

COP 3223: C Programming Spring 2009

Functions In C – Part 1

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
<http://www.cs.ucf.edu/courses/cop3223/spr2009/section1>

School of Electrical Engineering and Computer Science
University of Central Florida



Functions In C

- Most computer programs that solve real-world problems are much larger than the programs we have seen so far in this course.
- Software engineering has shown that the best way to approach writing large computer programs is a modular approach.
- A modular approach constructs the program from smaller pieces or **modules**, each of which is more manageable than a single large program.
- While we have not yet written programs using modules, we have shown you how to approach solving the problem and constructing your programs in a modular fashion, by first writing and testing small portions of your solution/code before moving on to other parts of the problem/program.



Functions In C

- In C, these modules are called **functions**. Most C programs are constructed by combining new functions that you write with “prepackaged” functions available in the C Standard Library.

Although the C Standard Library functions are not technically a part of the C language and are handled by the preprocessor before compilation occurs, they are nonetheless included with standard C systems and many programmers simply believe that they are part of the C language.

- So far in this semester, you have only written `main` functions yourself and called many different functions from the standard library, e.g., `printf`, `scanf`, `fscanf`, `fprintf`, etc..



Functions In C

- Functions are **invoked** (often referred to as **called**), by a **function call**, which specifies the function **name** and provides information (as **arguments** to the function) that the called function needs in order to perform its designated task.
- In C, everything is a function (including main), so functions call other functions to have tasks performed for them.
- In general:
 - A function performs some task.
 - A function may or may not need information (arguments) to perform its task.
 - A function may or may not return a value (or perhaps values).
 - A function can be called any number of times.



Functions In C

- Functions are **invoked** (often referred to as **called**), by a **function call**, which specifies the function **name** and provides information (as **arguments** to the function) that the called function needs in order to perform its designated task.
- In C, everything is a function (including main), so functions call other functions to have tasks performed for them.
- Good functional programming operates using a principle known as **information hiding**. In a functional setting, information hiding means that, I (as a caller of the function) know what kind of task the function can perform for me and what information I need to send to it in order for it to perform its task, BUT, I do not know how the function goes about actually performing the task; that information is hidden from me by the function.



Functions In C

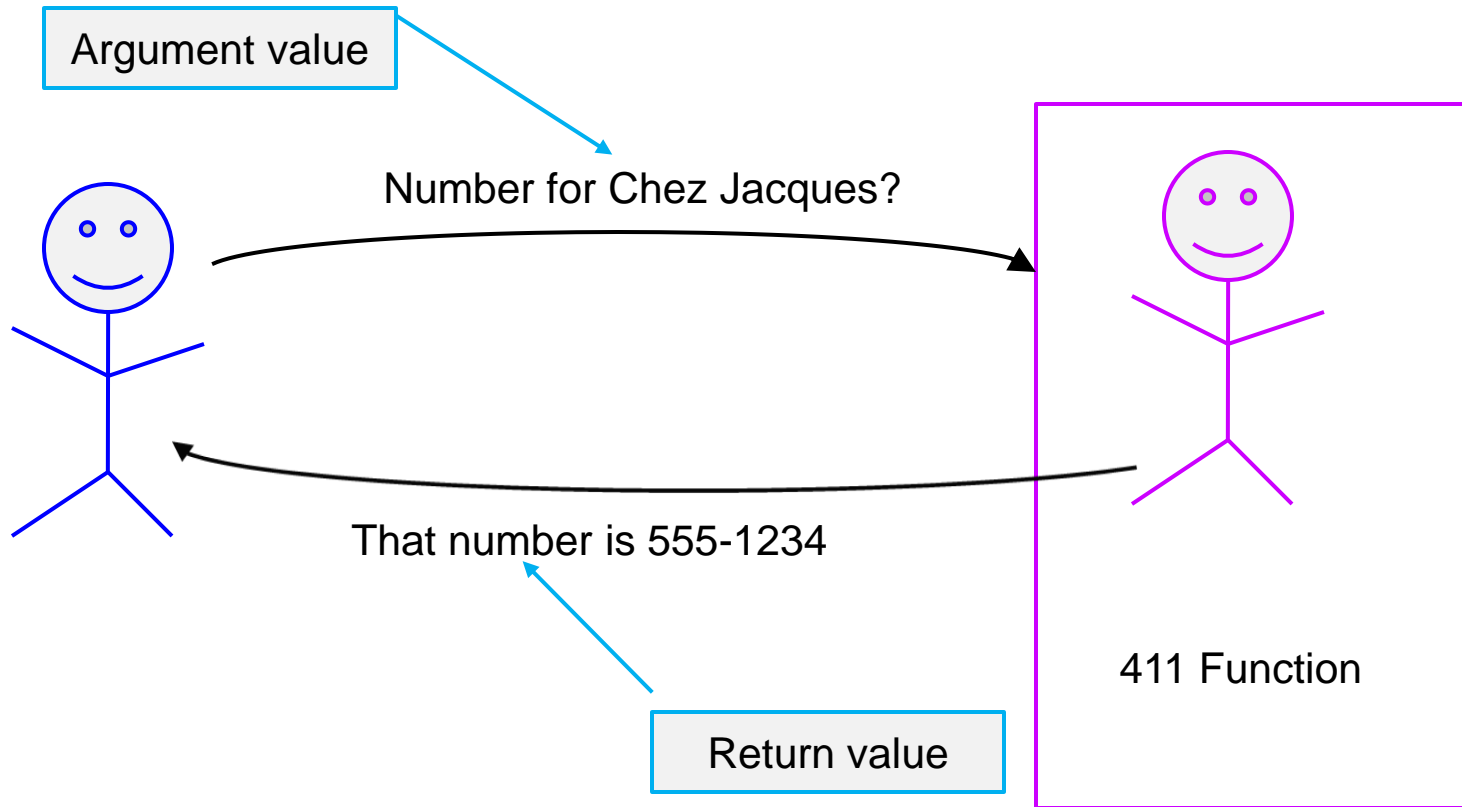
- Consider the following analogy to help you understand how function calling works:

