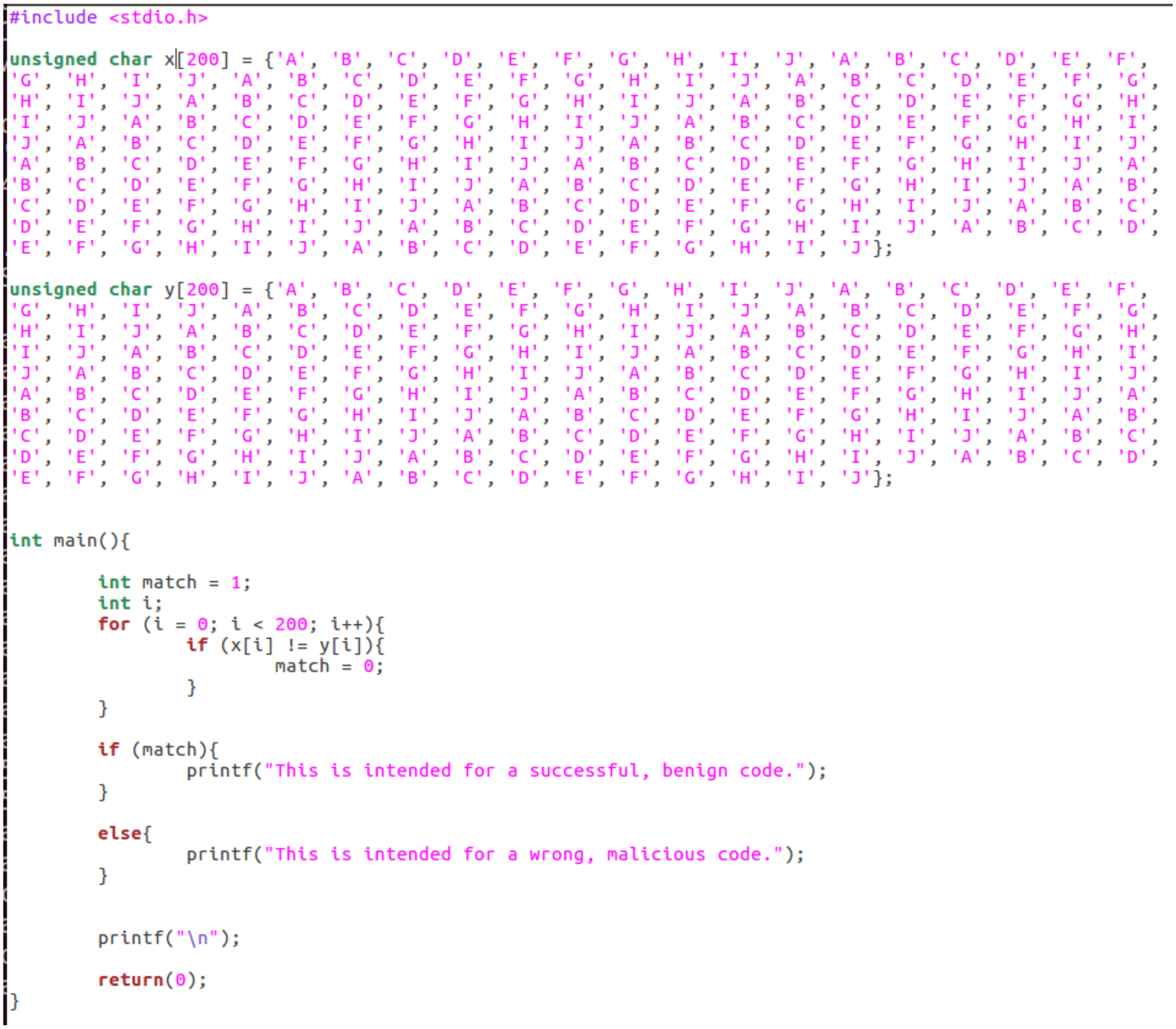
Here is the first file (lab3-2) using bless



```
[03/02/21]seed@VM:~/.../lab3$ tail -c 4353 lab3.out > commonend
[03/02/21]seed@VM:~/.../lab3$ cat commonend >> lab3-1
[03/02/21]seed@VM:~/.../lab3$ cat commonend >> lab3-2
[03/02/21]seed@VM:~/.../lab3$ chmod +x lab3-1
[03/02/21]seed@VM:~/.../lab3$ chmod +x lab3-2
[03/02/21]seed@VM:~/.../lab3$ ./lab3-1
4142434445464748494a4142434445464748494a4142434445464748494a41424344454647484
94a4142434445464748494a4142434445464748494a41424344939fb85fa302980f08cad1a3ea35
a44cb88e487d870972af84d3f59729cac1b4d3cce65c8467d4bc0643b6aeb6ae18c11a7d43e68d
1490266e8a82c144ad96e45e5107c321f6ca86b556d489ffc9395d7c5411dafc1ac9ec974d791d
7e5d78329cb637b188925c552954231cf6b812c555bf2329815dbae64cfed773000005a5958
[03/02/21]seed@VM:~/.../lab3$ ./lab3-2
