To:
Tenouk

C LAB WORKSHEET 11a_1
C & C++ Pointers Part 3: Pointers, Array and Functions

1. More on C/C++ pointers practice, questions and answers.
2. Pointers and memory addresses manipulations.
3. Tutorial references that should be used together with this worksheet are C & C++ pointers part 1 and C & C++ pointers part 2.

3. Using the variables in question #1, show the printouts for each of the following.

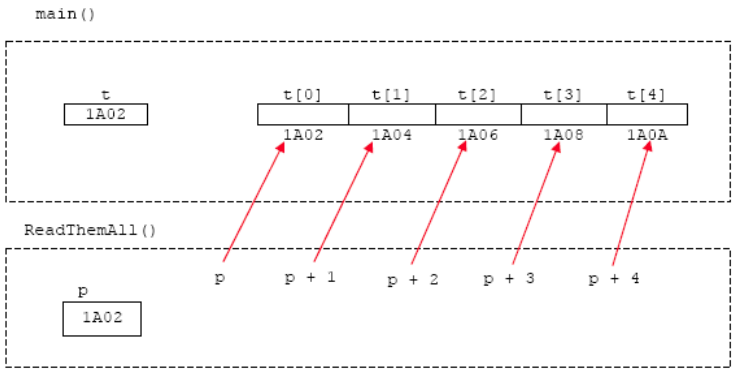
```
for(i = 0; i <= 4; i++)
  printf("j + i = %p, j[%d] = %d\n", j + i, i, j[i]);

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

for(i = 0; i <= 4; ++i)
  printf("x + i = %p, &x[%d] = %p, x[%d] = %.2f\n", x + i, i, &x[i], i, x[i]);

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

for(ptr2 = x + 4; ptr2 >= x; --ptr2)
  printf("ptr2 = %p, ptr2 - 1 = %p\n", ptr2, ptr2 - 1);
```

```
j + i = 0013FF44, j[0] = 4
j + i = 0013FF48, j[1] = 5
j + i = 0013FF4C, j[2] = 6
j + i = 0013FF50, j[3] = 7
j + i = 0013FF54, j[4] = 8
Press any key to continue . . .
```

```
x + i = 0013FF10, &x[0] = 0013FF10, x[0] = 4.00
x + i = 0013FF14, &x[1] = 0013FF14, x[1] = 5.00
x + i = 0013FF18, &x[2] = 0013FF18, x[2] = 6.00
x + i = 0013FF1C, &x[3] = 0013FF1C, x[3] = 7.00
x + i = 0013FF20, &x[4] = 0013FF20, x[4] = 8.00
Press any key to continue . . .
```

```
ptr2 = 0013FF20, ptr2 - 1 = 0013FF1C
ptr2 = 0013FF1C, ptr2 - 1 = 0013FF18
ptr2 = 0013FF18, ptr2 - 1 = 0013FF14
ptr2 = 0013FF14, ptr2 - 1 = 0013FF10
ptr2 = 0013FF10, ptr2 - 1 = 0013FF0C
Press any key to continue . . .
```

4. Define a float array called t[ ] with 10 slots. Send the address of the first element of the array to a function called ReadThemAll(). Also send 2 and 6 as integers. The function will assign the array address to a pointer called p and read floats in all the slots between these two numbers, inclusive. Although the ReadThemAll() function will read in the t[ ] array of main(), the ReadThemAll() function should not use brackets, [ ] but only the asterisk operator, *. When the function is done, have main() print out the entire array. See the following Figure for the detail. Write a complete program.




```
#include <stdio.h>

// a prototype
void ReadThemAll(int *x, int y, int z);

void main(void)
{
  int t[10];
  int j, p = 2, q = 6;
  // array's name without a bracket is a pointer
  // to the first element of the array
  ReadThemAll(t, p, q);
  for(j=p;j<=q;j++)
    printf("t[%d] = %d\n", j, t[j]);
}

void ReadThemAll(int *x, int y, int z)
{
  int i;
  for(i=y; i<=z;i++)
    // assign 7 to all the element
    x[i] = 7;
}
```

```
t[2] = 7
t[3] = 7
t[4] = 7
t[5] = 7
t[6] = 7
Press any key to continue . . .
```

