Lesson with Arrays and Pointers

CSC315 Programming Language Concepts

04 October 2017

1. What does the statement **int*** np = &n; accomplish?

It assigns the address of a variable named n to a variable named np.

2. What is the difference between the function labeled f and the function labeled g?

The function labeled f effectively does nothing, as the integer value passed to it is only a copy and the return value is void. On the other hand, the function labeled g is passed a pointer to an address in memory, and the function changes the value at that address to 42.

1 Source code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void f( int n ) {
5    n = 42;
6  } // f( int )
7
8  void g( int *np ) {
9    *np = 42;
10 } // g( int* )
11
12 int main( int argc, char** argv ) {
```

```
13    // create an array in a familiar way
14    int primes[8];
15    primes[0] = 2;
16    primes[1] = 3;
17    primes[2] = 5;
18    primes[3] = 7;
19    primes[4] = 11;
20    primes[5] = 13;
21    primes[6] = 17;
22    primes[7] = 19;
23
24    // access elements with both index and pointer plus offset
25    printf( "primes[0]      = %2d\n", primes[0] );
26    printf( "*primes        = %2d\n", *primes );
27
28    printf( "\n" );
29
30    printf( "primes[1]      = %2d\n", primes[1] );
31    printf( "*(primes + 1) = %2d\n", *(primes + 1) );
32
33    printf( "\n" );
34
35    printf( "primes[2]      = %2d\n", primes[2] );
36    printf( "*(primes + 2) = %2d\n", *(primes + 2) );
37
38    printf( "\n" );
39
40    printf( "primes[3]      = %2d\n", primes[3] );
41    printf( "*(primes + 3) = %2d\n", *(primes + 3) );
42
43    printf( "\n" );
44
45    *(primes + 7) = 21;
46    printf( "primes[7] = %d\n", primes[7] );
47
48    printf( "\n" );
49
50    // another way to create an array
51    int* fibonacci = (int*) malloc( 8 * sizeof(int) );
52    // assign values using name of array plus index
53    fibonacci[0] = 1;
54    fibonacci[1] = 1;
55    fibonacci[2] = 2;
56    fibonacci[3] = 3;
57    // assign values using address of start of
58    // block of memory plus offset
```

```
59    *(fibonacci + 4) = 5;
60    *(fibonacci + 5) = 8;
61    *(fibonacci + 6) = 13;
62    *(fibonacci + 7) = 21;
63
64    int i;
65    for( i = 0; i < 8; i++ ) {
66      printf( "fibonacci[%1d] = %2d\n", i, fibonacci[i] );
67    } // for
68
69    printf( "\n" );
70
71    for( i = 0; i < 8; i++ ) {
72      printf( "address of fibonacci[%1d] = %lu\n",
73          i, (unsigned long) (fibonacci + i) );
74    } // for
75
76    printf( "\n" );
77
78    int oneInteger = 27;
79    printf( "before call to f(), oneInteger = %d\n", oneInteger );
80    f( oneInteger );
81    printf( "after call to f(), oneInteger  = %d\n", oneInteger );
82
83    printf( "\n" );
84
85    int oneIntegerArray[1];
86    oneIntegerArray[0] = 27;
87    printf( "before call to g(), oneIntegerArray[0] = %d\n",
88        oneIntegerArray[0] );
89    printf( "before call to g(), ");
90    printf( "address of oneIntegerArray[0] = %lu\n",
91        (unsigned long) &oneIntegerArray[0] );
92    g( oneIntegerArray );
93    printf( "after call to g(), oneIntegerArray[0]  = %d\n",
94        oneIntegerArray[0] );
95    printf( "after call to g(), " );
96    printf( "address of oneIntegerArray[0]  = %lu\n",
97        (unsigned long) &oneIntegerArray[0] );
98
99  } // main( int, char** )
```