CS0449 Project2

1. Program1
    a. Passphrase: rxXQhNMkLKfuoIjeJJRYj
    b. Attempts:
        i. mystrings tim40_1//A lot of characters. Seems useless.
    c. Solution:
        i. gdb tim40_1//Load program into the debugger
        ii. b main//Set a breakpoint at main function
        iii. r//Run the program
        iv. disas// Display the assembler code
        v. b *0x0804830c//I noticed that there is a function “fget” which can get input, so edi would be my own input. There is also a line of code compare the input edi with a esi, so I set a breakpoint at that address

```
0x080482f2 <+35>:    mov    %edi,(%esp)
0x080482f5 <+38>:  call   0x80493f0 <fgets>

0x0804830c <+61>:  repz cmpsb %es:(%edi),%ds:(%esi)
```

        vi. c//Continue
        vii. something//My own input
        viii. x $edi//It is the my own input ”something”
        ix. x $esi//It is a string “rxXQhNMkLKfuoIjeJJRYj”
        x. r//Restart the program
        xi. rxXQhNMkLKfuoIjeJJRYj//Input “rxXQhNMkLKfuoIjeJJRYj”
        xii. Unlocked

2. Program2
    a. Passphrase: any one character repeats at least 14 times, e.g. “66666666666666”
    b. Attempts:
        i. mystrings tim40_2//A lot of characters. Don’t want to read them.
    c. Solution:
        i. gdb tim40_2
        ii. b main
        iii. r
        iv. disas//I notice that there are three unknown functions and two “cmp”

```
0x0804851f <+45>:  call   0x804845f <c>

0x08048527 <+53>:  call   0x804849b <p>

0x08048538 <+70>:  call   0x8048444 <s>

0x0804852c <+58>:  cmp    $0x1,%eax

0x0804853d <+75>:  cmp    $0xd,%eax
```

        v. b c
        vi. b p
        vii. b s//Set breakpoints in these functions
        viii. //Read through the function s, I find it would use eax to return the length of ebx
        ix. //Function c’s usage is testing the last position of input string, if it is “\n”, then replace it with null, the return it using ebx
        x. //Function p’s usage is testing how much different characters in the string. If there is only one character repeating, return 1 using eax. If not, return 0.
        xi. //Then the “cmp” function would test if the eax equals to 1, if not, the program will end with “Not correct”.
        xii. //If eax equals to 1, then the program would call function s to measure the length again, then compares it with 0xd, which means 13
        xiii. //The “jle” following the “cmp” means that if eax is less than or equals to 13, program ends with “Not correct”. If not, unlocked

3. Program3
    a. Passphrase: exactly 8 characters within “1”, “2”, “3”, “4”, “5” in first 10 characters
    b. Attempts:
        i. mystrings tim40_3//Much fewer characters than before, show that this is a program using dynamic linking
        ii. gdb tim40_3
            1. b main//There is no main function in this program
        iii. objdump -D tim40_3
            1. //Notice that there is a “push” operation in <.dynamic>, so set a breakpoint at that address using gdb

```
8049710:  06                    push   %es
```

            2. gdb tim40_3
            3. b *0x8049710
            4. //Find a function called “do_lookup_x”. But it has thousands lines of code and it doesn’t have something input. Give up this way
            5. //Find a function called “check_match.8631”. It doesn’t have input either. Give up
    c. Solution:
        i. objdump -D tim40_3
        ii. //Notice there is “getchar” function. And there exist “printf” and “puts” functions which are exactly the cases mean failure or unlock in program1 and program2.

```
804846e:  e8 c5 fe ff ff        call   8048338 <getchar@plt>

80484d4:  e8 8f fe ff ff        call   8048368 <printf@plt>

80484e2:  e8 91 fe ff ff        call   8048378 <puts@plt>
```

        iii. //-0xc(%ebp) is a counter which count how many characters have been read. Each time read a character, -0xc(%ebp) plus 1, if it is less than or equals to 9, read again

```
804846b:  8b 5d f4           mov    -0xc(%ebp),%ebx
804846e:  e8 c5 fe ff ff        call   8048338 <getchar@plt>
8048473:  88 44 1d e5           mov    %al,-0x1b(%ebp,%ebx,1)
8048477:  83 45 f4 01           addl   $0x1,-0xc(%ebp)
804847b:  83 7d f4 09           cmpl   $0x9,-0xc(%ebp)
804847f:  7e ea              jle    804846b <tolower@plt+0xe3>
```

        iv. //After read first 10 characters, continue
        v. //Let eax be the ASCII of a single character, then subs 0x31, which is 49 in decimal system, compares to 4, if it is larger than 4, jump to address 80484b5. Since “ja” function compares unsigned number, only ‘0’ ‘1’ ‘2’ ‘3’ ‘4’ wouldn’t jump. Because subs 49 before, these characters are “1”, “2”, “3”, “4”, “5” in

ASCII

```
80484a8:  83 e8 31            sub    $0x31,%eax
80484ab:  83 f8 04            cmp    $0x4,%eax
80484ae:  77 05               ja     80484b5 <tolower@plt+0x12d>
```

vi. //If not jump, -0x10(%ebp) means the counter of characters meet the requirement. Add 1 to it, then continue, go to the same code as jumped

```
80484b0:  83 45 f0 01            addl   $0x1,-0x10(%ebp)
```

vii. //If jump, -0xc(%ebp) means the counter of all characters have been tested. Add 1 to it, if it is less than or equals to 0xa(10), jumps to address 8048492, which means change eax to next character in the string. If not, continue

```
80484b5:  83 45 f4 01            addl   $0x1,-0xc(%ebp)
80484b9:  83 7d f4 0a            cmpl   $0xa,-0xc(%ebp)
80484bd:  7e d3               jle    8048492 <tolower@plt+0x10a>
```

viii. //After test all first 10 characters, the program will reach here and compare the counter -0x10(%ebp) with 8. The function “jne” means that if the counter exactly equals to 8, continue. If not, jump to address 80484db, which means “Not correct”

```
80484bf:  83 7d f0 08            cmpl   $0x8,-0x10(%ebp)
80484c3:  75 16               jne    80484db <tolower@plt+0x153>
```

ix. //If the counter equals to 8, the program will continue and reach the “printf” function, which means unlocked.