

CS 211, Winter 2004  
Lab Assignment L1: Basic C++  
Assigned: Friday, Jan. 9, Due: Friday, Jan. 16, 11:00 AM

Brian M.Dennis, Instructor  
Bin Lin, Tom Lechner, Teaching Assistants

## Aims and Goals

The purpose of this assignment is to introduce you to the basics of C++ programming. After you complete this lab, you should be able to do the following:

- Declare and define C++ variables
- Explain the difference between C++ integer and floating point variables
- Declare and define C++ procedures
- Write basic expressions and statements in the core syntax of C++, including integer and floating point arithmetic expressions, and boolean expressions
- Use conditional statements to control the flow of execution in a program
- Use iteration constructs to implement repetitive calculations
- Use basic C++ I/O facilities to read input from a user, do some computation, and generate simple text output
- Write a complete set of C++ source files and compile them into a standalone executable program

At this point in the course, we haven't really introduced any aggregate data structures. Thus, we will focus our exercises on mathematical calculations that can be done with integer and floating point variables. Combining C++'s core expressions with procedures, iteration, and conditionals we can still get some real work done.

## Recursive and Iterative Factorial (10 Points)

The factorial of a nonnegative integer  $n$  is written  $n!$  (pronounced " $n$  factorial") and is defined as follows:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1, n > 1$$

$$n = 1, \forall n = 0, 1$$

For example,  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ . For this part of the lab, you will implement a factorial procedure and use it for multiple purposes.

Write a short C++ program that reads a nonnegative integer, computes the factorial of the integer in three different ways, and prints each result.

- First, implement a linear recursive procedure `fact_lrecursive`.
- Second, implement an iterative recursive procedure `fact_irecursive`.
- Third, implement an iterative procedure `fact_iterative` that uses a C++ iteration statement to implement factorial.

To get you going we will provide a skeleton project in which you only have to fill in the three procedures, along with instructions on how to build and run the program. You may need to add additional helper functions to the ones provided in the skeleton code.

Test your code on a range of values including,  $n$  equals -10, 0, 10, and 20. For any odd behavior, explain in the comments of your code why the behavior happened and a reasonable means of handling the situation.

## Calculating $\pi$ (10 Points)

We can calculate  $\pi$  from the following infinite series:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} \dots$$

Print a table that shows the value of  $\pi$  approximated by 1 term of the series, by two terms, by three terms, etc. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14149?

To get you going we will provide a skeleton project in which you only have to fill in the  $\pi$  approximation procedure.

## Reuse and Separate Compilation (10 Points)

For this program we will provide a main driver program, which you are not allowed to change. This driver program will call specific separately compiled procedures. You must implement the following procedures in a file named `estimate.cpp` which you have to write from scratch. To test it you will have to compile `estimate.cpp` into an object file and link it with the object file compiled from the driver program we give you.

- Write a procedure `e_estimate` that estimates the value of the mathematical constant  $e$  by using the formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

- Write a procedure that `ex_estimate` estimates the value of  $e^x$  by using the formula:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

In both cases, reuse your code from the previous exercises.

## Exponentiation by Repeated Squaring (10 Points)

Given two integers  $b$  and  $n$  we can calculate  $b^n$  two different ways, through repeated multiplication or through repeated squaring, both of which we can define recursively:

- Repeated multiplication:  $b^n = b \cdot b^{n-1}$
- Repeated squaring:  
 $b^n = (b^{n/2})^2$  if  $n$  is even  
 $b^n = b \cdot b^{n-1}$  if  $n$  is odd

For this problem, you are to write a procedure that implements an iterative version of repeated squaring exponentiation. A first step is to write an iterative process, recursive version `expo_recursive` similar to `expo_iter` as described in the first chapter of SICP. An interesting topic that SICP does not cover until much later, is that recursive, iterative processes are easily transformed into C++ style loops. Good Scheme interpreters/compiler will do this automatically, making recursive procedures as fast as an iterative loop. Turn your `expo_recursive` procedure into `expo_iter`, a procedure that makes no procedure calls of its own.

To get you going we will provide a skeleton program in which you only have to fill in the two exponentiation procedures.

## Towers of Hanoi (10 Points)

Completely from scratch, write a C++ program that solves problem 3.41 from Deitel & Deitel, The Towers of Hanoi problem.

## Extra Credit: Primality Testing (10 Points)

Section 1.2.6 of SICP presents two different means of testing for prime numbers. Completely from scratch, write a C++ program that implements both the deterministic and probabilistic primality testing functions. Try to find the three smallest prime numbers larger than 1,000, larger than 10,000 and larger than 100,000.