Arrays and pointers

A dirty little secret revealed. When we have used arrays in the past, we have been using pointers all along. The array notation is simply another way of expressing pointers for things that are stored in contiguous sections of memory.

When we declare an array, like

```
int anArray[8];
```

the compiler takes this to mean a pointer, called `anArray`, pointing to a chunk of memory corresponding to an integer, along with 7 more contiguous integer-sized chunks of memory. Thus we can access the elements using pointers directly.

`*anArray` refers to the value stored in the first array element.

`*(anArray + 1)` refers to the value stored in the second array element.

etc.

Thus

`*(anArray + n)` is equivalent to `anArray[n]`, where `n` is an integer.

Recall this program:

```
//EE 285 - still more fun with pointers

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void){

   int anArray[8];   //an array of integers
   int* intPtr;      //a pointer to a integer
   int i;

   srand( (int)time(0) );

   for( i = 0; i < 8; i++ ){

      anArray[i] = rand()%10 + 1;
      intPtr = &anArray[i];
      printf( "%d: %d at %p.\n", i, *intPtr, (void*)intPtr );
    }

   printf( "\n\n" );
   return 0;
}
```

```c
//EE 285 - arrays & pointers

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void){

   int anArray[8];   //an array of integers
   int i;

   srand( (int)time(0) );

   for( i = 0; i < 8; i++ ){

      *(anArray + i) = rand()%10 + 1;
      printf( "%d: %d at %p.\n", i, *(anArray + i), (void*)(anArray + i) );
    }

   printf( "\n\n" );
   return 0;
}
```

It behaves identically.

```
0: 2 at 0x7ffeefbff450.
1: 7 at 0x7ffeefbff454.
2: 9 at 0x7ffeefbff458.
3: 6 at 0x7ffeefbff45c.
4: 6 at 0x7ffeefbff460.
5: 3 at 0x7ffeefbff464.
6: 6 at 0x7ffeefbff468.
7: 3 at 0x7ffeefbff46c.


Program ended with exit code: 0
```

A slight modification

```
//EE 285 - arrays & pointers

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void){

   int anArray[8];   //an array of integers
   int *aPtr;
   int i;

   srand( (int)time(0) );

   aPtr = anArray;      //pointer points to first array element

   for( i = 0; i < 8; i++ ){

      *aPtr = rand()%10 + 1;
      printf( "%d: %d at %p.\n", i, *aPtr, (void*)(aPtr) );
      aPtr = aPtr + 1;
    }

   printf( "\n\n" );
   return 0;
}
```

```
0: 5 at 0x7ffeefbff450.
1: 6 at 0x7ffeefbff454.
2: 4 at 0x7ffeefbff458.
3: 5 at 0x7ffeefbff45c.
4: 2 at 0x7ffeefbff460.
5: 1 at 0x7ffeefbff464.
6: 1 at 0x7ffeefbff468.
7: 4 at 0x7ffeefbff46c.

Program ended with exit code: 0
```

Same result.

Could use `aPtr++`, in place of `aPtr = aPtr + 1`.

```c
//EE 285 - find the max, redux

#include <stdio.h>

int main(void){

   int anArray[] = {-2, 7, 16, 9, -14, 4, 8, 9, 21, 1};
   int i, max = -50;

   for( i = 0; i < 10; i++ ){

      if( *(anArray + i) > max )
         max = *(anArray + i );
    }

   printf( “The biggest integer is %d.”, max );

   printf( "\n\n" );
   return 0;
}
```

```
The biggest integer is 21.

Program ended with exit code: 0
```

```c
//EE 285 - interleave 2 arrays
#include <stdio.h>

int main(void){
   int array1[] = {-2, 16, -14, 8, 21};
   int array2[] = {7, 9, 4, 9, 1};
   int array3[10];
   int *aPtr1, *aPtr2, *aPtr3;       //3 pointers
   int i = 0;

   aPtr1 = array1;
   aPtr2 = array2;
   aPtr3 = array3;

   while ( i++ < 5 ){
      *(aPtr3++) = *(aPtr1++);
      *(aPtr3++) = *(aPtr2++);
   }

   aPtr3 = array3;
   for(i = 0; i < 10; i++)
      printf( "%d ", *(aPtr3++) );

   printf( "\n\n" );
   return 0;
}
```