Suppose that you dial a 411 operator to find out the phone number for a nice restaurant you've heard about that you want to call to make a reservation to take your girlfriend/boyfriend to dinner. You call 411 (consider 411 to be a function) and give them the name of the restaurant (the value of the argument the function needs to operate). The 411 operator, somehow finds the number for the restaurant (you don't really know how they did it), and tells you the number for the restaurant (the function's return value).

Basically think of a function as a "black box". You know what it does (i.e., the service it provides), but you have no idea how it accomplishes its task. This information hiding principle is very important to the programmer's flexibility. In this manner, you are free to modify how the function works, without requiring any program that uses the function to be modified in any way.



Functions In C



Defining Functions In C

- The general form of a function definition in C is:

```
return-type  function-name ( parameters ) {  
    declarations  
    statements  
}
```

- The return-type of a function is the type of value that the function returns. Recall that all of our `main` functions have returned a value of type `int`.
- **A function in C cannot return an array.**
- Specifying that the return type is `void` indicates that the function does not return a value.



Defining Functions In C

```
return-type  function-name ( parameters ) {  
    declarations  
    statements  
}
```

- Inside the parentheses of a function definition is a list of parameters. Each parameter is preceded by its type definition and multiple parameters are separated by commas.
- **NOTE: the type for each parameter must be explicitly listed, even when several parameters have the same type.**
- If the function requires no parameters, the keyword `void`, should appear in the parentheses.



Defining Functions In C

```
return-type  function-name ( parameters ) {  
    declarations  
    statements  
}
```

- Functions can be thought of as little programs in and of themselves. Therefore, to accomplish the task that a function is to perform, it may require variables. These variables must be declared inside the function definition and they cannot be seen, examined, modified, or used in any way outside of the function in which they are declared.
- Variables declared inside a function are said to be **local** to that function.
- **NOTE: The parameters to a function are also considered to be local variables.**



Defining Functions In C

- Let's write a simple program that includes a function that will calculate x^y and return the value of this to the caller. This was basically the last problem on the exam.
- Notice that our function, let's call it `power`, will require two parameters, both of which are of the `int` type, and we expect it to return a value which is also an `int` type.
- Let's write the function first, and then include it in a program.



