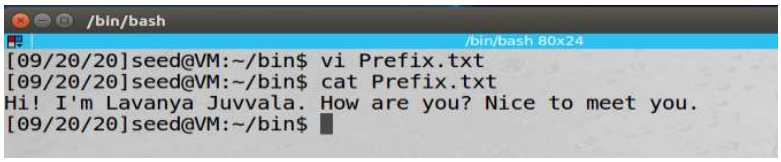
2.1 Task 1: Generating Two Different Files with the Same MD5 Hash

- Prefix file is created using the command `vi Prefix.txt`
- Displayed the contents of the file using `cat Prefix.txt`

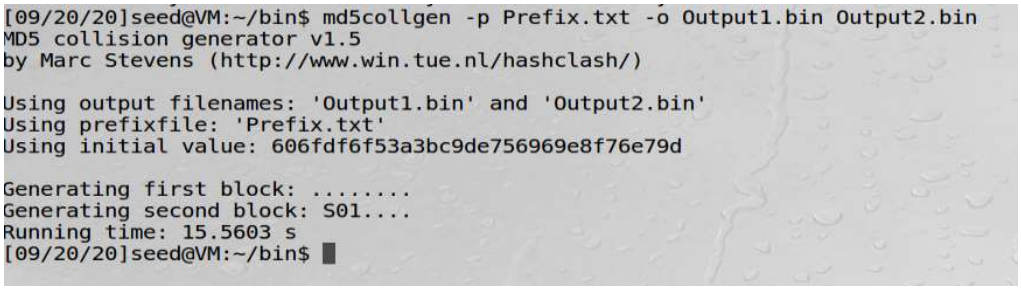


Question 1: If the length of your prefix file is not a multiple of 64, what is going to happen?

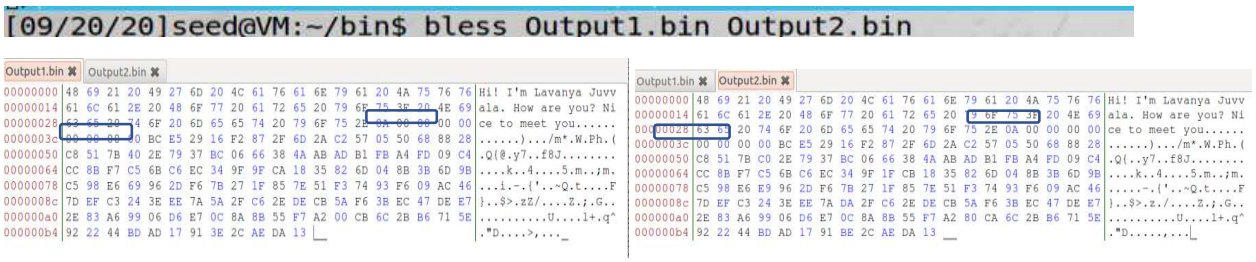
Answer: To test this out,

- I created a **Prefix.txt** file of size 56 bytes which is not a multiple of 64 . To verify this, check Properties of the file.

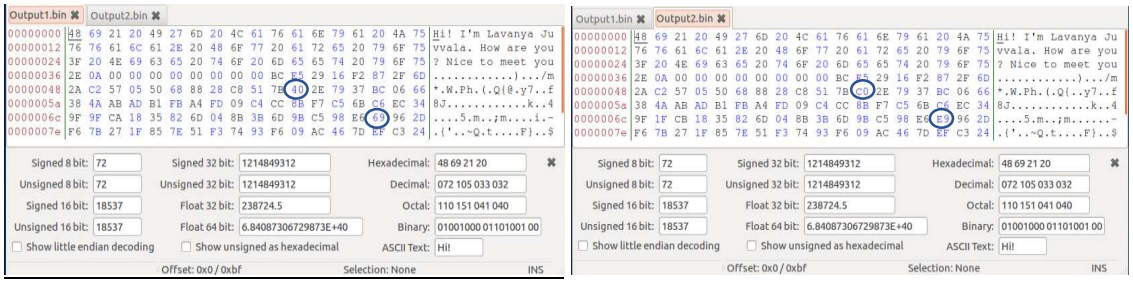
- Now, run `md5collgen -p Prefix.txt -o Output1.bin Output2.bin` with the above Prefix file .



