Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2013

Lab 2 - C Functions

Objective

The objective of this lab is to design, code and test some simple functions in C.

Students who took ECOR 1606 will find that this lab reviews much of the C/C++ taught in that course (pretty well everything up to but not including arrays).

Students who took SYSC 1005 wrote functions similar to these in Python. You've already learned all the programming constructs (functions, if statements, loops) you'll require; the only thing that's new is that you'll use C instead of Python to implement algorithms that use those constructs.

Attendance/Demo

To receive credit for this lab, you must make a reasonable effort to complete the exercises and demonstrate the code you complete.

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. Finish any exercises that you don't complete by the end of the lab on your own time. Also, you must submit your lab work to cuLearn by the end of the lab period.

General Requirements

For those students who already know C or C++: when coding your solutions, do not use arrays, structs or pointers. They aren't necessary for this lab.

None of the functions you write should produce console output; i.e., contain printf statements.

You have been provided with file main.c. This file contains incomplete implementations of four functions you have to design and code. It also contains a *test harness* (functions that will test your code) and a main function that calls these test functions. **Do not modify main() or any of the test functions.**

Instructions

1. Create a folder named Lab 2.
2. Launch Pelles C, and create a new project named functions inside the Lab 2 folder. The project type must be Win32 Console program (EXE). (If you've forgotten how to do this, review last week's lab.) You should now have a folder named functions inside a folder named Lab 2 (check this).
3. Download file main.c from cuLearn. Move this file into your functions folder. You must also add main.c to your project: from the menu bar, select Project > Add files to project... In the dialogue box, select main.c, then click Open. An icon labelled main.c will appear in the Pelles C project window.
4. Build the project. It should build without any compilation or linking errors.
5. Execute the project. The test harness will report several errors as it runs, which is what we'd expect: you haven't started working on the functions that are being tested. Read the console output carefully. The output indicates which of your functions is about to be called, the values that are being passed as arguments, the result we expect the function to return, the actual value returned by the function, and a summary of how many tests passed and failed.
6. Open main.c.
7. Design and code the functions described in Exercises 1 through 4. As you write each function, build the project and execute it to run the test harness. After one of your functions passes all of its tests, start on the next exercise.

Exercise 1

The factorial $n!$ is defined for a positive integer n as:

$$n! = n \times (n\text{-}1) \times (n\text{-}2) \times \ldots \times 2 \times 1.$$

For example, $4! = 4 \times 3 \times 2 \times 1 = 24$.

0! is defined as: $0! = 1$.

Write a C function called factorial that has one parameter, n. The function header is:

```
int factorial(int n)
```

This function calculates and returns n!. Your function should assume that n is 0 or positive; i.e., it should not check if n is passed a negative value when the function is called.

Aside: because the return type of the function is int and $n!$ grows rapidly as n increases, this function will be unable to calculate factorials greater than 15! on computers that use 32-bit integers (a variable of type int can't hold integers that are that large.).

Exercise 2
Suppose that we have n distinct objects. There are $n!$ ways of ordering or arranging n objects, so we say that there are $n!$ permutations that can be obtained with n objects. If we have n objects and select k of the objects, then there are $n!/(n-k)!$ possible ways to do this, each containing k objects. That is, the number of different ordered subsets of n objects taken k at a time is $n!/(n-k)!$.

For example, suppose we have the set of integers {1, 2, 3, 4} and want to select 2 integers at a time from this set. There are 4! / (4 - 2)! = 12 ways to do this: {1, 2}. {1, 3}, {1, 4}, {2, 1}, {2, 3}, {2, 4}, {3, 1}, :{3, 2}, {3 4}, {4, 1}, {4, 2} and {4, 3}.

Write a C function named permute that has two parameters, n and k, and has return type int. This function returns the number of ordered subsets of n objects taken k at a time. Your function should assume that n and k are positive and that n >= k. For each factorial calculation that's required, your permute function must call the factorial function you wrote in Exercise 1. In other words, don't copy/paste code from factorial into permute.