Defining Functions In C

```
int power (int x, int y)
{
    int i; //loop control variable
    int result; //the value of xy to return

    result = 1; //initialize result
    for (i = 1; i <= y; ++i) {
        result += x * x;
    } //end for stmt
    return result;
} //end function power
```



```
1 //Functions In C - Part 1 - Simple function example
2 //uses a power function to calculate x^y
3 //February 23, 2009    Written by: Mark Llewellyn
4
5 #include <stdio.h>
6
7 //function power - returns x^y
8 int power(int x, int y)
9 {
10     int i; //loop control variable
11     int result; //value of x^y returned to user
12
13     result = 1;
14     for (i = 1; i <= y; ++i) {
15         result *= x;
16     } //end for stmt
17     return result;
18 } //end function power
19
20 int main()
21 {
22     int number1, number2; //two integers
23     printf("Please enter two integer numbers x & y where x^y:\n");
24     scanf("%d%d", &number1, &number2);
25     printf("The result of x^y is: %d\n", power(number1,number2));
26     printf("\n\n");
27     system("PAUSE");
28     return 0;
29 } //end main function
```

The function definition

This is basically the same technique that I used on the exam key to solve this problem.

The function call



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3...  
Please enter two integer numbers x & y where x^y:  
3  
4  
The result of x^y is: 81  
  
Press any key to continue . . . .
```



Calling Functions In C

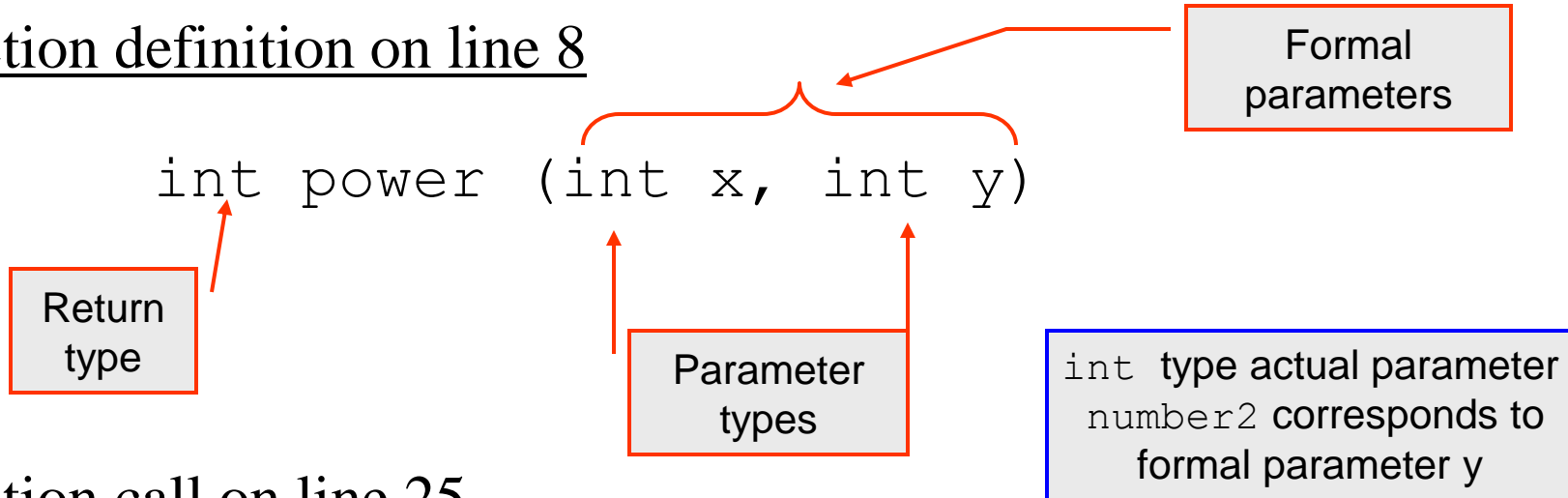
- Function definitions contain a return type for the function and the type of each parameter to the function.
- The parameters in the function definition are referred to as **formal parameters**.
- When a call is made to a function, no direct reference is made to either the return type of the function nor to the types of the individual parameters.
- The call to the function must contain the same number of parameters, referred to as actual parameters, as the function definition and their types must match in a 1:1 correspondence to the formal parameters.
- The function call must be used in a manner so that the type of value returned is consistent with its use.



Calling Functions In C

- Let's look more closely at the function definition and the function call in the program on page 13.

The function definition on line 8



The function call on line 25

```
printf ("...%d\n", power (number1, number2) );
```

return int type

int type actual parameter
number1 corresponds to
formal parameter x



Calling Functions In C

- Let's suppose at the time of the call to function `power`, that the variable `number1 = 4` and the variable `number2 = 3`. The values of the actual parameters are copied into the formal parameters at the time of the function call.

definition

```
int power (int x, int y)
```

4

3

call

```
printf ("...%d\n", power (number1, number2) );
```

Return 64



Defining Functions In C

- To consider how the principle of information hiding is applied to functions, let's revisit the program we just wrote that used a function to compute the value of x^y .
- In our first version of the program the function computing the value of x^y used a for loop and calculated x^y by repeatedly multiplying x by itself y times.
- The second version of the program shown on the next page solves exactly the same problem, but the function uses an entirely different approach to calculate the value of x^y . In the second version the function just calls the `pow` function found in the `math.h` library.



