OBJECTIVE-C - POINTER ARITHMETIC

As explained in main chapter, Objective-C pointer is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -

To understand pointer arithmetic, let us consider that **ptr** is an integer pointer, which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer:

```
ptr++
```

Now, after the above operation, the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer location, which is 4 bytes next to the current location. This operation will move the pointer to next memory location without impacting actual value at the memory location. If **ptr** points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

Incrementing a Pointer

We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer. The following program increments the variable pointer to access each succeeding element of the array:

```
#import <Foundation/Foundation.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = var;
   for ( i = 0; i < MAX; i++)
   {

      NSLog(@"Address of var[%d] = %x\n", i, ptr );
      NSLog(@"Value of var[%d] = %d\n", i, *ptr );

      /* move to the next location */
      ptr++;
   }
   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
2013-09-14 00:08:36.215 demo[32000] Address of var[0] = 7e6f2a70
2013-09-14 00:08:36.216 demo[32000] Value of var[0] = 10
2013-09-14 00:08:36.216 demo[32000] Address of var[1] = 7e6f2a74
2013-09-14 00:08:36.216 demo[32000] Value of var[1] = 100
2013-09-14 00:08:36.216 demo[32000] Address of var[2] = 7e6f2a78
2013-09-14 00:08:36.216 demo[32000] Value of var[2] = 200
```

Decrementing a Pointer

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

```
#import <Foundation/Foundation.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = &var[MAX-1];
   for ( i = MAX; i > 0; i--)
   {

      NSLog(@"Address of var[%d] = %x\n", i, ptr );
      NSLog(@"Value of var[%d] = %d\n", i, *ptr );

      /* move to the previous location */
      ptr--;
   }
   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
2013-09-14 00:12:22.783 demo[13055] Address of var[3] = ea4c618
2013-09-14 00:12:22.783 demo[13055] Value of var[3] = 200
2013-09-14 00:12:22.783 demo[13055] Address of var[2] = ea4c614
2013-09-14 00:12:22.783 demo[13055] Value of var[2] = 100
2013-09-14 00:12:22.783 demo[13055] Address of var[1] = ea4c610
2013-09-14 00:12:22.783 demo[13055] Value of var[1] = 10
```

Pointer Comparisons

Pointers may be compared by using relational operators, such as ==, <, and >. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

The following program modifies the previous example one by incrementing the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1]:

```
#import <Foundation/Foundation.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have address of the first element in pointer */
   ptr = var;
   i = 0;
   while ( ptr <= &var[MAX - 1] )
   {

      NSLog(@"Address of var[%d] = %x\n", i, ptr );
      NSLog(@"Value of var[%d] = %d\n", i, *ptr );

      /* point to the previous location */
      ptr++;
      i++;
   }
   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
2013-09-14 00:15:49.976 demo[24825] Address of var[0] = ae1235a0
2013-09-14 00:15:49.976 demo[24825] Value of var[0] = 10
2013-09-14 00:15:49.977 demo[24825] Address of var[1] = ae1235a4
2013-09-14 00:15:49.977 demo[24825] Value of var[1] = 100
2013-09-14 00:15:49.977 demo[24825] Address of var[2] = ae1235a8
2013-09-14 00:15:49.977 demo[24825] Value of var[2] = 200
```