[*Pointers*]

One of the first things you learned in C++ was how to create variables and store information in them, for example:

```
int x;
```

when you write this command, you are asking the computer to give you enough memory to hold one integer and this integer is called **x**. The following command:

```
x = 5;
```

tells the computer to place the integer 5 into the memory location that we named **x**. This also means that in the future if we want to read this integer or store a different integer in this memory location we can refer to it as **x**.

```
std::cout<<x;
```

this command will print the integer 5 to the screen.

```
x = 7;
```

this command will change the integer stored in **x** to 7. However this will only update the variable in the scope where the variable **x** was declared. If you create **x** in main and then pass **x** to a function, anything you do to **x** in the function will not update the variable **x** in the main function, example:

```
void doSomething(int x){
    x = 10;
}
int main( ){
    int x = 5;
    doSomething(x);
```

```
    std::cout<<x;
    return 0;
}
```

this program first creates a variable by asking the computer for enough memory to store one integer, we call this integer **x** and store the integer 5 there. Then we pass the variable **x** into the function called **doSomething(int x)**. Inside this function we change **x** to the integer 10. Then we exit the function and print the integer stored in **x**, however this will print 5 not 10. The reason is because the variable **x** in the function **doSomething** is not the same as the variable in main. When we pass a variable to a function, C++ creates a copy of that variable in a different memory location with whatever name you called it in the function.

If we want to be able to update the variable **x** in the main function whenever we change it in another function we can pass by reference or pass by pointers. Before we learn how to do that, first let's learn what pointers are and how to use them.

A pointer is a variable that stores the address of another variable. We declare pointers by putting an asterisk (*) between the data type and name of the variable:

```
int* pX;
```

this can also be written like the following two (these all mean the same thing):

```
int * pX;
int *pX;
```

this tells the computer to give us enough memory to hold an address for an integer. Now we ask the computer to give us enough memory to hold an integer.

```
pX = new int;
```

this tells the computer to give us enough memory to hold one integer, the address of this integer is then stored in pX.

```
*pX = 3;
```

this stores the integer 3 into the memory location that we requested in the earlier command. We can think of this in the following way, the memory location that is storing an address is called pX, hence, *pX is a pointer pointing to the address that it has stored and this address is holding an integer, in this example the integer that it is holding is 3.

You can declare a pointer and create a new int all in one step with the following code:

```
int* pX = new int;
```

In C++ when putting the ampersand sign (&) in front of a data type, this returns the address of where the data type is stored in the memory instead of what is stored in the data type.

The following line of code will print the memory location of the pointer pX:

```
std::cout<<&pX;
```

The following line of code will print the memory location that is stored in pX, this is the integer that pX is pointing to when we created it in the previous like using "**new int**":

```
std::cout<<pX;
```

both of these lines of code will print the address in hexadecimal form because that is how C++ stores memory locations. If you want to convert these numbers to integers we can do this by adding the following to the beginning of pX:

```
std::cout<<(long int)(&pX);
std::cout<<(long int)(pX);
```

Whenever we use pointers and ask the computer for memory using the **new int** command, we need to remember to delete the pointer and return the memory back to the computer. We can delete a pointer with the following code:

```
delete pX;
```

We normally delete pointers at the very end of the program or once we know we will not need them any longer in the program. Once we delete the pointer, the memory is returned to the computer and the computer can then assign that memory to a different program on the computer. If you forget to delete your pointers after you close the program, then you have lost access to that part of your computer's memory and this will cause a memory leak. You will gain access to that memory once you restart your computer.

Practice:
Explain what each line of code is doing in the following example.

```
#include<iostream>

int main( ){
    int* a;
    a = new int;
    *a = 15;
    std::cout<<*a <<std::endl;
    std::cout<<&a <<std::endl;
    *a += 3;
    std::cout<<*a <<std::endl;

    delete a;
    return 0;
}
```

Explanation:

1. **int* a;** asks the computer for enough memory to hold a pointer that we will call **a;**

2. **a = new int;** asks the computer for enough memory to hold an integer and store the address of this integer in our pointer that we called **a;**
3. ***a = 15;** now we store the value 15 in the variable that **a** is pointing to.
4. **std::cout<<*a <<std::endl;** will print 15, the integer **a** is pointing to.
5. **std::cout<<&a <<std::endl;** will print the address of the pointer **a.**
6. ***a += 3;** will add 3 to the integer that **a** is pointing to, increasing it to 18;
7. **std::cout<<*a <<std::endl;** will print 18;
8. **delete a;** deletes the pointer we called **a** and returns the memory back to the computer.