Week 04

Deadline : Saturday, September 30th 2017

Preparation before lab

1. In this tutorial, we will playing with Pointer and I/O with some C program
2. Login to your badak account
3. Change your directory to "work" and create new directory named “work04” inside “work” directory

```
$ cd work
$ mkdir work04
```

4. Move all the file attached at scele along with this file to your work04 directory. Hint : use WinSCP or Tunnels. The files you need to move :
   (1) open-close.c
   (2) write.c

5. Change your directory to work04

```
$ cd work04
```

Global Variable Address vs Local Variable Address

1. In this tutorial, you will learn about global and local variable address.
2. First create a new C program. Create and **save** a file named **global-char.c**
3. Write this code to import **stdio.h**

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

Explanation:

"stdio.h" is a header file, where this and other similar functions are defined. stdio.h stands for standard input output header. And so you include and let complier know that you will need standard input output commands and he better include the code beforehand. (quora)

4. Then declare 4 **char-variable**, and assign them with your choosen character, example:

```
char varchar0='a';
char varchar1='b';
char varchar2='c';
char varchar3='d';
```

*it is recommended for you to name these variable as stated in the example above or you have to change all the respective variable names on the following codes.

5. Next, create the main function that print the value of the variable and its address. Add the program with the following code :

```
void main(void) {
        printf("=======================================\n");
        printf("varchar0: value = %2c, address = %p\n", varchar0, &varchar0);
        printf("varchar1: value = %2c, address = %p\n", varchar1, &varchar1);
        printf("varchar2: value = %2c, address = %p\n", varchar2, &varchar2);
        printf("varchar3: value = %2c, address = %p\n", varchar3, &varchar3);
        printf("=======================================\n");
}
```

6. Compile the program and run it.

```
$ gcc global-char.c -o global-char
$ ./global-char
```

7. If your program is correct, you will see the output which tells you the value and the address of each of your variables. Now run the **global-char** program and save the output to the result1.txt.

```
$ ./global-char > result1.txt
```

8. Run the program twice more and add the output to result1.txt. You can do that by repeat this step twice.

```
$ ./global-char >> result1.txt
```

*note that the (>>) sign in is different with (>). Remember that >> sign used for append text to a file, meanwhile > sign will replace all the text on the files.

9. You will need to see the result1.txt later. Now, create and **save** a new C program named **local-char.c**
It's actually a quite similar program, but this time move all the variable declaration into the main function. Your **local-char.c** should look like this:

```
#include <stdio.h>
void main(void) {
        char varchar0='a';
        char varchar1='b';
        char varchar2='c';
        char varchar3='d';

        printf("=======================================\n");
        printf("varchar0: value = %2c, address = %p\n", varchar0, &varchar0);
        printf("varchar1: value = %2c, address = %p\n", varchar1, &varchar1);
        printf("varchar2: value = %2c, address = %p\n", varchar2, &varchar2);
        printf("varchar3: value = %2c, address = %p\n", varchar3, &varchar3);
        printf("=======================================\n");
}
```

10. Compile the second program

```
$ gcc local-char.c -o local-char
```

11. Now run the program three times and write the output to result2.txt. You can do that by repeat the following command three times.

```
$ ./local-char >> result2.txt
```

12. Take a look on both programs output, and read again the code from both program. Take your time to learn about the flow and the output of the program.
13. Then you must answer some of question below and write it on **lab04.txt** point number 1 :
    a. What are the differences about the code and the output from both program?
    b. Why Global Variable Address number and Local Variable Address number have a great difference in their address?
    c. Write your knowledge about Global Variable Address and Local Variable Address (don't forget to add the sources if you use reference)
    (*warning : write the answer by your own, plagiarism rule applies)

Pointer

14. Next, we will learn about **pointer** in C language. We will explore how it works and how to implement it.
15. Create and **save** a new C program named **00-pointer-basic.c**
16. Just like the earlier program, write this code to import **stdio.h**

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

17. Then declare 4 **char-variable**, and assign them with your choosen character. (you can copy the variable from the previous code).

```
char varchar0='a';
char varchar1='b';
char varchar2='c';
char varchar3='d';
```

*its recommended for you to name these variable as stated in the example above or you have to change all the respective variable names on the following codes.