```
1 //Functions In C - Part 1 - Simple function example illustrates info. hiding
2 //when compared with original simple function example on page 13.
3 //this version uses the pow function found in math.h to generate the result
4 //February 23, 2009    Written by: Mark Llewellyn
5
6 #include <stdio.h>
7 #include <math.h>
8
9 //function power - returns x^y
10 int power(int x, int y)
11 {
12     return (int) (pow(x,y));
13 }//end function power
14
15 int main()
16 {
17     int number1, number2; //two integers
18     printf("Please enter two integer numbers x & y where x^y:\n");
19     scanf("%d%d", &number1, &number2);
20     printf("The result of x^y is: %d\n", power(number1,number2));
21     printf("\n\n");
22     system("PAUSE");
23     return 0;
24 }//end main function
```

Notice that the way the function `power` determines the value to return has changed drastically, but the “interface”, the way it appears on the outside, has not changed. No changes were made, at all, in the main function where `power` was called.



Writing Functions In C

- Remember way back when we first started looking at repetition structures in C (seems like a long time ago now doesn't it – but it was actually only 1 month ago almost to the day!); one of our first examples was computing the sum of the first n integers.
- Let's create a program that uses a function to generate the sum of the first n integers, where the function will return an integer value and requires a single integer parameter which is the value of n . The main function will simply ask the user to enter the value of n and print out the result returned from the function.
- We'll write two versions of the program which differ only in how the function is called.



```
1 //Functions in C - Part 1 - Using a function to sum first n integers
2 //
3 //February 24, 2009   Written by: Mark Llewellyn
4
5 #include <stdio.h>
6
7 int sumFirstNIntegers(int x) {
8     int i; //loop control
9     int runningSum = 0; //running sum and the return value
10
11     for (i = 1; i <= x; ++i) {
12         runningSum += i;
13     } //end for stmt
14     return runningSum;
15 } //end function sumFirstNIntegers
16
17 int main()
18 {
19     int n; //user wants sum of first n integer values
20
21     printf("\nEnter the value of N, where you want the sum of 1...N\n");
22     scanf("%d", &n);
23     printf("The sum of the first %d integers is: %d \n", n, sumFirstNIntegers(n));
24
25     printf("\n\n");
26     system("PAUSE");
27     return 0;
28 } //end main function
```

In this version of the program the function call is used as a parameter to a printf statement. Notice that the conversion specifier (%d) must match the return type of the function.



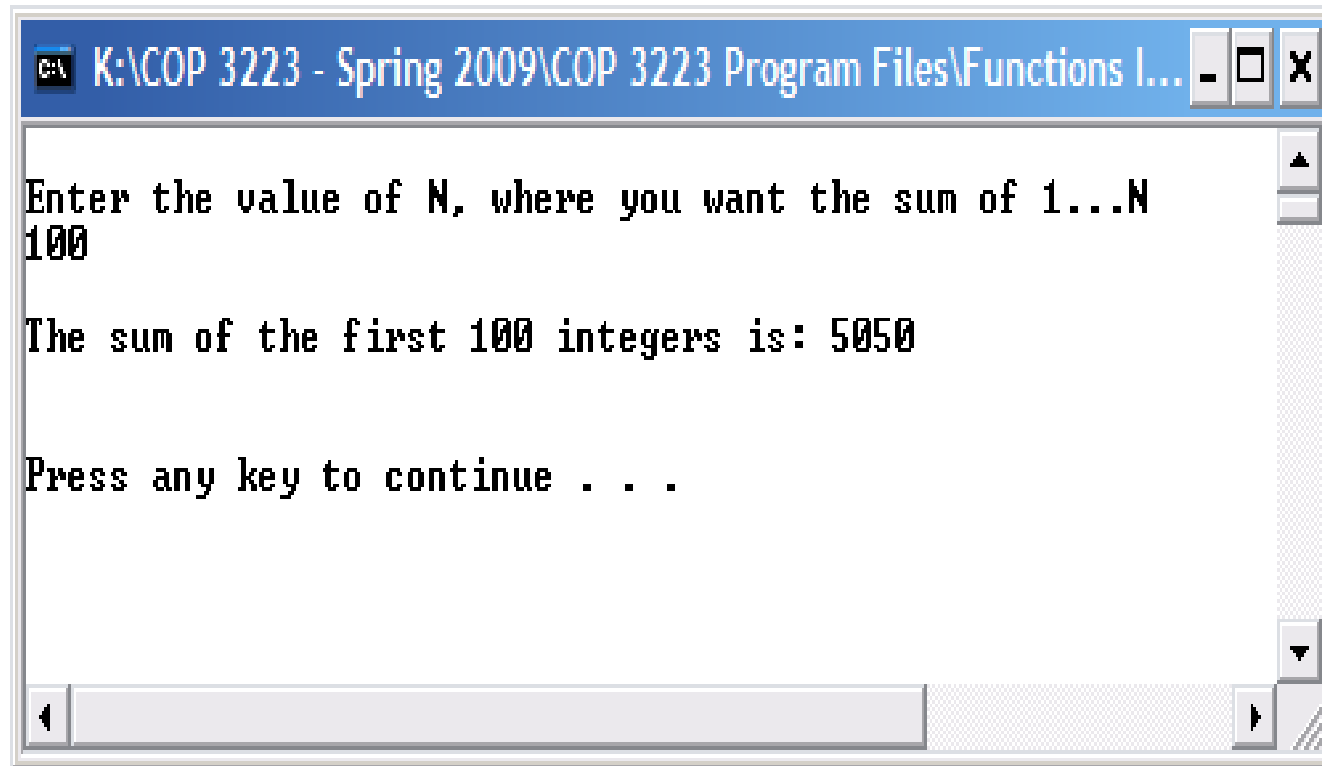
```
K:\COP 3223 - Spring 2009\COP 3223 Program Files\Functions In C ...  
Enter the value of N, where you want the sum of 1...N  
100  
The sum of the first 100 integers is: 5050  
Press any key to continue . . .
```