4142434445464748494a4142434445464748494a4142434445464748494a41424344454647484
94a4142434445464748494a4142434445464748494a41424344939fb85fa302980f08cad1a3ea35
a44cb88e47d870972af84d3f59729cac1b4d3cce65c8467d4bc0643b62eb7ae18c11a7d43e68d1
490266ea82c144ad96e45e5107c321f6ca86b556d489ffc9395d745411dafc1ac9ec974d791d7e
5d78329cb637b188925c5529584221cf6b812c555bf2329815dba664cfed773000005a5958
[03/02/21]seed@VM:~/.../lab3$ ./lab3-1 > 3-1output
[03/02/21]seed@VM:~/.../lab3$ ./lab3-2 > 3-2output
[03/02/21]seed@VM:~/.../lab3$ diff -q 3-1output 3-2output
Files 3-1output and 3-2output differ
[03/02/21]seed@VM:~/.../lab3$
```

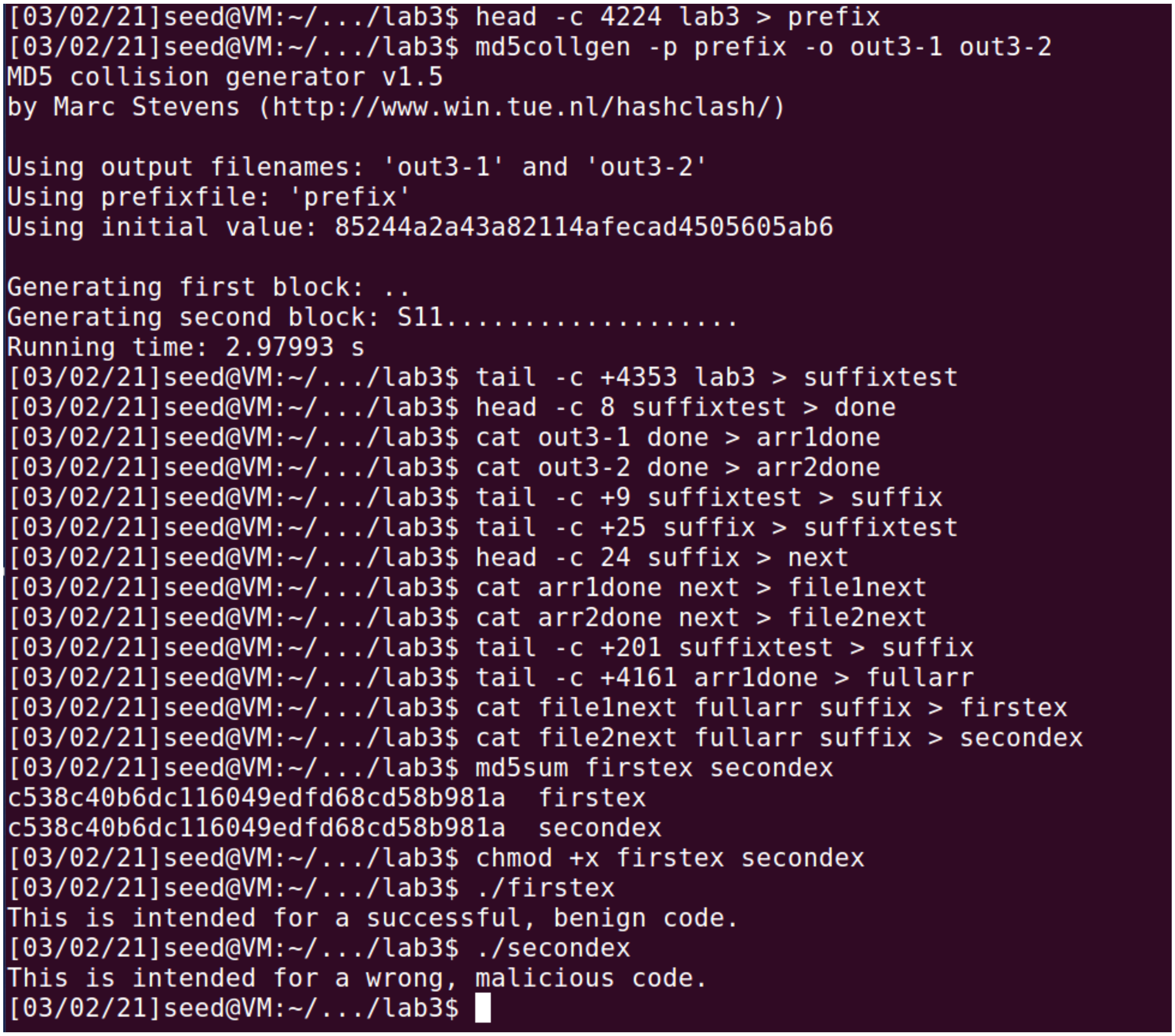
Finally, we can see the different outputs for each file (I needed to get a permission to run the executables here). But as you can see, the two outputs differ even though they have the same MD5 hash.