5. Write a function called FindIt(). This function will receive z[ ], a float array; size, an integer specifying the number of slots in the array and look, a float. The function will look for look in the array and return the address where that number was found in the array. Assume that the number will be found. Write main() to go with the function that will pass the appropriate variables and receive the address where the number was found into a pointer. Then print out that address.

```
#include <stdio.h>

// a prototype, the return value is an address
// so we use a pointer to a float value
float * FindIt(float *z, int size, float look);

void main(void)
{
  // f or F just a modifier to force the type/value to float,
  // because double will be assumed by default
  // it looks similar in Java :-)
  float num[5] = {3.5f, 5.7f, 2.3f, 1.7f, 7.8f};
  float find = 2.3f;
  int slot = 5;
  // this will store the return 'address'
  float *addr;

  // bring along all necessary arguments...
  // an array, array's size and a float to be searched
  addr = FindIt(num, slot, find);
  printf("The address of value %.2f is %p\n", find, addr);
}

float * FindIt(float *z, int size, float look)
{
  int i;
  float *loc;
  for(i=0; i < size; i++)
  {
    printf("z[%d] = %.2f at %p\n", i, z[i], &z[i]);
    // take note here. We are comparing an array value
    // to a fix value
    if(z[i] == look)
    loc = &z[i];
  }
```

```
    // return the address through a pointer
    return loc;
}
```

```
z[0] = 3.50 at 0013FF50
z[1] = 5.70 at 0013FF54
z[2] = 2.30 at 0013FF58
z[3] = 1.70 at 0013FF5C
z[4] = 7.80 at 0013FF60
The address of value 2.30 is 0013FF58
Press any key to continue . . .
```

More Pointers Questions And Practice

1. Let us first review how computers count:

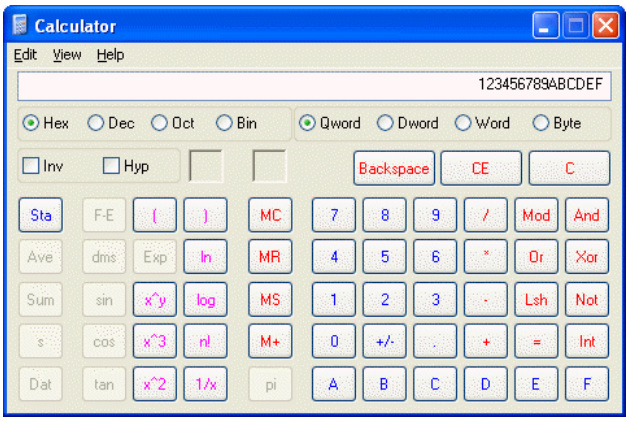
   1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21 and so on.

   This method of counting numbers is called the **hexadecimal numbering system** or **hex** for short. Keep in mind that computers process data based on **binary numbering system**, 0 and 1. You can use scientific calculator to convert to other numbering system for example, binary to hex or to **decimal numbering system** that we are familiar to.

   a. In hex, what number comes after 9, 19, F and 1D?
   b. In hex, what number comes before 10, 20, F and 2E?
   c. In hex, what number comes after 99, 299, 29F, FFEF and 9FF?

   a. A, 1A, 10 and 1E respectively.
   b. F, 1F, E and 2D respectively.
   c. 9A, 29A, 2A0, FFF0 and A00 respectively.

Using the Hex mode from Windows scientific calculator.



2. Each memory location (also called Random Access Memory - RAM) in a computer has an address so that the system knows which location to access. Each location typically stores one byte of data. The amount of data RAM can store is measured in bytes. In the following Figure, each memory location's address is shown. What is the memory address of the location that is 5 addresses away from FFAE?

FFAE	FFAF	FFB0	FFB1	FFB2	FFB3	FFB4	FFB5	FFB6	FFB7	FFB8

FFB3 (FFAE + 5)

3. More memory locations are shown here after location FFB8. For those locations, their contents are also shown. In which location is 23 stored? What is the address of the memory location that is 10 addresses before the location where 112 is stored?