18. Now it's time to declare the **pointer-variable**, add the following code:

```
char* ptrchr0=&varchar0;
char* ptrchr1=&varchar1;
char* ptrchr2=&varchar2;
char* ptrchr3=&varchar3;
```

Explanation:

a. These code means, we declare a "**char**" "**pointer-variable**" named "**ptrchr0**".
b. The star(*) means that this variable is a **pointer-variable**. **pointer-variable** store an address.
c. **char*** means that this **pointer-variable** points to variable containing **char**.
   Ex: pointer **ptrchr0** point to **varchar0**(address 0x0001), and its type is **char**.
d. To make it clearer here is what happpened in the memory:

Address	**Value**	**Explanation**
0x0001	a	varchar0
0x0002	b	varchar1
0x0003	c	varchar2
0x0004	d	varchar3
0x0005	0x0001	ptrchr0
0x0006	0x0002	ptrchr1
0x0007	0x0003	ptrchr2
0x0008	0x0004	ptrchr3

e. "**&varchar0**" means, we extract the address from variable "**varchar0**"

19. Next we will test these variable by printing it. So add main method below the declare of 4 char-variable and 4 pointer-variable

```
char varchar0='a';
char varchar1='b';
char varchar2='c';
char varchar3='d';
char* ptrchr0=&varchar0;
char* ptrchr1=&varchar1;
char* ptrchr2=&varchar2;
char* ptrchr3=&varchar3;

void main(void){

}
```

20. In main method, write this :

```
printf("varchar0: val=%c, adr=%p\n",varchar0, &varchar0);
```

Do yourself : print variable **varchar1** through **varchar3** using above pattern

21. Compile and Run the program
    a. save this file
    b. exit from your beloved text editor, then enter the code below to your command line

```
$ gcc 00-pointer-basic.c -o 00-step-1
$ ./00-step-1
```

22. Return to your beloved text editor and open "**00-pointer-basic.c**"

23. Add this code below the code in step 20

```
printf("ptrchr0: pts=%c, val=%p, adr=%p\n", *ptrchr0, ptrchr0, &ptrchr0);
```

Explanation :

There is new pattern here, ***ptrchr0**. This statement will extract the value of variable it points (in this case **varchar0**). Because **ptchr0** points to **varchar0**, so ***ptrchr0** will return **varchar0** value and that is 'a'

Do yourself : print variable **ptrchr1** through **ptrchr3** using above pattern

24. Run it.
    a. save this file
    b. exit from your beloved text editor, then enter the code below to your command line

```
$ gcc 00-pointer-basic.c -o 00-step-2
$ ./00-step-2
```

25. Return to your beloved text editor and open "**00-pointer-basic.c**"

26. Now we arrive at the tricky part. We will implement **nested pointer-variable**. Nested pointer-variable will point to another **pointer-variable**. Now write this code **in main method**

```
char** ptrptr0=&ptrchr0;
```

Explanation:

    a. Two stars means this **nested pointer-variable** points to a pointer-variable and that **pointer-variable** point directly to the real value.
    b. n stars means that it need n pointing to reach the real value

Do yourself : declare **ptrptr1** through **ptrptr3** (**2-stars** nested pointer-variable) using above pattern.

27. Print those nested variables

```
printf("ptrptr0: ppt=%c, pts=%p, val=%p, adr=%p\n", **ptrptr0, *ptrptr0, ptrptr0, &ptrptr0);
```

Explanation:

To make it clearer here is what happpened in the memory:

Addres	**Value**	**Explanation**
0x0001	a	varchar0 == *ptrchr0 == **ptrptr0
0x0002	0x0001	ptrchr0 == *ptrptr0
0x0003	0x0002	ptrptr0

Do yourself : print also variable **ptrptr1** through **ptrptr3** using above pattern.