Task 4



```c
#include <stdio.h>

unsigned char x[200] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F',
'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A',
'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B',
'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D',
'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};

unsigned char y[200] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F',
'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A',
'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B',
'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D',
'E', 'F', 'G', 'H', 'I', 'J', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};


int main(){

        int match = 1;
        int i;
        for (i = 0; i < 200; i++){
                if (x[i] != y[i]){
                        match = 0;
                }
        }

        if (match){
                printf("This is intended for a successful, benign code.");
        }

        else{
                printf("This is intended for a wrong, malicious code.");
        }


        printf("\n");

        return(0);
}
```

This is the code that is used for task 4. Essentially takes in the two arrays, compares the values, if they match, it will say successful and benign code, otherwise will print out malicious code.



```
[03/02/21]seed@VM:~/.../lab3$ head -c 4224 lab3 > prefix
[03/02/21]seed@VM:~/.../lab3$ md5collgen -p prefix -o out3-1 out3-2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out3-1' and 'out3-2'
Using prefixfile: 'prefix'
Using initial value: 85244a2a43a82114afecad4505605ab6

Generating first block: ..
Generating second block: S11..................
Running time: 2.97993 s
[03/02/21]seed@VM:~/.../lab3$ tail -c +4353 lab3 > suffixtest
[03/02/21]seed@VM:~/.../lab3$ head -c 8 suffixtest > done
[03/02/21]seed@VM:~/.../lab3$ cat out3-1 done > arr1done
[03/02/21]seed@VM:~/.../lab3$ cat out3-2 done > arr2done
[03/02/21]seed@VM:~/.../lab3$ tail -c +9 suffixtest > suffix
[03/02/21]seed@VM:~/.../lab3$ tail -c +25 suffix > suffixtest
[03/02/21]seed@VM:~/.../lab3$ head -c 24 suffix > next
[03/02/21]seed@VM:~/.../lab3$ cat arr1done next > file1next
[03/02/21]seed@VM:~/.../lab3$ cat arr2done next > file2next
[03/02/21]seed@VM:~/.../lab3$ tail -c +201 suffixtest > suffix
[03/02/21]seed@VM:~/.../lab3$ tail -c +4161 arr1done > fullarr
[03/02/21]seed@VM:~/.../lab3$ cat file1next fullarr suffix > firstex
[03/02/21]seed@VM:~/.../lab3$ cat file2next fullarr suffix > secondex
[03/02/21]seed@VM:~/.../lab3$ md5sum firstex secondex
c538c40b6dc116049edfd68cd58b981a  firstex
c538c40b6dc116049edfd68cd58b981a  secondex
[03/02/21]seed@VM:~/.../lab3$ chmod +x firstex secondex
[03/02/21]seed@VM:~/.../lab3$ ./firstex
This is intended for a successful, benign code.
[03/02/21]seed@VM:~/.../lab3$ ./secondex
This is intended for a wrong, malicious code.
[03/02/21]seed@VM:~/.../lab3$ █
```

We see a successful attempt here because each executable gives the different message, one being a good message, the other bad. Essentially, we are completing the different arrays, taking the prefix, suffix, etc. and moving them around to the various files. In some cases, we copy the content of one file to another, but the end goal is to run the executables and get the two different messages. I completed all the steps and have a good understanding of what this task is aiming to do. This is how the MD5 collision vulnerability can be exploited.