- In the hex files of the output files , it is observed that Zero padding is added as the length of the prefix is not a multiple of 64. Using the command `bless Output1.bin Output2.bin`



- The differences in the two output files seen with Bless:



- Checking if the two output files are different using the `diff Output1 Output2` command. With `md5sum` command we can see that both the files have the same hash value.

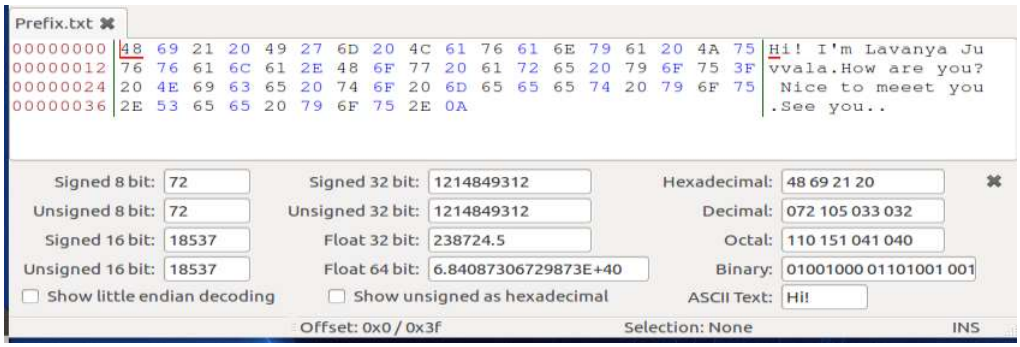
```
[09/20/20]seed@VM:~/bin$ diff Output1.bin Output2.bin
Binary files Output1.bin and Output2.bin differ
[09/20/20]seed@VM:~/bin$ md5sum Output1.bin
3dd740d38e2b3832d1d062b770e0f45d  Output1.bin
[09/20/20]seed@VM:~/bin$ md5sum Output2.bin
3dd740d38e2b3832d1d062b770e0f45d  Output2.bin
[09/20/20]seed@VM:~/bin$
```

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

Answer :

- Use `vi Prefix.txt` to add few more characters and make the Prefix file exactly of 64 bytes . To verify this, check Properties of the file.

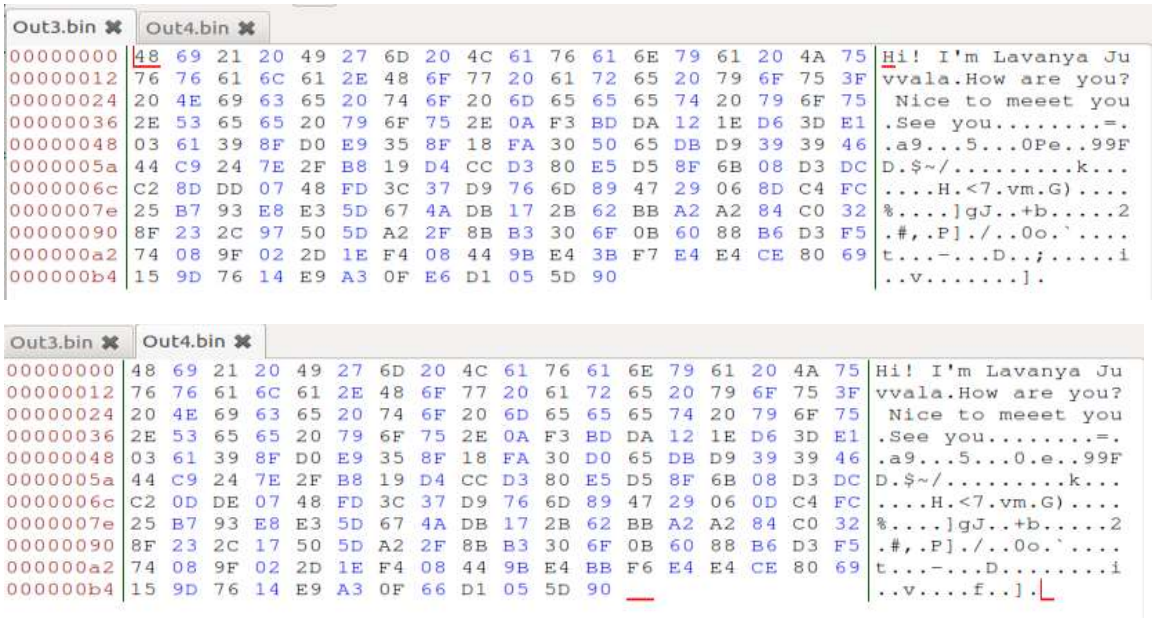
```
[09/20/20]seed@VM:~/bin$ vi Prefix.txt
[09/20/20]seed@VM:~/bin$
```



