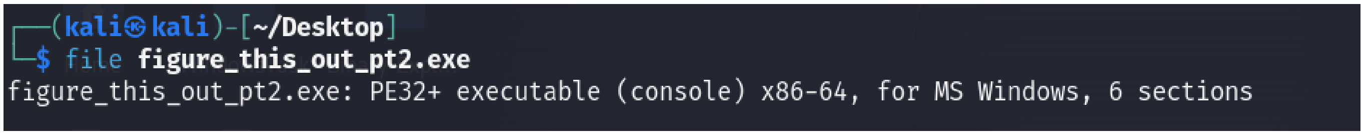
Windows - Task 3

Name : Ishayu Potey
College : VJTI SY Btech - Electronics
Unique ID : LTXBGRJ6

Aim : Reverse engineer the executable to determine what the **original input string** was, given the encoded output.

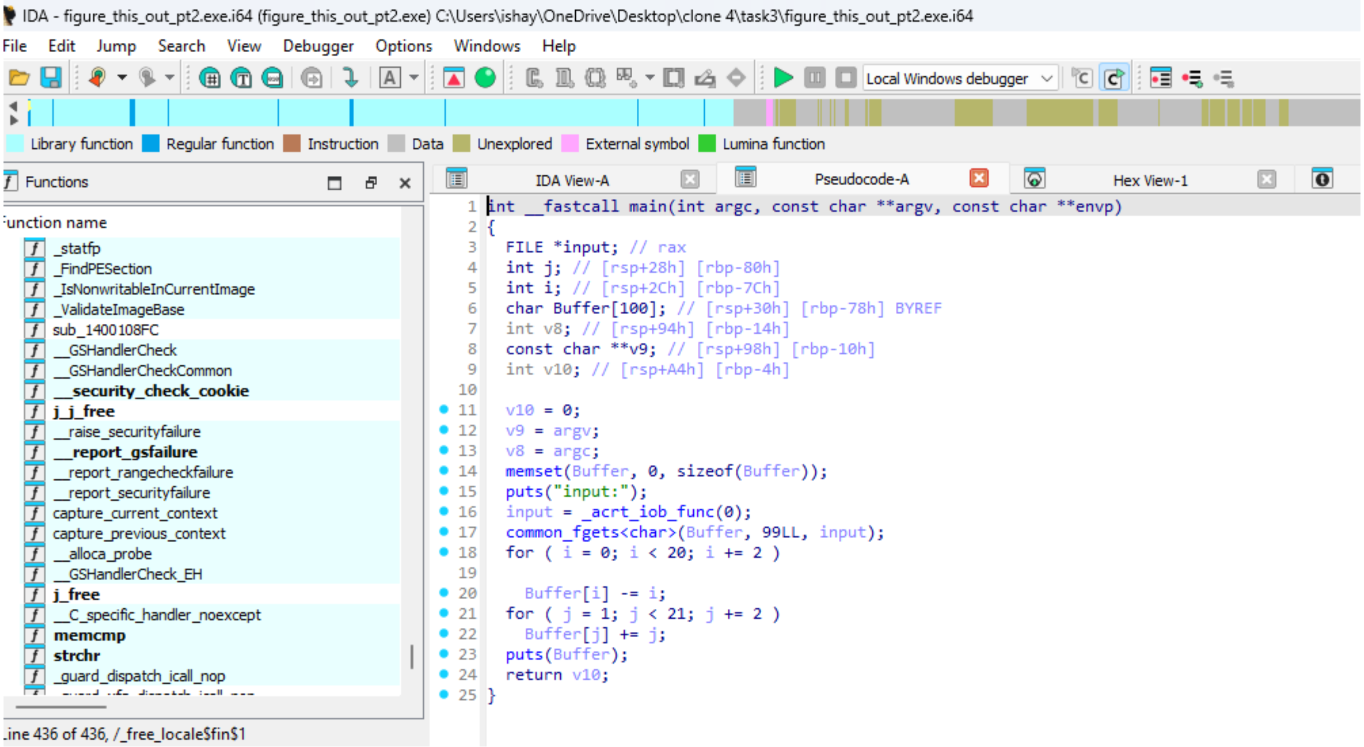
The encoded output string provided is: `WPUbutof1xJj+U%nVB"L`

In this we have an **Windows Portable Executable** File (PE) with 64-bit Architecture



```
┌──(kali㉿kali)-[~/Desktop]
└─$ file figure_this_out_pt2.exe
figure_this_out_pt2.exe: PE32+ executable (console) x86-64, for MS Windows, 6 sections
```