FFAE	FFAF	FFB0	FFB1	FFB2	FFB3	FFB4	FFB5	FFB6	FFB7	FFB8

89	-3	45	66	23	48	1	112	456	-20	56
FFB9	FFBA	FFBB	FFBC	FFBD	FFBE	FFBF	FFC0	FFC1	FFC2	FFC3

23 is stored at FFBD. The address is FFB6 (FFC0 - A where 10 is A in hex)

4. In the following array declaration, array a[ ] is defined and initialized. Some computers need two bytes of memory to store integers as shown in the Figure. What is the starting address of the location where 28 is stored? How many bytes are needed to store a 4-element integer array shown below?

   ```
   int a[4] = {4, 77, 28, 12};
   ```

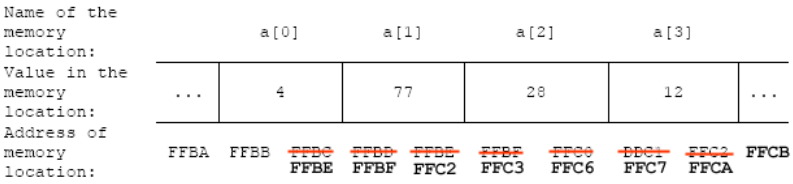
Name of the memory location:		a[0]	a[1]	a[2]	a[3]	
Value in the memory location:	...	4	77	28	12	...
Address of memory location:	FFBA	FFBB FFBC	FFBD FFBE	FFBF FFC0	DDC1	FFC2

The starting address for the 28 is FFC3. It takes 16 bytes to store 4-element integer array. Every integer will take 4 bytes each, then 4 x 4 = 16 bytes. 16 is equal t o10 in hex. Hence, FFBB + 9 = FFCA (9 is used because the first address index is 0) or the first address that is FFBB already considered as 1 location.

The following code sample can be used to determine the array size.

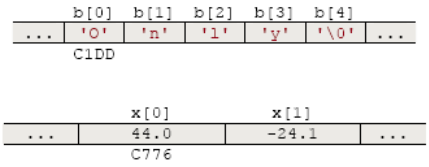
```
#include <stdio.h>

void main(void)
{
  int a[4] = {4, 77, 28, 12};
  printf("The size of a is %d\n", sizeof(a));
}
```

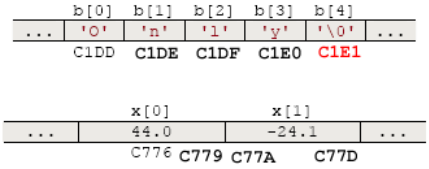
```
The size of a is 16
Press any key to continue . . .
```



5. If 1 byte is needed to store characters on a certain computer and 4 bytes are needed to store floats, show the starting addresses of each array slot in the diagram for these two arrays. The starting addresses of b[ ] and x[ ] are given. Also, where does x[ ] end?

```
char b[5] = "Only";
float x[2] = {44.0, -24.1};
```





The x[ ] ends at C77D.

6. Answer the following questions based on the structure data type.

```
struct money
{
    char Country[10];
    int Denomination;
    float ConversionFactor;
};

struct money rupees[6];
```

a. How many bytes does each slot of rupees[ ] occupy?
b. What is the (starting) address of each slot of rupees[ ] if the first one starts at FF10?

a. Theoretically every slot of rupees[ ] will occupy 10 + 4 +4 = 18 bytes.
b. Theoretically they are FF10, FF28, FF40, FF58, FF70, FF88, ..., ...

In the real situation memory allocation should be slightly different (normally allocated more than theoretically they should be) because of the memory alignment. The 32 bits will consist of 4 bytes. Each bytes is 8 bits in size. A single byte will consist of 2 nibbles that is 4 bits each and nibble may be the smallest alignment. In this case the allocated memory will be in multiple of 8 bits (1 byte) though the variable just only need 5 bits more, 8 bits (1 byte) in size will be allocated. For example, in 32 bit system, this struct will have 20 bytes in size. In this case using 4 bytes for every 32 bits in size, we need 4 bytes x 5 memory slots alignment = 20 bytes. If the system allocate 4 memory slots alignment, it is not enough because 4 bytes x 4 memory slots = 16 bytes while the struct need 18 bytes. Hence the nearest alignment is 4 bytes x 5 memory alignment slots = 20 bytes. However this also depend on the processor and/or the computer architecture itself such as PowerPC, Power and Intel based system.

```
#include <stdio.h>

struct money
{
  char Country[10];
  int Denomination;
  float ConversionFactor;
};

void main(void)
{
  struct money rupees[6];
  printf("The size of char is %d\n", sizeof(char));
  printf("The size of int is %d\n", sizeof(int));
  printf("The size of float is %d\n", sizeof(float));
  printf("The size of struct money is %d\n", sizeof(struct money));
  printf("The size of rupees is %d\n", sizeof(rupees));
}
```

```
The size of char is 1
The size of int is 4
The size of float is 4
The size of struct money is 20
The size of rupees is 120
Press any key to continue . . .
```