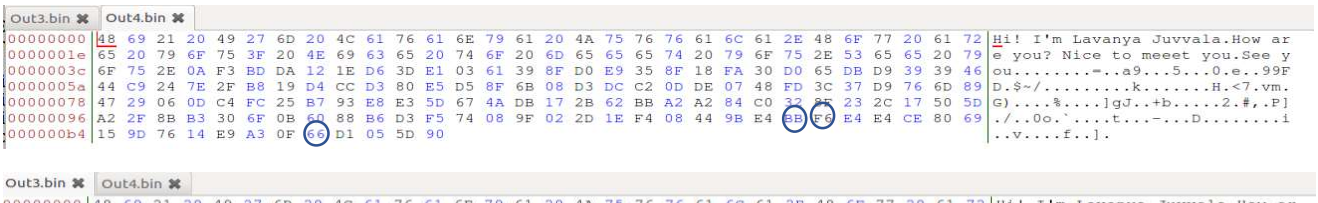
- . Run `md5collgen -p Prefix.txt -o Out3.bin Out4.bin` with the Prefix file to get two output files.

```
[09/20/20]seed@VM:~/bin$ md5collgen -p Prefix.txt -o Out3.bin Out4.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'Out3.bin' and 'Out4.bin'
Using prefixfile: 'Prefix.txt'
Using initial value: 28c72851761d1c0eadf40eb5a24fc6ce

Generating first block: ................................
Generating second block: S10................................
Running time: 106.012 s
```

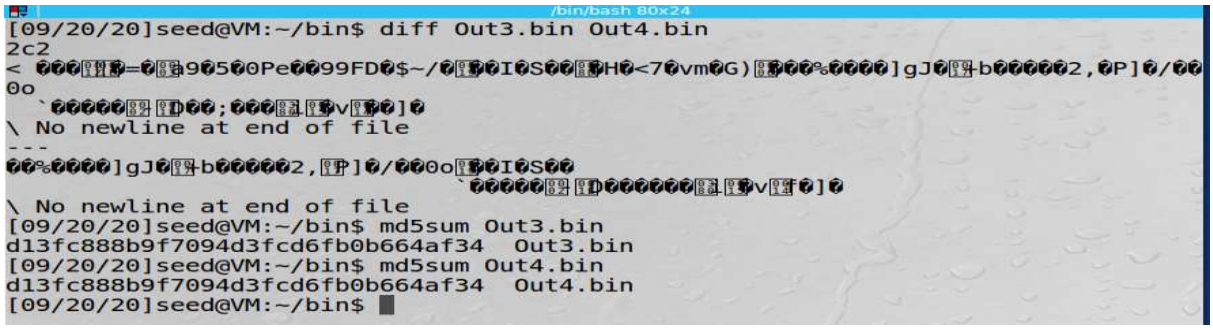
- There is no Zeroes padding in the output Hex file of ***Out3.bin Out4.bin*** this time.