Exercise 3
Permutations are concerned with order, but combinations are not. Given n distinct objects, there is only one combination of n objects taken n at a time. The number of combinations of n objects taken k at a time is $n!/((k!)(n-k)!)$.

For example, suppose we have the set of integers {1, 2, 3, 4} and want to select 2 integers at a time from this set, without regard to order. There are 4! / (2! * (4 - 2)!) = 6 combinations {1, 2}. {1, 3}, {1, 4}, {2, 3}, {2, 4} and {3 4}.

Write a C function named combine that has two parameters, n and k, and has return type int. This function returns the number of combinations of n objects taken k at a time. Your function should assume that n and k are positive and that n >= k. Your combine function must call your permute and factorial functions.

Exercise 4

The cosine of an angle x can be computed from the following infinite series:

$$\cos x = 1 - x^2/2! + x^4/4! - x^6/6! + \ldots$$

Write a C function named `cosine` that has two parameters, x and n, and has return type double. This function calculates and returns the cosine of angle x by calculating the first n terms of the series. Note that x is measured in radians, not degrees. (Recall that there are Π radians in 180 degrees.) Your cosine function must call your factorial function.

Your cosine function must call C's pow function. The function prototype is:

```
// Return x raised to the y power.
double pow(double x, double y);
```

This prototype is found in header file math.h, so you'll have to remember to put

```
#include <math.h>
```

at the start of your program.

Note that it's o.k. to pass an integer argument to pow. C will convert this value to a double value when the function is called.

For this exercise, we'll use a different approach to testing the function. The C standard library has a function called cos, so we'll compare the cosines calculated by this function with the values returned by your cosine function.

Here is part of the test function for this exercise. It first calls C's cos function to calculate the cosine of 0 radians. It then repeatedly calls your cosine.function to calculate the cosine of 0 radians. The first time cosine is called, only the first term of the series is calculated. The second time cosine is called, two terms of the series are summed. During the final iteration, seven terms are summed. When you run this code, you'll see how rapidly the value returned by cosine converges on the correct value (as returned by C's cos function).

```
printf("Calculating cosine of 0 radians\n");
printf("Calling standard library cos function: %.8f\n", cos(0));
printf("Calling cosine function\n");
for (int i = 1; i <= 7; i += 1) {
   printf("# terms = %d, result = %.8f\n", i, cosine(0, i));
}
printf("\n");
```

Notice that the character string argument in the fourth call to printf is "# terms = %d, result = %.8f\n". When this string is displayed, the %d will be replaced by the value of variable i and the %.8f will be replaced by the value returned by cosine. %.8f specifies that this value should be formatted as a double (a real number), with 8 digits after the decimal point.

The test function calculates the cosines of 0 radians (0 degrees), Π /4 radians (45 degrees), Π/2 radians (90 degrees), and Π radians (180 degrees). For each of these values, we have cosine calculate 1 term of the series, 2 terms of the series, etc., all the way up to 7 terms. How close are the values returned by cosine to the values returned by cos?

Wrap-up

1. Remember to have a TA review and grade your solutions to the exercises before you leave the lab.
2. The next thing you'll do is package the project in a ZIP file (compressed folder). From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. Edit the File name field, changing the name from functions to lab2. Click Save. Pelles C will create a compressed (zipped) folder named lab2.zip, which will contain copies of the the source code and several other files associated with the project. (The original files will not be removed). The compressed folder will be stored in your project folder (i.e., folder functions).
3. Log in to cuLearn and click the Submit Lab 2 link. Submit lab2.zip. Instructions for submitting file-based assignments in cuLearn can be found here:

   www5.carleton.ca/culearnsupport/students/evaluation-tools/assignments/

   Remember to click the Send for marking button (See Step 6 of the instructions).

Posted: Sunday, September 15, 2013.