- We then Open the `figure_this_out_pt2.exe` File in IDA Decompiler.
- Opening the Pseudocode for following executable binary we get this code



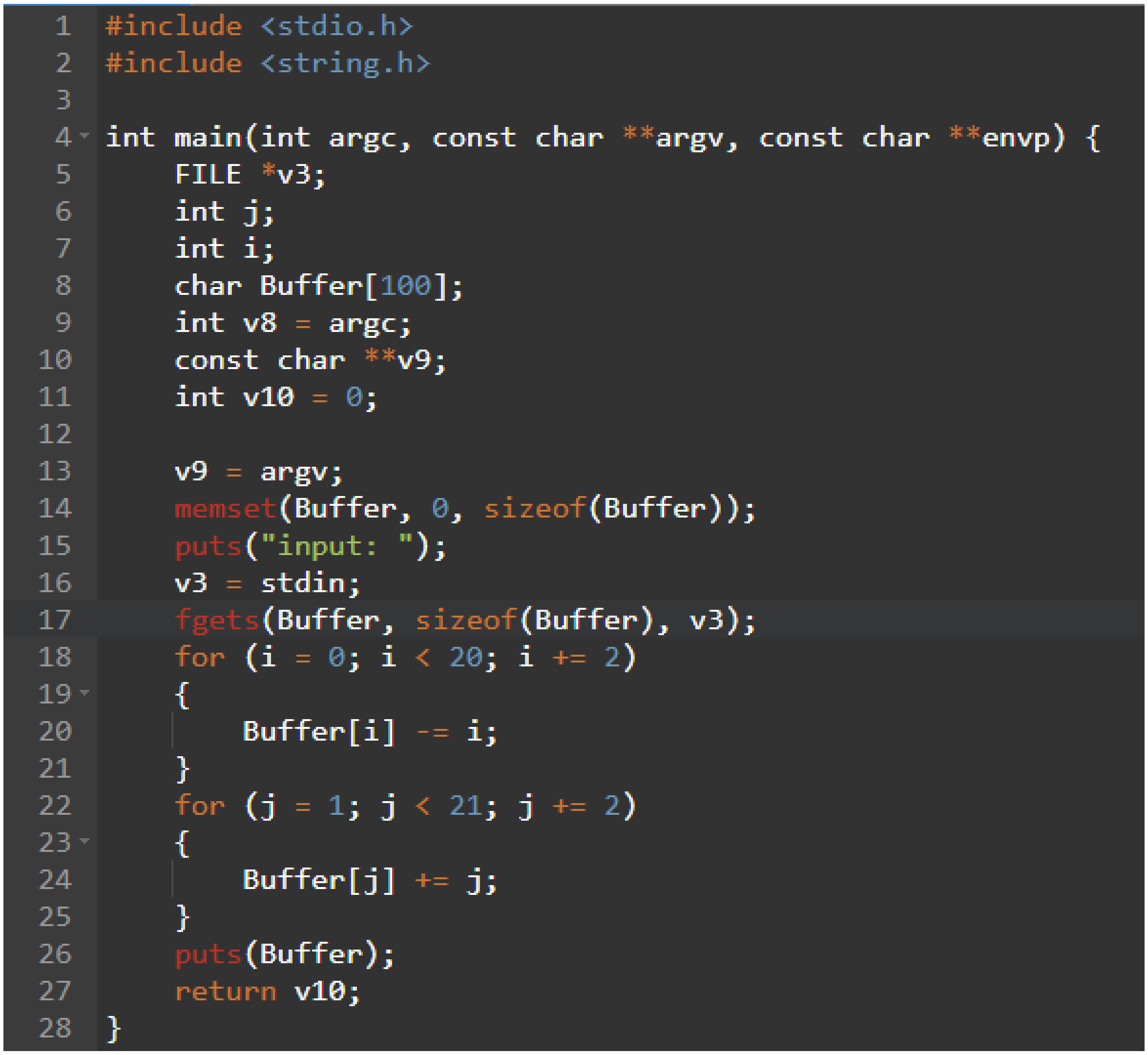
```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  FILE *input; // rax
  int j; // [rsp+28h] [rbp-80h]
  int i; // [rsp+2Ch] [rbp-7Ch]
  char Buffer[100]; // [rsp+30h] [rbp-78h] BYREF
  int v8; // [rsp+94h] [rbp-14h]
  const char **v9; // [rsp+98h] [rbp-10h]
  int v10; // [rsp+A4h] [rbp-4h]

  v10 = 0;
  v9 = argv;
  v8 = argc;
  memset(Buffer, 0, sizeof(Buffer));
  puts("input:");
  input = _acrt_iob_func(0);
  common_fgets<char>(Buffer, 99LL, input);
  for ( i = 0; i < 20; i += 2 )

    Buffer[i] -= i;
  for ( j = 1; j < 21; j += 2 )
    Buffer[j] += j;
  puts(Buffer);
  return v10;
}
```

This is a raw code which we have obtained.

