**The Assignment:**

You will implement and test four short recursive functions. With the proper use of recursion, none of these function should require more than a dozen lines of code.

**Purposes:**

Ensure that you can write and test small recursive functions.

**Before Starting:**

Read all of Chapter 9, especially Sections 9.1 and 9.2.

**File that you must write:**

1. rec\_fun.cpp: This file should contain the implementations of the four functions described below. You might also want to put the functions prototypes in a separate file rec\_fun.h and write a test program that includes rec\_fun.h.

**Implement and Test Four Small Recursive Functions**

**1. Triangle Pattern**

void triangle(ostream& outs, unsigned int m, unsigned int n)

// Precondition: m <= n

// Postcondition: The function has printed a pattern of 2\*(n-m+1) lines

// to the output stream outs. The first line contains m asterisks, the next

// line contains m+1 asterisks, and so on up to a line with n asterisks.

// Then the pattern is repeated backwards, going n back down to m.

/\* Example output:

triangle(cout, 3, 5) will print this to cout:

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*/

Hint: Only one of the arguments changes in the recursive call. Which one?

**2. Section Numbers**

Write a function with this prototype:

void numbers(ostream& outs, const String& prefix, unsigned int levels);

The function prints output to the ostream outs. The output consists of the String prefix followed by "section numbers" of the form 1.1., 1.2., 1.3., and so on. The levels argument determines how many levels the section numbers have. For example, if levels is 2, then the section numbers have the form x.y. If levels is 3, then section numbers have the form x.y.z. The digits permitted in each level are always '1' through '9'. As an example, if prefix is the string "THERBLIG" and levels is 2, then the function would start by printing:

THERBLIG1.1.

THERBLIG1.2.

THERBLIG1.3.

and end by printing:

THERBLIG9.7.

THERBLIG9.8.

THERBLIG9.9.

The stopping case occurs when levels reaches zero (in which case the prefix is printed once by itself followed by nothing else).

The string class from <string> has many manipulation functions, but you'll need only the ability to make a new string which consists of prefix followed by another character (such as '1') and a period ('.'). If s is the string that you want to create and c is the digit character (such as '1'), then the following statement will correctly form s:

s = (prefix + c) + '.';

This new string s can be passed as a parameter to recursive calls of the function.

**3. A Teddy Bear Picnic**

This question involves a game with teddy bears. The game starts when I give you some bears. You can then give back some bears, but you must follow these rules (where n is the number of bears that you have):

1. If n is even, then you may give back exactly n/2 bears.
2. If n is divisible by 3 or 4, then you may multiply the last two digits of n and give back this many bears. (By the way, the last digit of n is n%10, and the next-to-last digit is ((n%100)/10).
3. If n is divisible by 5, then you may give back exactly 42 bears.

The goal of the game is to end up with EXACTLY 42 bears.

For example, suppose that you start with 250 bears. Then you could make these moves:

 --Start with 250 bears.

 --Since 250 is divisible by 5, you may return 42 of the bears, leaving you with 208 bears.

 --Since 208 is even, you may return half of the bears, leaving you with 104 bears.

 --Since 104 is even, you may return half of the bears, leaving you with 52 bears.

 --Since 52 is divisible by 4, you may multiply the last two digits (resulting in 10) and return these 10 bears. This leaves you with 42 bears.

 --You have reached the goal!

Write a recursive function to meet this specification:

bool bears(int n)

// Postcondition: A true return value means that it is possible to win

// the bear game by starting with n bears. A false return value means that

// it is not possible to win the bear game by starting with n bears.

// Examples:

// bear(250) is true (as shown above)

// bear(42) is true

// bear(84) is true

// bear(53) is false

// bear(41) is false

Hint: To test whether n is even, use the expression ((n % 2) == 0).

**4. Fibonacci sequence**

(Performance comparison to calculate Fibonacci Numbers)

In C++, you can measure the time it takes to execute certain task in this manner:

**#include "stdafx.h"**

**#include <iostream>**

**#include <chrono>**

**using namespace std;**

**using namespace std::chrono;**

**int main()**

**{**

**high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();**

**int i;**

**for (i = 0; i < 1000000; i++)**

**;**

**high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();**

**duration<double> time\_span = t2 - t1;**

**std::cout << "It took " << time\_span.count() << " seconds.";**

**std::cout << std::endl;**

**}**

In this problem assignment, compare the performance of generating a Fibonacci sequence with recursion and iteration:

A Fibonacci sequence is defined as f1=1, f2=1, fn= fn-1 + fn-2

You’ll generate this sequence in 2 different ways:

1. You can easily write a Fibonacci sequence with recursion.
2. You can also calculate Fibonacci numbers using an iterative approach:

It can be shown that the 2X2 matrix A, with

a11=1, a12=1, a21=1, a22=0 can generate the Fibonacci sequence. Specifically, When A multiplies itself n times,

a11=fn+1,

a12=fn,

a21=fn,

a22=fn-1.

Here is a screenshot of the calculation using Wolfram Alpha for the matrix raised to a power of 40.



It shows you that f41 = 165580141, f40 = 102334155, f39 = 63245986

Compare the performance (using the chrono library) with the recursive approach, for n up to 45. You will find the following code useful:

**int A[2][2] = { { 1, 1 },{ 1, 0 } };**

**int B[2][2] = { { 1, 1 },{ 1, 0 } };**

**int temp[2][2] = { { 0, 0 },{ 0, 0 } };**

**while (n >= 0) {**

**for (int i = 0; i < 2; i++)**

**for (int k = 0; k < 2; k++) {**

**temp[i][k] = 0;**

**for (int j = 0; j < 2; j++)**

**temp[i][k] += A[i][j] \* B[j][k];**

**}**

**for (int i = 0; i < 2; i++)**

**for (int j = 0; j < 2; j++)**

**B[i][j] = temp[i][j];**

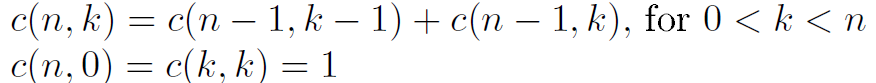
**n--;**

**}**

Prob 5

In math, the binomial coefficient c(n,k) is defined as:



Blaise Pascal discovered a recursive relationship among the binomial coefficients by arranging them in a triangle. (n is the row number, k is the column number): 

For example,



Now create two functions to calculate the binomial coefficients given non-negative integers n and k.

(a) Recursively, int cr(int n, int k)

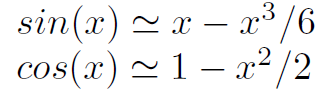
(b) Iteratively, int ci(int n, int k)

Prob 6

Congratulations! You have just been hired by a company specialized in making scientific calculators. Your first task: write a sine and a cosine function in C++.

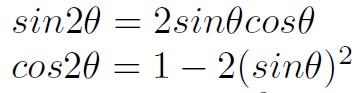
Given the following:

When the angles (measured in radians) are small,



In this problem, you can use the above approximations above when -0.005 < x < 0.005

Also, in trigonometry we learn the "double angle formula":



Now create 2 functions,

double s(double x);

double c(double x);

to calculate the sine and cosine functions using recursion.

To convince yourself that the 2 functions are actually working, compare your two functions with the cmath library (sin and cos)

for angles 0.0 to 1.0, in steps of 0.1 radian.