- The differences in the two output files as seen in the *Bless:*

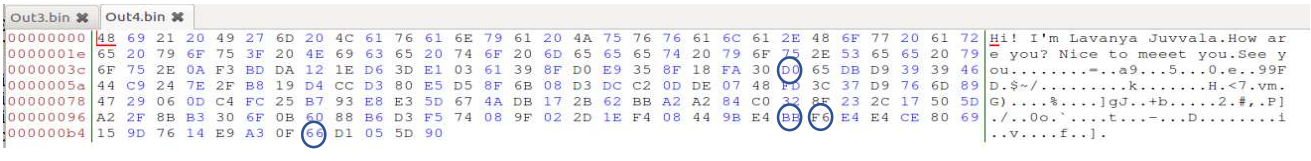


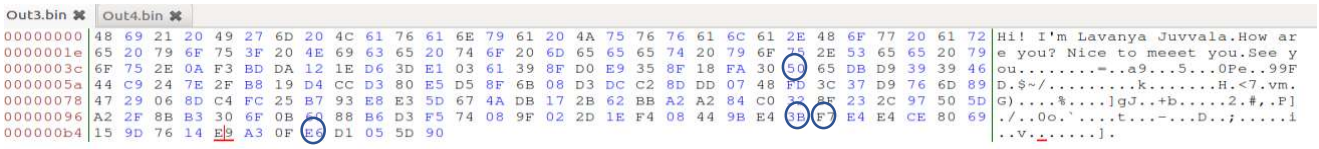
- It's observed that the two output files are different using the *diff* command .But by using md5sum command ,we can see that their MD5 hash values are the same.



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

Answer: No All bits are not Different. Some are similar. After multiple runs, it is noted that these differences are not constant.





2.2 Task 2: Understanding MD5's Property

- Run the md5collgen program generating two output files J.txt and L.txt:

```
[09/20/20]seed@VM:~/bin$ md5collgen -p Prefix.txt -o J.txt L.txt
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'J.txt' and 'L.txt'
Using prefixfile: 'Prefix.txt'
Using initial value: 28c72851761d1c0eadf40eb5a24fc6ce

Generating first block: ........................................
Generating second block: S01.......
Running time: 101.274 s
```

- We can see that the MD5 hash values of these two files are the same:

```
[09/20/20]seed@VM:~/bin$ md5sum J.txt
bdc4cc32bfda67a41a72c76cda20fa6b  J.txt
[09/20/20]seed@VM:~/bin$ md5sum L.txt
bdc4cc32bfda67a41a72c76cda20fa6b  L.txt
```

- Now concatenating K.txt to both J.txt and L.txt and save the output in JK.txt and LK.txt respectively

```
[09/20/20]seed@VM:~/bin$ cat J.txt K.txt > JK.txt
[09/20/20]seed@VM:~/bin$ cat L.txt K.txt > LK.txt
```

- We can see that even after concatenating a file T.txt to these two files, the MD5 hash value of the new files will be equal:

```
[09/20/20]seed@VM:~/bin$ md5sum JK.txt
fc6b49c7bbd3dc0dded980363832d283  JK.txt
[09/20/20]seed@VM:~/bin$ md5sum LK.txt
fc6b49c7bbd3dc0dded980363832d283  LK.txt
```

- Thus, we can conclude that,
  Given two inputs J and L if MD5(J) = MD5(L), i.e., the MD5 hashes of J and L are the same, then for any input K, MD5(J || K) = MD5(L || K), where || represents concatenation.