28. Run it.
    a. save this file
    b. exit from your beloved text editor, then enter the code below to your command line

```
$ gcc 00-pointer-basic.c -o 00-step-3
$ ./00-step-3
```

29. Return to your beloved text editor and open "**00-pointer-basic.c**"

30. Now do the following task by yourself
    a. declare a **local 3-star** variable-pointer named "**ultimate_pointer**", assign it using **available variable**.
    b. print its value, address, *ultimate_pointer, **ultimate_pointer and ***ultimate_pointer (print in above order)

31. Run it.
    a. save this file
    b. exit from your beloved text editor, then enter the code below to your command line

```
$ gcc 00-pointer-basic.c -o 00-step-4
$ ./00-step-4
```

I/O Process

1. Take a look some of system calls in linux:

```
$ man -s 2 open
$ man -s 2 close
$ man -s 2 read
$ man -s 2 write
```

2. Compile and run the **open-close** program

```
$ gcc open-close.c -o open-close
$ ./open-close
```

3. Check the file on your directory

```
$ ls -al
```

4. Take a look on the new file that are the result from the program, and check the permission of the file. Takes time to read the code and learn it.

5. Then you must answer some of question below and write it on **lab04.txt** point number 2 :
    a. Explain the parameter of **flags** in open function that used in the code !
    b. Explain the parameter of **modes** in open function that used in the code !
    c. Write your knowledge about open and close I/O process (don't forget to add the sources if you use reference)
    (*warning : write the answer by your own, plagiarism rule applies)
6. Compile and run the write program

```
$ gcc write.c -o write
$ ./write
```

7. Take a look on the output of the program and takes time to learn the code of the program.

8. Write your knowledge about the write program on lab04.txt point number 3
(*warning : write the answer by your own, plagiarism rule applies)

Privacy Matters, Encryption and Digital Signature using GnuPG

1. Hash and sign your works so the other know it truly your works

```
$ sha1sum * > SHA1SUM
$ sha1sum -c SHA1SUM
$ gpg --sign --armor --detach SHA1SUM
```

2. verify the works

```
$ gpg --verify SHA1SUM.asc
```

3. create a tar ball. Tar is a way to create an archive file. You can ask uncle G for more information

```
$ cd ..
$ tar cvfj work04.tbj work04/
```

4. encrypt your files (work04.tbj)

```
$ gpg --output work04.tbj.gpg --encrypt --recipient OSTEAM --recipient
your@email.com work04.tbj
```

*Use the same email as your Email input on GnuPG key generator.

5. copy the file to your github account, under the file week04/

```
$ cp work04.tbj.gpg ~/os172/week04/work04.tbj.gpg
```

6. change your directory to **os172/week04/**
7. remove file named ”**dummy**”
8. check whether there is a file named ”**work04.tbj.gpg**” if you dont find it, do the copy once more.
9. push the change to GitHub server
10. done!

Review Your Work

Dont forget to check your files/folders. After this lab, your current os172 folder should looks like:

os172

- key
  - mypublickey1.txt
- log
  - log01.txt
  - log02.txt

prepared by **HZ** and **RF**

- log03.txt
- log04.txt

SandBox

- <some_random_name>

week00

- report.txt

week01

- lab01.txt
- report.txt
- whyStudyOS.txt
- what-time-script.sh

week02

- work02.tbj.gpg
  - *work02
    - *00-toc.txt
    - *01-public-osteam.txt
    - *02-ls-al.txt
    - *03-list-keys1.txt
    - *04-list-keys2.txt
    - *hello.c
    - *hello
    - *status.c
    - *status
    - *loop.c
    - *loop
    - *exercise.c
    - *exercise
    - *SHA1SUM
    - *SHA1SUM.asc

week03

- work03.tbj.gpg
  - *work03
    - *.profile
    - *sudo-explanation.txt
    - *what-is-boot.txt

*SHA1SUM
*SHA1SUM.asc

week04

work04.tbj.gpg

*work04
*01-public-osteam.txt
*lab04.txt
*global-char.c
*global-char
*local-char.c
*local-char
*open-close.c
*open-close
*write.c
*write
*result1.txt
*result2.txt
*demo-file1.txt
*demo-file2.txt
*demo-file3.txt
*demo-file5.txt
*00-pointer-basic.c
*00-step-1
*00-step-2
*00-step-3
*00-step-4
*SHA1SUM
*SHA1SUM.asc

week05

dummy

week06

dummy

week07

dummy

week08

dummy

week09

dummy

week10

dummy

xtra

dummy

keep in mind for every files/folders with **wrong name**, you will get penalty point.

*means file that should be inside the archived file.