```
2 //
3 //February 24, 2009   Written by: Mark Llewellyn
4
5 #include <stdio.h>
6
7 int sumFirstNIntegers(int x) {
8     int i; //loop control
9     int runningSum = 0; //running sum and t
10
11     for (i = 1; i <= x; ++i) {
12         runningSum += i;
13     } //end for stmt
14     return runningSum;
15 } //end function sumFirstNIntegers
16
17 int main()
18 {
19     int n; //user wants sum of first n integer values
20     int summation; //the value returned by the function
21
22     printf("\nEnter the value of N, where you want the sum of 1...N\n");
23     scanf("%d", &n);
24     summation = sumFirstNIntegers(n);
25     printf("\nThe sum of the first %d integers is: %d \n", n, summation);
26
27     printf("\n\n");
28     system("PAUSE");
29     return 0;
30 } //end main function
```

Same program, the difference is how the function call is made. In this case, the function call is used on the right side of an assignment statement. Note that the variable on the left hand side of the assignment must be an int type to match the return type of the function.



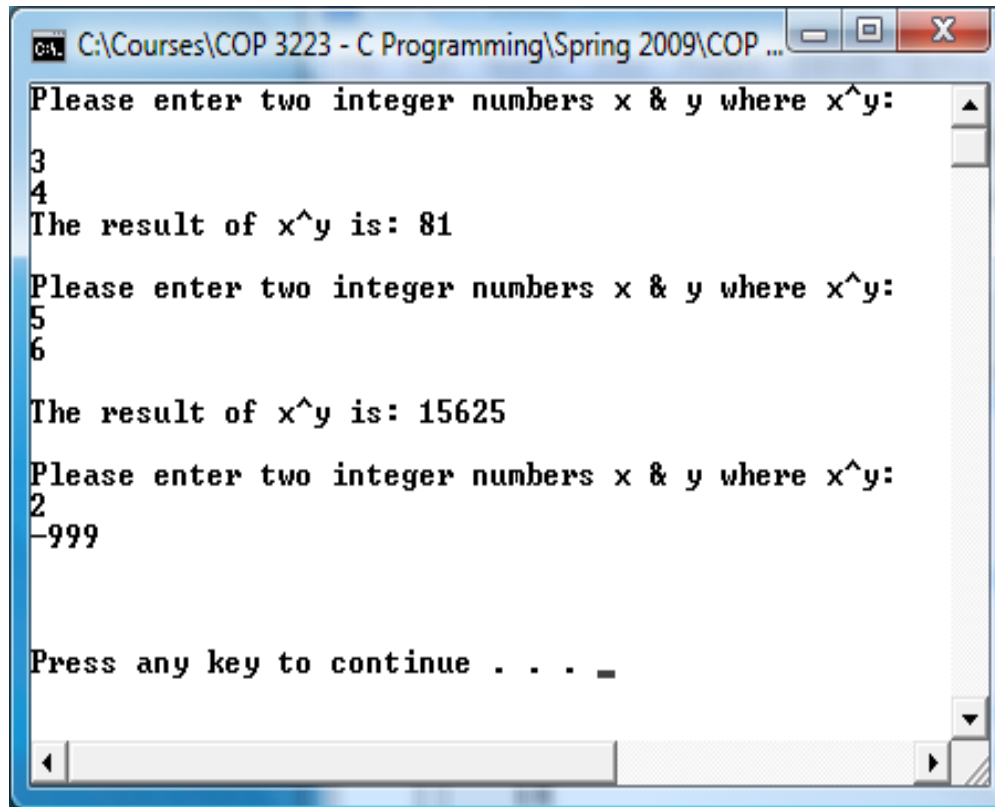