2.3 Task 3: Generating Two Executable Files with the Same MD5 Hash

- The code:

```c
#include <stdio.h>

unsigned char a[200] = { 'A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A'
,'A','A','A','A','A','A'};

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

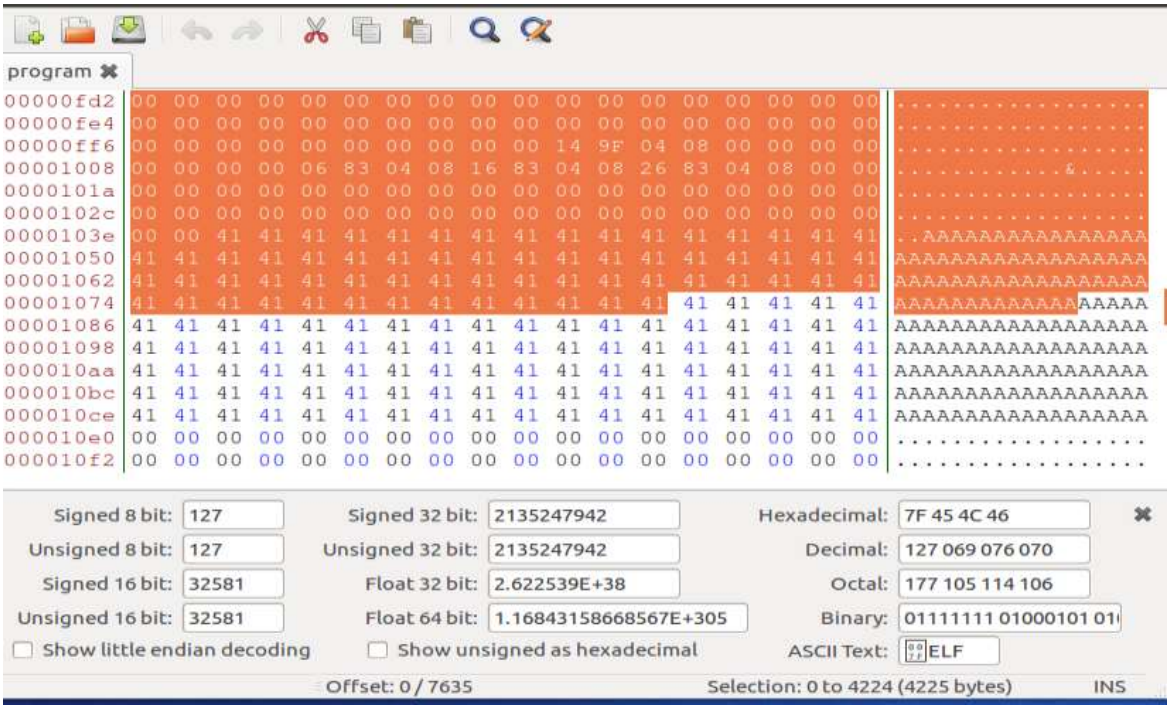
- The array a[200] is filled with 200 A's so that they can easily be located in the hex file. The below picture shows a clip of the hex file.

```
0000103e 00 00 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 ..AAAAAAAAAAAAAAAA
00001050 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001062 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001074 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001086 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001098 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010aa 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010bc 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010ce 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010e0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010f2 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001104 41 41 41 41 47 43 43 3A 20 28 55 62 75 6E 74 75 20 35 AAAAGCC: (Ubuntu 5
00001116 2E 34 2E 30 2D 36 75 62 75 6E 74 75 31 7E 31 36 2E 30 .4.0-6ubuntu1~16.0
00001128 34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 36 30 4.4) 5.4.0 2016060
0000113a 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 9.................
0000114c 00 00 00 00 54 81 04 08 00 00 00 00 03 00 01 00 00 00 ....T.............
0000115e 00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 00 ..h...............
00001170 88 81 04 08 00 00 00 00 03 00 03 00 00 00 00 00 AC 81 ..................
00001182 04 08 00 00 00 00 03 00 04 00 00 00 00 00 CC 81 04 08 ..................
00001194 00 00 00 00 03 00 05 00 00 00 00 00 2C 82 04 08 00 00 ..............,...
```

- Run the code to get the executable file :

