Pointer Arithmetic

- C++ enables pointer arithmetic—a few arithmetic operations may be performed on pointers.
- Pointer arithmetic is appropriate only for pointers that point to *built-in array elements*.
- **A pointer may be incremented (++) or decremented (--)**

```
int array[] = {5, 10, 15, 20, 25, 30, 35};
int* ptr = array;

cout << "ptr at " << *ptr << endl;          // 5
++ptr;
cout << "++ Now ptr at " << *ptr << endl;   // 10
++ptr;
cout << "++ Now ptr at " << *ptr << endl;   // 15
++ptr;
cout << "++ Now ptr at " << *ptr << endl;   // 20
--ptr;
cout << "-- Now ptr at " << *ptr << endl;   // 15
```

- When incrementing or decrementing a pointer, it is incremented or decremented by *the size of the memory object to which the pointer refers*.

```
cout << "array element size:" << sizeof(array[0]) << endl;
```

- Most computers today have four-byte or eight-byte integers. Because the results of pointer arithmetic depend on the size of the memory objects a pointer points to, pointer arithmetic is machine dependent.
- **An integer may be added to (+ or +=) or subtracted from (- or -=) a pointer**
- When an integer is added to, or subtracted from, a pointer, the pointer is incremented or decremented by that integer times *the size of the memory object to which the pointer refers*.

```
ptr += 4;
cout << "+4 Now ptr at " << *ptr << endl;   // 35
```

- There's no bounds checking on pointer arithmetic. You must ensure that every pointer arithmetic operation that adds or subtracts from a pointer results in a pointer that references an element within the built in array's bounds.

One pointer may be subtracted from another pointer of the same type

- Pointer variables pointing to the same built-in array may be subtracted from one another, returning the *number of built-in array elements*.

```
long diff = ptr - array;
cout << "Index differenece " << diff << endl; // 6
```

Pointers v Built-In Arrays

- Pointers, such as int* `ptr` in the above, and built-in arrays, such as `int array[]` in the above, may be used *almost* interchangeably, except built in arrays may not have the value of the built in array's name *modified* with pointer arithmetic.

```
// ++array;    // Gives error, cannot modify built in value
```