```
K:\COP 3223 - Spring 2009\COP 3223 Program Files\Functions I...
Enter the value of N, where you want the sum of 1...N
100
The sum of the first 100 integers is: 5050
Press any key to continue . . .
```



Practice Problems

1. Modify the example on page 3 so that the user is repeatedly asked to enter two integer values and the result is displayed, until the user enters a value of -999 for either x or y, at which point the program terminates. different happens.



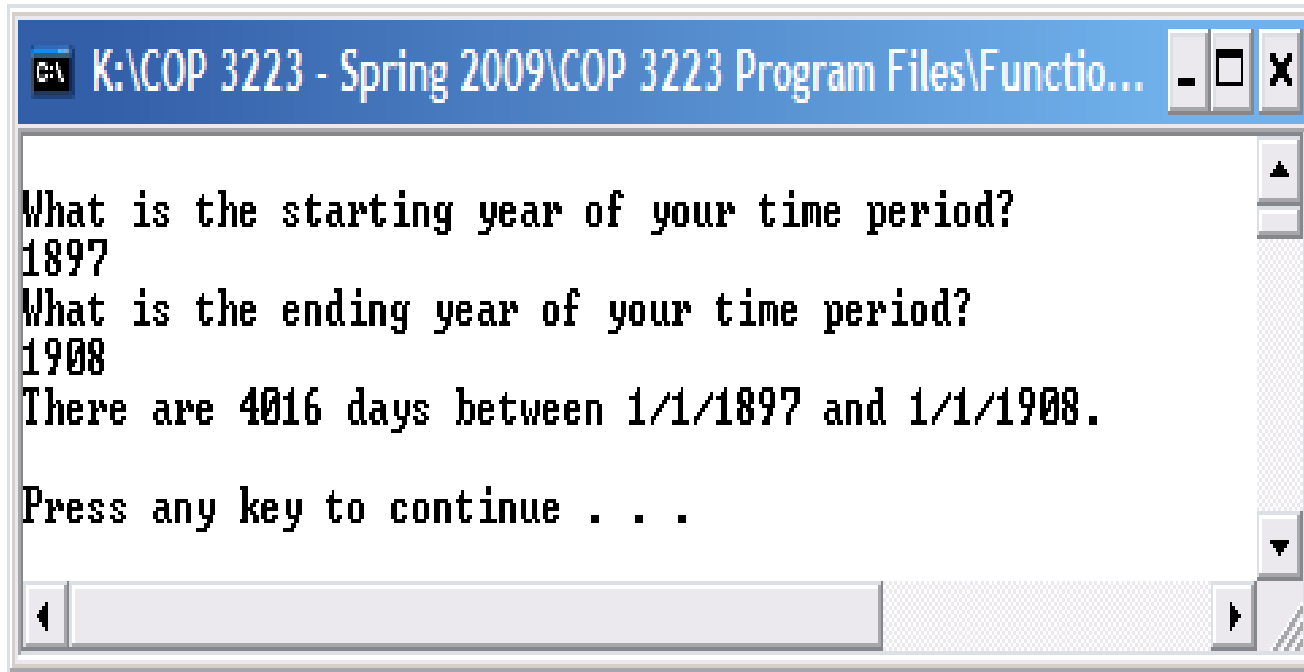
```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP ...
Please enter two integer numbers x & y where x^y:
3
4
The result of x^y is: 81
Please enter two integer numbers x & y where x^y:
5
6
The result of x^y is: 15625
Please enter two integer numbers x & y where x^y:
2
-999

