CSE 20212 Fundamentals of Computing II
Spring 2016

Pre-lab/warm up: due 1/20/16 2pm

Required reading:

Week of 1/11: Chapter 2 and 3 in D&D (basic C++ & classes)
Next week: Chapters 9 and 10 in D&D (advanced class design)

Objectives

1. Refresher of basic C/C++ (Chapters 2-8)
2. Brief re-introduction and implementation of a C++ class (Chapter 3)
3. Introduction to rubrics, grading, and coding requirements for this course

Problems:

1. (6 points) Estimating Pi (Ch 2 in D&D)

   Write a simple but adequately commented C++ program to calculate the value of π from the infinite series:

   $\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - \ldots$

   This C++ program does not require a class or a special function but should correctly estimate π. To do so, it should ask the user for the number of terms *n* of this series (*n* must validated as greater than or equal to 2; see rubric later in this document) and display the estimate of π after each term. For example:

   Please enter the number of terms in the series: 1
   Error! Provided value must be >= 2!
   Please enter the number of terms in the series: 3

   Estimate after 1 term(s): 4
   Estimate after 2 terms(s): 2.666
   Estimate after 3 terms(s): 3.466

2. (15 points) Mortgage calculator (redux) (Chapter 3 in D&D)

   For this problem, we are going to briefly revisit Lab 2 from the fall.

   (a) Develop an adequately commented C++ class called "Mortgage" for easily storing and processing multiple personal mortgages. Use double variables like "Principal," "Rate," and "Payment" to store the remaining principal owed, the interest rate and the desired monthly payment, respectively, in this new class.

(b) Provide a constructor that receives user-provided initial values for a mortgage. As in Fundamentals I, we will require you to check if an numeric input is invalid, specifically if a negative interest rate is provided or a payment would be too small. However, since the objective of this lab is to introduce you to basic class design and error detection, your program can simply display an error message and does not need perform additional corrections (see rubric).

(c) Provide a default constructor, i.e. a constructor with no parameters and "hard coded" default values inside OR provide "fall back" values for the non-default constructor for part b. Both are acceptable for this assignment.

(d) Implement the following member functions:

- void credit (double value): deducts value from the principal remaining
- double getPrincipal(): returns the current principal remaining
- void amortize (): calculates and displays a correct and clean amortization table. You will also be required to calculate the last payment properly (see rubric). This is the most challenging aspect of this problem.

(e) Finally, provide a simple but well commented driver program (main.cpp) that clearly demonstrates your class is working as intended. We'll cover driver programs in the first week of class, and an example is below for your benefit. It is perfectly acceptable to copy and use the below for this assignment.

However, to receive full credit for this assignment, your class definition must be in a .h file, and the member functions for the Mortgage class in a .cpp file.

You must also use a makefile to aid the TAs in grading (and yourself for compiling). We'll use them throughout the semester, so this will be great practice.

Example class driver code is below that you are welcome to submit for 2e:

```
#include<iostream>  //    for cout
#include"mortgage.h"   // declaration of Mortgage class; for 2a
 using namespace std;

int main () {

    Mortgage first (10000, 5.0, 500);   // uses non-default initialization; for 2b
    Mortgage second;            // uses default constructor, mortgage of $100k; for 2c

   // demonstrate functions requested in 2d
    second.credit(10000);  // credit 10,000; careful as this should ideally be 10000.00
    cout << Current balance after crediting second mortgage 10K: "<<
    second.getPrincipal() << endl;
    cout << "Amortization schedule for first mortgage:" << endl;
    first.amortize();
}
```

Could display the following to the user:

Current balance after crediting second mortgage 10K: 90000
Amortization schedule for first mortgage:

Month	Payment	Interest	Balance
1	500.00	41.67	9541.67
2	500.00	39.76	9081.42
3	500.00	37.84	8619.26
…			

Handing it in

Please read and follow the general lab guidelines available on the course website:
http://www.cse.nd.edu/courses/cse20212/www/labs.html

Grading rubric

We'll use rubrics to provide structured, constructive feedback and the prelab is designed in part as a low-cost introduction to grading (and a quick warm up). We may move some points around for the real labs, but we think this is a good start based on last year.

Part 1: /6
+1 Ensures n >= 2
+1 files are commented adequately
+4 accurately estimates Pi based on n displayed terms (see this handout's example)

Part 2: /15
+1 files are commented adequately
+1 uses .h, .cpp and a makefile to compile provided solution
+1 Uses reasonable data member names in C++ class
+2 default and/or non-default constructor works as intended (1 pt each)
+2 member credit and getPrincipal member function work as intended (1 pt each)
+3 Computes amortization table correctly (see above for an example).
+2 Handles bad input values: negative interest and payment too small (1 pt each)
+2 Handles final reduced payment correctly.
+1 Output is formatted cleanly

Report: /4 (choose at least one part above to document)
+1 Explains how the user uses the programs.
+2 Explains how the program works internally.
+1 Explains how the program was verified.

Additional Notes for grading:
- The mortgage result may be off by a few cents. We will not deduct it that happens
- 0.5 can be either 50% or 1/2 of a percent; either way is OK for the assignment as long as the output is correct to the grader

FAQs from previous springs

Question: "I remember that in C we could specify scale/precision of variables in a printf statement using something like "%7.2lf" as a placeholder in the string wherever there would be a double. How can I achieve the same effect in C++ with cout? For example, how would do the following in C++:"

printf("%-3d \t $%7.2lf \t $%7.2lf \t $%10.2lf\n", num1, num2, num3, num4);

Answer: if you always want a precision of two digits, you can apply the setprecision function as follows:

cout<<fixed<<setprecision(2)<<...

NOTE: If you don't set the std::fixed flag to true before using setprecision, then the number supplied to setprecision will apply to the total number of places in the output number. But if you pass std::fixed into cout before using setprecision, it will apply to the number of digits after the decimal.

You can also look into using setw(width_amount) to control the overall width (including spaces) and setiosflags(ios::left) and setiosflags(ios::right) to justify.

If all else fails, you can use printf (and most other C functions) in C++, here by including <stdio.h>

Question: "For the class driver code, are we allowed to use the example given on the previous or are we supposed to make up our own code?"

Answer: Throughout the semester driver programs (main.cpp) will generally be short and more debug focused. Here it doesn't really matter, so use the example or your own code, whichever you prefer.

Question: "Does anyone know if when we have to read the user's input for how many terms to use in the estimate for pi, do we have to make sure the user inputs a digit as opposed to a character or string? If so, it seem simple enough using the built in isdigit function we used in C last semester."

Answer: This is a warm up, so please don't sanity check this input. For future labs, however, isdigit is a very viable solution.