Let's all Repeat Together

Last Time

- We
  - Introduced the **`while`** repetition structure
- Today we will
  - Cover counter and sentinel controlled loops
  - Formatting output

Problems

- Write a program that reads in the salary of 5 employees and calculates the gross pay
  - We know, before the program runs, how many times the loop will iterate
  - Counter-controlled repetition
- Write a program that reads an undetermined number of student grades and calculates the average student grade
  - We don't know, before the program runs, how many times the loop will iterate
  - Sentinel-controlled repetition

Counter-Controlled Repetition

- We know, before we run the program, the number of repetitions that the loop will make
- Also called definite repetition
- 11.1: Write a program that interactively reads in the salary of 5 employees and calculates their gross pay. For each employee the program will read in the number of hours worked and the hourly wage. If an employee works over 40 hours then those hours are considered overtime and the employee will be paid one and a half of their hourly wage for each hour

Sentinel-Controlled Repetition

- We have no idea how many times the loop will need to iterate
- Write a program that reads an undetermined number of student grades and calculates the average student grade
- How will we know when we've read all employee's salaries?
  - i.e. How will we know when to stop looping?

Sentinel-Controlled Repetition

- Use a sentinel value
  - User types employee salaries until all legitimate salaries have been entered
  - User then types in sentinel value to indicate that there are no more legitimate salaries
- Also called indefinite repetition
- Sentinel value must be chosen so that it cannot b confused with legitimate inputs
  - -1 is a good value to use in most cases

Formatting C++ Output

- So far, the only formatting that we have done to our output has been adding spaces and blank lines
- We can also format floating point numbers so that they display a specific number of digits in the fractional part
- You need to include the preprocessor directive **<iomanip>**

Formatting C++ Output

```
cout << "Class average is " << setprecision( 2 )
          << fixed << average << endl;
```

- **setprecision(2)** indicates that there should only be 2 digits in the fractional part of the number
  - The default is 6
- fixed indicates that the number should appear in the fixed point format
  - i.e. no scientific notation

Formatting C++ Output

- Another useful formatting operator is **setw**
- This is also part of the **<iomanip>** library and is in the std namespace
- Format:
  - **cout << setw(12) << temp;**
- This will display the value stored in temp in a space 12 characters wide
- By default the output will be right-justified

Formatting C++ Output

```
int binary = 1010;
int decimal = 10;
cout << setw(7) << "decimal";
cout << setw(10) << "binary";
cout << setw(7) << decimal;
cout << setw(10) << binary;
```


Problem

- Write a program that reads an undetermined number of student grades and calculates the average student grade
- The answer is on the following slides

Solution

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
   int gradeCounter;
   int grade;
   double average;
   double total;

   total = 0;
   gradeCounter = 0;
```

Solution

```
cout << "Enter grade, -1 to end: ";
cin >> grade;

while ( grade != -1 )
{
   total = total + grade;
   gradeCounter = gradeCounter + 1;

   cout << "Enter grade, -1 to end: ";
   cin >> grade;
}

if ( gradeCounter != 0 )
{
   average = total / gradeCounter;

   cout << "Class average is " << average << endl;
}
else
   cout << "No grades were entered" << endl;
```


Summary

- In today's lecture we covered
  - Counter and sentinel-controlled repetitions
  - Formatting output
- Readings
  - P. 83 - 94 counter and sentinel loops
  - P. 93, p. 113 formatting output