```
[09/21/20]seed@VM:~$ cd bin
[09/21/20]seed@VM:~/bin$ vi program.c
[09/21/20]seed@VM:~/bin$ gcc program.c -o program
[09/21/20]seed@VM:~/bin$ bless program
```

- See the Hex file program using bless program. Now, select offset of the file including some parts of the array.



- Here, I have selected first 4224 bytes(a multiple of 64) using the command : `head -c 4224 program > prefix` and save it to file prefix.
- Now, skip 128 bytes and include the content from 4224 +128 = 4352 to end of the file.
- Use the command `tail -c +4352 program > suffix` and save it to suffix file.

```
[09/21/20]seed@VM:~/bin$ head -c 4224 program > prefix
[09/21/20]seed@VM:~/bin$ tail -c +4352 program > suffix
```

- Run md5collgen program to get two output files Out1.bin Out2.bin

```
[09/21/20]seed@VM:~/bin$ md5collgen -p prefix -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'
Using initial value: f60d6a67bfa8f078e4e4fba3c032cf51

Generating first block: ..........................................................
..........................................................................
.....................................
Generating second block: S10........
Running time: 423.84 s
[09/21/20]seed@VM:~/bin$
```

- Clip the last 128 bytes from both the output files and save them in P and Q

```
[09/21/20]seed@VM:~/bin$  tail -c 128 Out1.bin > P
[09/21/20]seed@VM:~/bin$ tail -c 128 Out2.bin > Q
[09/21/20]seed@VM:~/bin$
```

- Now, concatenate the prefix+ P + suffix to give the first program program1.out
- Also, concatenate the prefix + Q + suffix to form the second program program2.out

```
[09/21/20]seed@VM:~/bin$ cat prefix P suffix > program1.out
[09/21/20]seed@VM:~/bin$ cat prefix Q suffix > program2.out
[09/21/20]seed@VM:~/bin$ chmod a+x program1.out program2.out
[09/21/20]seed@VM:~/bin$
```

- Using diff command, it can be proved that the MD5 hash values of the two programs are same even though the programs are different.

```
[09/21/20]seed@VM:~/bin$ diff program1.out program2.out
Binary files program1.out and program2.out differ
[09/21/20]seed@VM:~/bin$ md5sum program1.out
c94d4b9e0be7d3709dd78175238013ed  program1.out
[09/21/20]seed@VM:~/bin$ md5sum program2.out
c94d4b9e0be7d3709dd78175238013ed  program2.out
[09/21/20]seed@VM:~/bin$
```

- By running the two programs we notice that they run smoothly. This is because only the contents of the array were changed and not the code of the program.
  The content of both the arrays is displayed.