Try change the size of the Country's array to different sizes, rebuild and re-run this program and notice the new struct size. Different array sizes may have same size of struct because of the memory alignment.

7. Pointers are special kinds of variables used to store memory addresses. You use an asterisk in front of the variables when defining a pointer.

```
int i, *p1;
float x, *p2;
```

a. Which of the four variables above are pointers: i, x, p1 and/or p2?
b. Which of these four items can store addresses?

a. *p1 and *p2 are pointer variables.
b. *p1 and *p2 can store addresses.

8. Not only is p1 a special kind of variable (a pointer variable) it is also a special kind of a pointer. p1 is a pointer to integers only. Therefore, if 1 is added to p1, then actually 4 is added to it because we are assuming that integers take 4 bytes to store on our computer.

```
int i, *p1;
float x, *p2;
```

   a. If we say, p1 = p1 + 3; and the initial value of p1 is FF18, what will be the final value of p1?
   b. If the initial value of p1 was CD1E and we add 5 to it, what will p1 become?
   c. If floats need 4 bytes of RAM to be stored and initially p2 is 7D50, what will be the value of p2 after 3 is added to it?
   d. If p2 is initially CABB and 5 is added to it, what will p2 become?

   a. Final value of p1 is FF24. Here 3 x 4 bytes each = 12 bytes far away. 12 in hex is C. Then FF18 + C = FF24.
   b. Here 5 x 4 bytes each = 20 bytes. 20 in hex is 14. Then CD1E + 14 = CD32.
   c. Here 3 x 4 bytes each = 12 bytes in total. 12 is C in hex. Then 7D50 + C = 7D5C.
   d. Similar to (b), 5 x 4 bytes each = 20 bytes. Then CABB + 14 = CACF.

9. As before, p1 is a pointer to integers and p2 is a pointer to floats. Before each part below, assume p1's initial value is F100 and p2's initial value is F2CC. Give the values of p1 and p2 after each of these statements:

   a. p1 = p1 + 2;
   b. p1 = p1 – 2;
   c. p2 = p2 + 2;
   d. p2 = p2 – 2;

   a. F100 + 8 = F108.
   b. F100 - 8 = F0F8.
   c. F2CC + 8 = F2D4.
   d. F2CC - 8 = F2C4.

10. Assume that p1 has the address of i[0] and p2 has the address of x[0].

```
int i[5], *p1;
float x[5], *p2;
```

   a. After the statement p1 = p1 + 3; p1 will contain the address of which slot in the array i[ ]?
   b. After this statement p2 = p2 + 3; p2 will contain the address of which slot in the array x[ ]?
   c. In total how many address locations were added in a? In b?
   d. In total how many indexes were added in a? In b?

   a. Slot number 3 (index number 2).
   b. Slot number 3 (index number 2).
   c. A total of 3 x 4 = 12 address locations both in a. and b. (inclusive the first slot else just 11 address locations).
   d. 3 indexes were added for both a. and b..

11. i[0] is stored at F400 and x[0] is stored at BBB0. Assume that initially, p1 has the address of i[0] and p2 has the address of x[0] answer the following questions.

```
int i[15], *p1;
float x[15], *p2;
```

   a. If p1 contains F410, it has the address of which slot of the array i[ ]?
   b. If p2 contains BBC0, it has the address of which slot of the array x[ ]?

   a. Slot number 4. (F410 - F400) = 10. 10 is 16 in decimal. So 16 / 4 bytes per integer = 4.
   b. Slot number 4. (BBC0 - BBB0) = 10. 10 is 16 in decimal. So 16/4 bytes per float = 4.

