
Returning a value from a function—review

As you remember, we can return one, and only one value from a function. We do this via the return statement. For example, this function returns the larger of two integers:

```
int larger(int a, int b)
{
  if (a<b)
    return a;
  else
    return b;
}
```

Returning two values from a function with structs

Suppose though, that a function is computing two values, not just one.
One way to return two values from a function is to make a new type—a struct.

A struct is a way to *create a new type*—in addition to int, double, char, char *, int *, etc.
A struct has members inside it: for example:

```
struct Point {
  double x;
  double y;
};
```

Notice the semicolons after each member of the struct, and after the end of the entire struct.

We can create a function that returns an item that is of type "struct Point", and by doing so, we can end up returning both a value for x, and a value for y.

For example, if we have an angle theta (θ) on the unit circle, then the value of a point on that circle is computed by these math formulas:

$x = r \cos \theta$ $y = r \sin \theta$ where for the unit circle, $r = 1$

In C code, we can represent that this way:

```
struct Point unitCirclePoint(double theta)
{
   struct Point p;
   p.x = cos(theta);
   p.y = sin(theta);
   return p;
}
```

Please turn over for more problems

Continued from other side

Returning two values from a function with pointers

Another way to return two values from a function is to use points.

Look back at the function on the other side that computes the larger of two numbers. Suppose though, that we wanted the function to compute BOTH the larger number and the smaller number, and return them to us.

We can use pointers to accomplish this. Look especially at the parts in bold in the programs below.

- In the main program, we set up two variables that will *receive* the answers from the function
- We pass the address of those variables to the function—the actual parameters are addresses, and the formal parameters are pointers
    - But as we know from lecture, **an address is a pointer, and a pointer is an address**
- Inside the function we dereference the pointers, and set the values

Here's an example—this example will be (or was) discussed more in lecture on

```
// computeBigSmallDemo.c  P. Conrad for CS16, Winter 2010
#include <stdio.h>

void computeBigAndSmall(int first, int second,
                        int *bigOnePtr, int *smallOnePtr);


int main(int argc, char *argv[])
{
  int firstOne, secondOne;
  int bigOne, smallOne;

  if (argc!=3)
    {
      printf("Usage: %s num1 num2\n", argv[0]); return 1;
    }

   // convert command line args
  firstOne = atoi(argv[1]); secondOne = atoi(argv[2]);
  // compute result
  computeBigAndSmall(firstOne, secondOne, &smallOne, &bigOne);
  // print result
  printf("The small one is %i and the big one is %i\n", smallOne, bigOne);

  return 0;
}

void computeBigAndSmall(int first, int second,
                        int *smallOnePtr, int *bigOnePtr)
{
  if (first < second)
    {
      // this reaches back to the variables pointed to in the calling function
      // and THOSE variables receive the values of first and second
      (*smallOnePtr) = first; (*bigOnePtr) = second;
    }
  else // first one is bigger, or they are the same
    {
      (*smallOnePtr) = second; (*bigOnePtr) = first;
    }
}
```

End of Handout for 02.16