```
[09/21/20]seed@VM:~/bin$ program1.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141 84ad49cb11b9366222ddd38adade15d6
7fbb4e9d7f535bc5d2681b56838c423355ebe25e263ed9cf2143d4f0156d7fc497f87cf446eb9921
02ede455254523a14fb4264d2278c1cc9671437fc1fcd1143a84c5773ff2db3106f3b6068c6cbbd2
0549f445374b56c49a20f8fad236f5fbac1b30c4f695898fa4adeb3cdc00000000
[09/21/20]seed@VM:~/bin$ program2.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141414141414141414141414184ad49cb11b9366222ddd38adade15d6
7fbb469d7f535bc5d2681b56838c423355ebe25e263ed9cf2143d4f0956d7fc497f87cf446eb9921
02e5e455254523a14fb4264d2278c1cc9671437fc1fcd1143284c5773ff2db3106f3b6068c6cbbd2
0549f445374b56c49a2078fad236f5fbac1b30c4f695898f24adeb3cdc00000000
[09/21/20]seed@VM:~/bin$
```

- As P and Q differ only by a few bits, the outputs look similar. But there is some difference in the output(marked above).

2.4 Task 4: Making the two programs behave differently

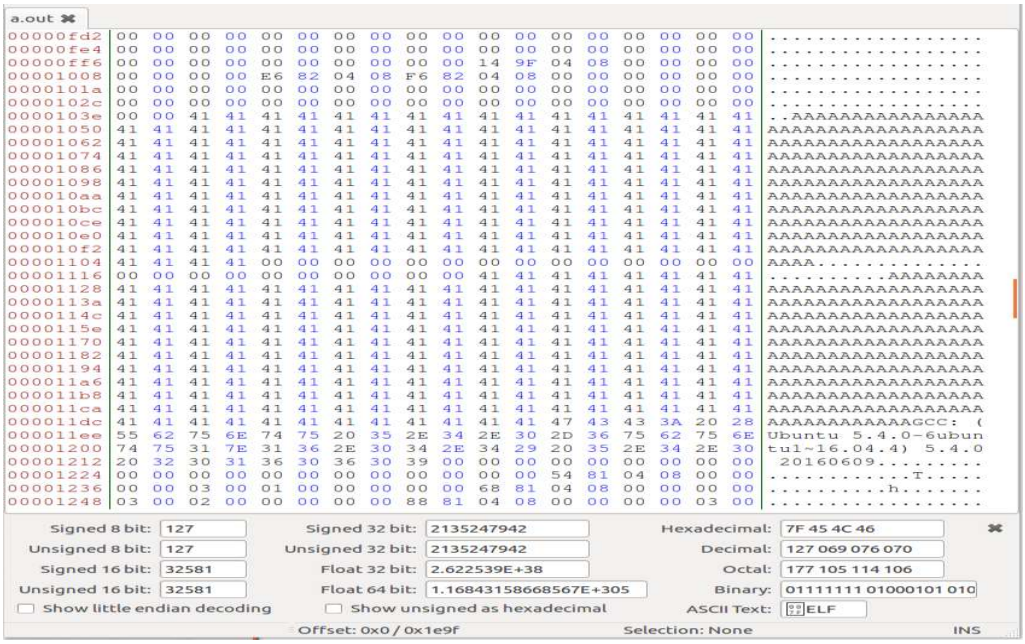
- The file sampleprogram.c which contains two arrays *a[], b[]* filled with 200 A's

```c
#include <stdio.h>
#define LEN 200

unsigned char a[] = { 'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A'};

unsigned char b[] = { 'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A','A',
'A','A','A','A','A'};

int main()
{
        int i = 0;
        for(i = 0; i < LEN; i++)
        {
                if(a[i]!=b[i])
                        break;
        }
        if(i == LEN){
                printf("%s\n","Executing the benign code");
         }
        else {
                printf("%s\n","Executing the malicious code");
         }
return 0;
}
~
~
~
:wq
```

- There are two parts to this program : if both the arrays have same content then benign code gets executed else if the content is different then the malicious code gets executed .
- Compile the program with the command `gcc sampleprogram.c` and produce the `a.out` file.