12. Which of the following statements are valid and give reason?

   a. p1 = &i[0];
   b. p1 = &x[0];
   c. p1 = i[0];

   a. This is valid because the address of the first array element is assign to the pointer *p1 and both have a similar type, int.
   b. Not valid because address of the first float type array is assigned to an integer type pointer.
   c. Not valid because we cannot assign a value (i[0] - array's element) to a pointer that hold an address.

13. The name of the array (without the bracket) is equal to the address of the first slot/index in that array. If x[5] is stored starting at AA00, then what is the value of x? What is the value of &x[0]?

The value of x (a pointer to the first address of the array element) and &x[0] (address of the first array's element) is AA00.

14. If x[15] is stored starting at AA00, then what is the address of each?

   a. x + 2
   b. &x[2]

   a. AA00 + 2 = AA02.
   b. AA00 + the address of the third array's element depending on the type of the array.

15. Arrays are similar to pointers. One difference is that we can't change the value of the array address, but we can change the value of a pointer variable. Using the declaration given below, which of the following statements are valid and give reason?

```
int i[15], *p1;
float x[15], *p2;
```

   a. x = x + 1;
   b. x = p2 + 1;
   c. p2 = x + 1;
   d. p2 = p2 + 1;
   e. x[2] = x[2] + 1;
   f. &x[2] = p2;

   a. Valid. Add 1 to the first array's address, x and assign the new address to the first array's address, x overwriting the previous address.
   b. Valid. Add 1 to the pointer p2, and assign the new address value to the first array element's address, x.
   c. Valid. Add 1 to the first array element's address, x and assign this new address to the pointer *p2.
   d. Valid. Add 1 to pointer p2 and assign this new value to pointer p2 (previous value will be overwritten).
   e. Valid. Add 1 to the third array's element, x[2] and assign this new value to the third array's element, x[2], overwriting the previous value.
   f. Valid. Assign pointer p2 to the third array element address, &x[2].

16. An array can be considered as a **pointer constant** and its address can't be changed. A pointer variable such as p2 can have the address stored in it changed. Which of the following two initializations are valid?

```
int *ptr1;
int i[10] = &ptr1;
```

```
int i[10];
int *ptr1 = &i[0];
```

The second one is valid. In the first one we try to change the address of the 11th array element by assigning pointer ptr1.

17. When we pass an array to a function, we actually pass the address of that array. Hence, a function that receives an array as an argument can receive it as a pointer. Write a function header for a function called MyFunction(), where an array is received into float pointer called p. A float pointer is returned by the function.

```
float * MyFunction(float *p);
```

18. Write the main() code that will pass this array to MyFunction(). Also write the MyFunction() that will print the addresses of the first five elements of the array and return the address of the fifth element. main() will receive that address into a pointer called q and it will print it.

```
#include <stdio.h>

// a function prototype
float * MyFunction(float *p);

void main(void)
{
  float test[10] = {1.55f,2.4f,3.5f,4.7f,5.2f,6.1f,7.28f,8.8f,9.32f,11.77f};
  float *q;

  // call MyFunction bringing along the array through a pointer
  q = MyFunction(test);
  printf("The address of the fifth element is %p\n", q);
}

float * MyFunction(float *p)
{
  int i;
  // a good practice to assign uninitialized
  // pointer to NULL. NULL point to the address 000000 of memory.
  float *fifth = NULL;

  // print the addresses of the first five elements
  for(i=0;i<5;i++)
    printf("p[%d] = %.2f at %p\n", i, p[i], &p[i]);
  // assign and return the fifth element's address
  fifth = &p[4];
  return fifth;
}
```

```
p[0] = 1.55 at 0013FF3C
p[1] = 2.40 at 0013FF40
p[2] = 3.50 at 0013FF44
p[3] = 4.70 at 0013FF48
p[4] = 5.20 at 0013FF4C
The address of the fifth element is 0013FF4C
Press any key to continue . . .
```

19. Answer the following questions.

    a. Is an array a **pointer constant** or a **pointer variable**?
    b. If p is a pointer and x[ ] is an array, can we print p[2]?
    c. If p is a pointer and x[ ] is an array, can we print x + 2?
    d. What did we assume the storage units (number of bytes) for characters, integers and floats to be?

    a. An array is a pointer constant.
    b. No, we can't.
    c. Yes, we can.
    d. Depending on the platform or machine, character is 1 byte, integer and float are 4 bytes each.