Fall 2016 CST8234 – C Programming
Lab 04: Pointers & Addresses

Create a directory `Lastname_04` you are going to develop your lab here

Program #1:

You are to write a small program **`04_Address_A.c`** to print information about variable addresses and its content.

0. Define **two *local int variables*** *to* `main( )` and ***one pointer to an integer***. Do not initialize your variables.

   For example:

```
    int x, y;
    int * p;
```

1. Print the size of an integer and the size of the memory address.

```
------------------------------------------------------------
STEP 1:  Printing Sizes
Sizeof( int ) = 4
Sizeof( memory address ) = 8
------------------------------------------------------------
```

   `sizeof` return the amount of space a designated data type would occupy in memory in bytes. The return data type is a `size_t`, define as an unsigned long. Use `%ul` when you print a datatype `size_t`.

2. Print all the variables information:

   (A) For the integers, print their memory location (address) and content

   (B) For the pointer, print its memory location, content and what it points to.

```
------------------------------------------------------------
STEP 2:  Variables -- Before initialization
x:    &x = 0x7fff1ddb45e8     x = 0
y:    &y = 0x7fff1ddb45ec     y = 0
p:    &p = 0x7fff1ddb45e0     p = 0x7fff1ddb46d0      *p = 1
------------------------------------------------------------
```

   Notice my variables `x, y,` and `p`. While it looks like I have initialized the variables, I haven't. As a coincidence, the “garbage” values here are 0, and 1.

3. Initialize the variables. Give any values to the integers, and assign one of the integer's memory to the pointer. Print all the variables information – as in (2)

```
------------------------------------------------------------
STEP 3:  Variables -- After initialization
x:    &x = 0x7fffb15141f8     x = 25
y:    &y = 0x7fffb15141fc     y = 1986
p:    &p = 0x7fffb15141f0     p = 0x7fffb15141f8      *p = 25
------------------------------------------------------------
```

4. Change the content of what the pointer points to by assigning a value to it. Print all the variables information, as in (2)

```
------------------------------------------------------------
STEP 4:  Pointer content with new value
x:    &x = 0x7fffa32c1938     x = 55
y:    &y = 0x7fffa32c193c     y = 1986
p:    &p = 0x7fffa32c1930     p = 0x7fffa32c1938      *p = 55
------------------------------------------------------------
```

5. Assign to one of the variables, the pointer. This will create a warning at compilation time. Print all the variables information, as in (2)

```
root@luna:~15F_CST8234/04_Sol# gcc -o 04_Addresses_A 04_Addresses_A.c
-ansi -Wall -pedantic
04_Addresses_A.c: In function 'main':
04_Addresses_A.c:89:4: warning: assignment makes integer from pointer
without a cast [enabled by default]
```

This is an exercise, so the warning is what you must expect. Normally, if you get a warning like this one, you must "fix" it before moving forward.

```
------------------------------------------------------------
STEP 5:   Variable assignation to pointer
x:    &x = 0x7fff560af4b8     x = 1443558584
y:    &y = 0x7fff560af4bc     y = 1986
p:    &p = 0x7fff560af4b0     p = 0x7fff560af4b8      *p = 1443558584
------------------------------------------------------------
```

Notice the conversion from hex to decimal and the truncation. Your memory location can hold a bigger number that your integer.

To print a memory location, use `%p` in `printf()` you will additionally need to cast the value to a void *.

```
int *p;
printf("p:  &p = %p\tp = %p\n", (void *)&p, (void *)p );
```

Program #2:

Copy your `04_Address_A.c` to a new program `04_Address_B.c`.

1. Write a function `modify_var( )`, that requires as arguments an integer and a pointer to an integer. Your function should:
   (A) Print the variable information as in the last program
   (B) Modify both variables, the integer and the pointer to the integer
   (C) Print the variable information.
2. The `main( )` function should:
   (A) Declare two integer variables and a pointer to an integer. Initialize all the variables. Set the pointer to one of the variables
   (B) Print the variables information
   (C) Call `modify_var( )`. Pass the pointer and the variable you did not use to set the pointer
   (D) Print the variables information

```
root@luna:~15F_CST8234/04_Sol# ./04_Addresses_B
-------------------------------------------------------------
FUNCTION main( ):
x:    &x = 0x7fffa0412248     x = 5
y:    &y = 0x7fffa041224c     y = 10
p:    &p = 0x7fffa0412240     p = 0x7fffa0412248      *p = 5

-------------------------------------------------------------

Calling mod_var( y, p ):
FUNCTION modify_var( ) -- before modifications
a:    &a = 0x7fffa041222c     a = 10
q:    &q = 0x7fffa0412220     q = 0x7fffa0412248      *q = 5

FUNCTION modify_var( ) -- after modifications
a:    &a = 0x7fffa041222c     a = 100
q:    &q = 0x7fffa0412220     q = 0x7fffa0412248      *q = 45

-------------------------------------------------------------
FUNCTION main( ):
x:    &x = 0x7fffa0412248     x = 45
y:    &y = 0x7fffa041224c     y = 10
p:    &p = 0x7fffa0412240     p = 0x7fffa0412248      *p = 45

-------------------------------------------------------------
```

Program #3:

Copy your `04_Address_A.c` to a new program `04_Address_C.c`.

1. The `main( )` function should:
   (A) Declare one integer variable and two pointers to an integer.
   (B) Initialize the integer variable, and set one of the pointers to point to it.
   (C) Allocate memory in the `heap` for an integer and set the second pointer to point to it
   (D) Print the variables information

```
root@luna:~15F_CST8234/04_Sol# ./04_Addresses_C
-------------------------------------------------------------
In main( ) after memory allocation
x:    &x = 0x7ffffa39e17c     x = 5
q:    &q = 0x7ffffa39e170     q = 0x7ffffa39e17c      *q = 5
p:    &p = 0x7ffffa39e168     p = 0x1fa4010           *p = 50

-------------------------------------------------------------
```

I have allocated memory in the `heap` for `p`. Notice the starting address in this case. Stack and heap are totally different memory spaces.

For each program, create a memory map of what it is happening at each point of execution. Be able to recognize when the memory is being allocated in the stack or in the heap. You can create the memory map by hand.

For example, for 04_Address_A – Step 3:

```
int x;
int *p;

x = 5;
p = &x;
```