```
[09/22/20]seed@VM:~/bin$ gcc sampleprogram.c
[09/22/20]seed@VM:~/bin$ bless a.out
```

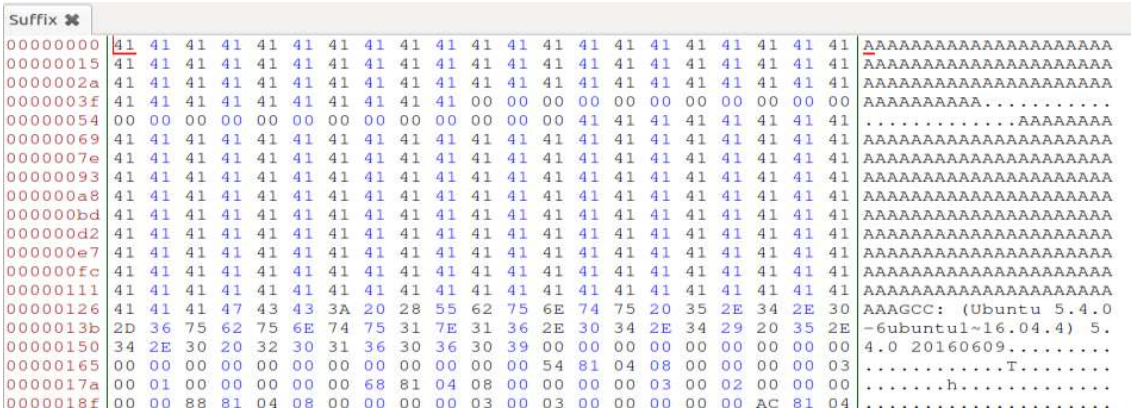
- Displaying the contents of a.out file.



- The bytes from 0 to 4160 are saved in *Prefix* and the bytes from 4288 till the end of the file are saved in *Suffix*.
- Run the md5collgen program to get two output hex files *O1.bin, O2.bin*
  Now, save the last 128 bytes of each output file to P and Q respectively.

```
[09/22/20]seed@VM:~/bin$ gcc sampleprogram.c
[09/22/20]seed@VM:~/bin$ head -c 4160 a.out > Prefix
[09/22/20]seed@VM:~/bin$ tail -c +4288 a.out > Suffix
[09/22/20]seed@VM:~/bin$
[09/22/20]seed@VM:~/bin$ md5collgen -p Prefix -o O1.bin O2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'O1.bin' and 'O2.bin'
Using prefixfile: 'Prefix'
Using initial value: 4a92a182f9362b9c17aa07a68651bfb1

Generating first block: .............
Generating second block: S01...
Running time: 29.1013 s
[09/22/20]seed@VM:~/bin$ tail -c 128 O1.bin > P
[09/22/20]seed@VM:~/bin$ tail -c 128 O2.bin > Q
[09/22/20]seed@VM:~/bin$
```

- Viewing the `Suffix` file:



- Divide the Suffix file into two halves: *Suffix1, Suffix2*
- The first part of the Suffix is the first 96 bytes saved in Suffix1.
- And the second part will be from offset 96+128=224 till the end of the file which is saved in Suffix2.

```
[09/22/20]seed@VM:~/bin$ head -c 96 Suffix > Suffix1
[09/22/20]seed@VM:~/bin$ tail -c +224 Suffix > Suffix2
[09/22/20]seed@VM:~/bin$
[09/22/20]seed@VM:~/bin$
```

Concatenate everything together using:

- cat *Prefix, P, Suffix1, P, Suffix2* and save it in benign.out
- cat *Prefix, Q, Suffix1, P, Suffix2* and save it in malicious.out

```
[09/22/20]seed@VM:~/bin$ cat Prefix P Suffix1 P Suffix2 > benign.out
[09/22/20]seed@VM:~/bin$ cat Prefix Q Suffix1 P Suffix2 > malicious.out
[09/22/20]seed@VM:~/bin$ chmod a+x benign.out malicious.out
[09/22/20]seed@VM:~/bin$
```

- Now run the two programs, benign.out and malicious.out

```
[09/22/20]seed@VM:~/bin/lav$ benign.out
Executing the benign code
[09/22/20]seed@VM:~/bin/lav$ malicious.out
Executing the malicious code
[09/22/20]seed@VM:~/bin/lav$ md5sum benign.out
6b263849043f4c79e73a92af25fc73ee  benign.out
[09/22/20]seed@VM:~/bin/lav$ md5sum malicious.out
6b263849043f4c79e73a92af25fc73ee  malicious.out
[09/22/20]seed@VM:~/bin/lav$
```

Thus, one program executes benign code and other one executes malicious code but the md5sum proves that the hash of both the programs is same.