All of the action is inside the while loop. (Note, a `for` loop would have worked just as well.)

```
while ( i++ < 5 ){
    *(aPtr3++) = *(aPtr1++);
    *(aPtr3++) = *(aPtr2++);
 }
```

To see how it works, consider what happens during the first time through the loop (`i` = 0):

The first element of `array3` is made equal to the first element of `array1`. The pointers are incremented so that `aPtr3` is pointing to the second element of `array3` and `aPtr1` is pointing to the second element of `array1`.

Then, the second element of `array3` (pointed to by `aPtr3`) is made equal to the first element of `array2` (pointed to by `aPtr2`). Both of these pointers are incremented so that `aPtr3` is pointing to the *third* element of `array3` and `aPtr2` is pointing to the second element of `array2`.

Look at memory map at the very end of the first loop.

aPtr1 →

array1[0]	-2
array1[1]	16
array1[2]	-14
array1[3]	8
array1[4]	21

aPtr2 →

array2[0]	7
array2[1]	9
array2[2]	4
array2[3]	9
array2[4]	1

aPtr3 →

array3[0]	-2
array3[1]	7
array3[2]	
array3[3]	
array3[4]	
array3[5]	
array3[6]	
array3[7]	
array3[8]	
array3[9]	

If all of the nested increments are confusing at first, note that every step can be done explicitly, as shown below.

```
while ( i < 5 ){
    *aPtr3 = *aPtr1;
    aPtr3 = aPtr3 + 1;
    aPtr1 = aPtr1 + 1;
    *aPtr3 = *aPtr2;
    aPtr3 = aPtr3 + 1;
    aPtr2 = aPtr2 + 1;
    i++;
}
```

In fact, when first putting together a program like this, it might be clearer to start with something more explicit like this, and then combine the various parts to reduce the code to the version shown initially. Again, be careful with `i++` and `++i`, when incrementing.

Strings are arrays, too

```
//EE 285 - string length

#include <stdio.h>

int main(void){

   char nameArray[] = "Donald Trump";
   char *djtPtr;
   int length = 0;

   djtPtr = nameArray;

   while ( *djtPtr != '\0' ){
      length = length + 1;
      djtPtr = djtPtr + 1;
   }

   printf( "The string length is %d.", length );

   printf( "\n\n" );
   return 0;
}
```

```
The string length is 12.

Program ended with exit code: 0
```

Another version

```
//EE 285 - string length

#include <stdio.h>

int main(void){

   char nameArray[] = “Barak Obama";
   char *bhoPtr;
   int length = 0;

   bhoPtr = nameArray;

   while ( *(bhoPtr++) != ‘\0’ )
      length++;

   printf( “%s has %d characters.", nameArray, length );

   printf( "\n\n" );
   return 0;
}
```

```
Barak Obama has 11 characters.

Program ended with exit code: 0
```

One more example

```
//EE 285 - reverse a string

#include <stdio.h>

int main(void){

   char nameArray[] = “Abraham Lincoln”;
   char *alPtr, temp;
   int i, length = 0;

   alPtr = nameArray;

   while ( *(alPtr++) != ‘\0’ )
      length++;

   alPtr = nameArray;

   for( i = 0; i < length/2; i++){

      temp = *(alPtr + i);
      *(alPtr + i) = *(alPtr + length - 1 - i);
      *(alPtr + length - 1 - i) = temp;
   }

   printf( “reversed string is %s.”, nameArray );

   printf( "\n\n" );
   return 0;
}
```

```
reversed string is nlocniL maharbA.

Program ended with exit code: 0
```

There may be more efficient ways to do this.