Analyzing Assembly and Pseudocode

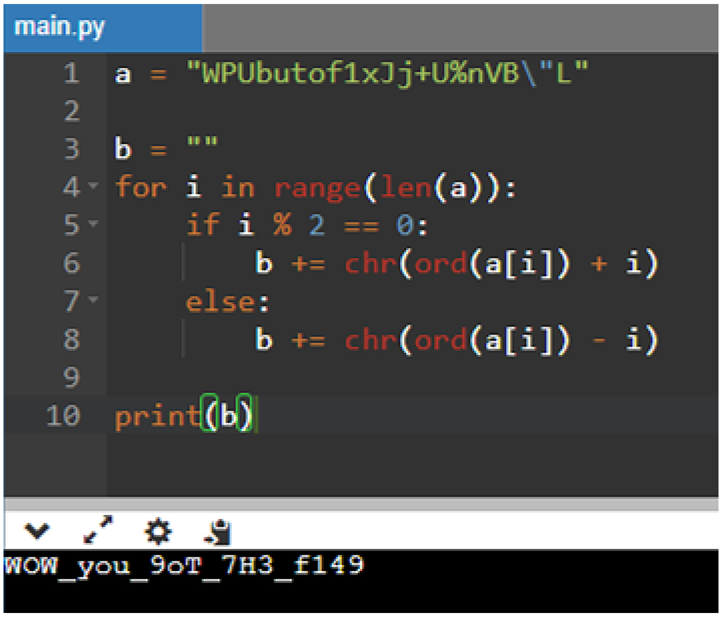
After Analyzing the pseudocode and understanding how its Assembly is working we write a **Working Code** similar to Pseudocode to Analyze the **Mechanism of String Encoding**

This is the **C - Code** we get , I have used GDB online compiler here



```c
#include <stdio.h>
#include <string.h>

int main(int argc, const char **argv, const char **envp) {
    FILE *v3;
    int j;
    int i;
    char Buffer[100];
    int v8 = argc;
    const char **v9;
    int v10 = 0;

    v9 = argv;
    memset(Buffer, 0, sizeof(Buffer));
    puts("input: ");
    v3 = stdin;
    fgets(Buffer, sizeof(Buffer), v3);
    for (i = 0; i < 20; i += 2)
    {
        Buffer[i] -= i;
    }
    for (j = 1; j < 21; j += 2)
    {
        Buffer[j] += j;
    }
    puts(Buffer);
    return v10;
}
```

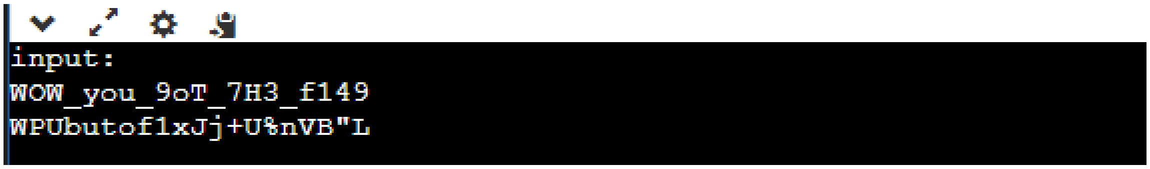
- The Basic Working of code lies in the Two For Loops which are encoding the input string
- Lets Try to Reverse Engineer this For loops in python.
- We will only take one specific input as this string `WPUbutof1xJj+U%nVB"L`



```python
a = "WPUbutof1xJj+U%nVB\"L"

b = ""
for i in range(len(a)):
    if i % 2 == 0:
        b += chr(ord(a[i]) + i)
    else:
        b += chr(ord(a[i]) - i)

print(b)
```

```
WOW_you_9oT_7H3_f149
```

There We got our **Flag : WOW_you_9oT_7H3_f149**
Lets Verify it once again with our C Code which we had made earlier.



```
input:
WOW_you_9oT_7H3_f149
WPUbutof1xJj+U%nVB"L
```

We get our original String back , This concludes our C - Code Algorithm was on Point

Flag : WOW_you_9oT_7H3_f149