Arrays, Pointers and Pointer Arithmetic

There is a very close relationship between arrays and pointers. The name of an array is a pointer **constant** to the first element. Because the array's name is a pointer constant, it cannot be changed. Since the array name is a pointer constant to the first element, the address of the first element and the name of the array both represent the same location.

We can use the array name anywhere we can use a pointer.
i.e. if an array of ints is decalared as,

```
int a[10];
```

a *is equivalent to* **&a[0]**

If the above array was stored at location 1123F7h then,

```
cout << &a[0] << " " << a << endl;
```

will print the same thing – the address of the first element in the array:

```
oX1123F7 oX1123F7
```

Since the name of an array is really a pointer, the following expressions are exactly the same when ***a*** is the name of an array and ***n*** is an integer.

```
a[n] is equivalent to *(a + n)
```

Given:

```
int a[5] = { 3, 5, 7 11, 13};
int n;
int *ptr;
```

You can print the contents of the array a using pointer notation instead of subscript notation:

```
for(n = 0; n < 5; n++)
    cout << *(a + n) << endl;            // same as cout << a[n] << endl;
```

Another way:

```
ptr = a;   // store address of first element in ptr – remember a is a constant ptr
for(n = 0; n < 5; n++){
    cout << *ptr << endl;
    ptr++;   //point to next element – next address of type int
}
```

Yet another way:

```
ptr = a;
for(n = 0; n < 5; n++) // Can you think of another way to print that eliminates n?
    cout << *ptr++ << endl;
```

The star and increment operators associate right to left and have the same precedence, so the expression *ptr++ is equivalent to *(ptr++).

When the expression ptr++ is evaluated, ptr is incremented (so that ptr points to the next cell in the array), and because ++ occurs on the right side of ptr (post – increment), the *original* value of ptr becomes the value of the *expression* ptr++. The de-referencing operator * thus applies to the original value of ptr; that is, the contents originally pointed to by ptr is used for the cout, not the contents of the updated address. To summarize, the effect of evaluating the expression *ptr++ is to increment ptr and to reference the cell to which ptr originally pointed. Thus when the statement

cout << *ptr++ << endl;

is executed, the value of the current array cell is printed and ptr is incremented so that it points to the next cell in the array a.

Given:

```
ptr = a;          //places the address of a[0] in ptr
(*ptr)++;   /* Access the item in the cell to which ptr points (a[0]),
               and then increment that item. a[0] now contains 4.
               The address stored in ptr remains unchanged.        */
```

Given:

```
int a[3] = {11, 13, 17};
int *ptr = a;   // ptr contains address  of a[0] – same as writing int *ptr = &a[0]
int temp;

temp= a[0];             // temp contains 11
temp = *(a+2);          // temp contains 17
temp = *(ptr + 1);      // temp contains 13
temp = *ptr;            // temp contains 11

ptr = a + 1;            // ptr contains address of a[1]
temp = *ptr;            // temp contains 13
temp = *(ptr + 1)       // temp contains 17

ptr = a;                // ptr contains address  of a[0]
temp = *++ptr;          // ptr contains &a[1] - temp contains 13
temp = ++*ptr;          // temp contains 14 – a[1] contains 14
temp = *ptr++;          // temp contains 14 – ptr contains &a[2]
temp = *ptr;            // temp contains 17
```