Press any key to continue . . .
```



Practice Problems

2. Modify your solution to the leap problem from assignment #2 so that a function is used to determine if the current year being considered within the range of years in question is a leap year, or not.

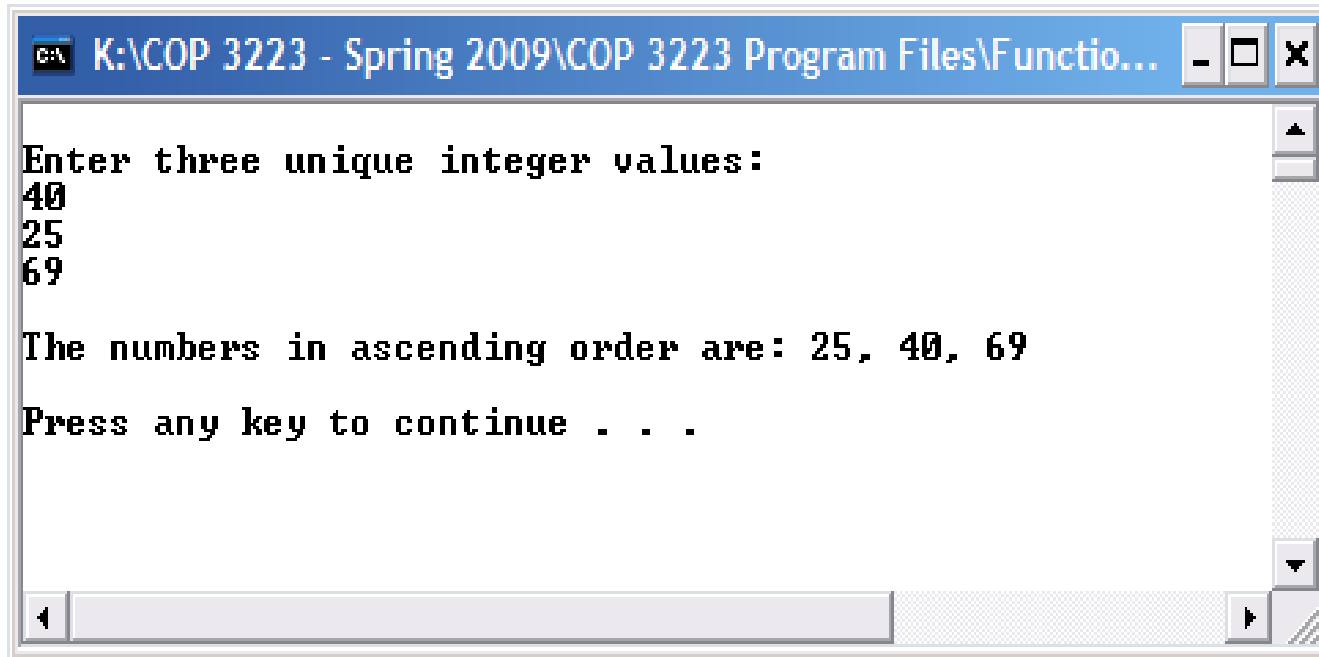


```
K:\COP 3223 - Spring 2009\COP 3223 Program Files\Functio...
What is the starting year of your time period?
1897
What is the ending year of your time period?
1908
There are 4016 days between 1/1/1897 and 1/1/1908.
Press any key to continue . . .
```



Practice Problems

3. Write a C program that uses two functions each of which accept 3 integer values as parameters and one of the functions returns the smallest value of the 3 parameters and the other function returns the largest of the three parameters. The main function should then print the three values in ascending order.



```
Enter three unique integer values:  
40  
25  
69  
  
The numbers in ascending order are: 25, 40, 69  
  
Press any key to continue . . .
```

