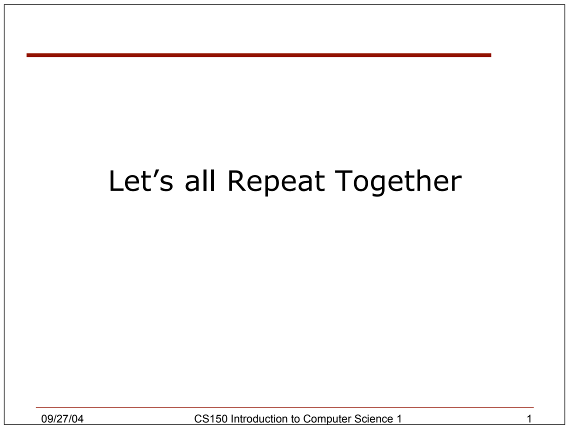
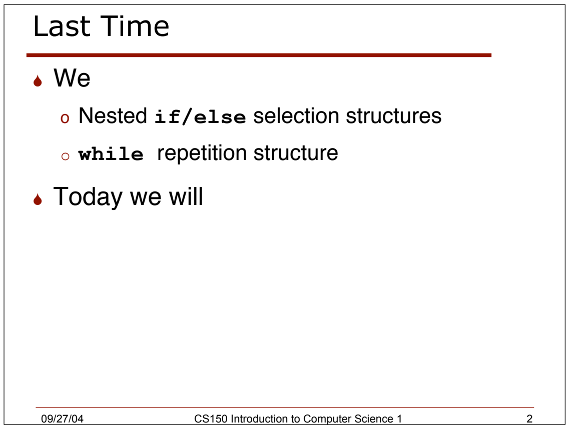
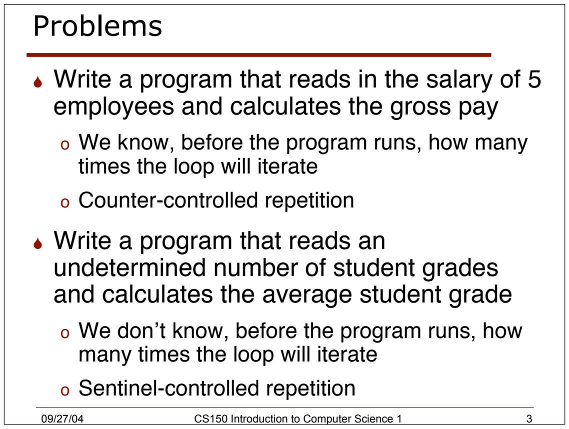


Let’s all Repeat Together



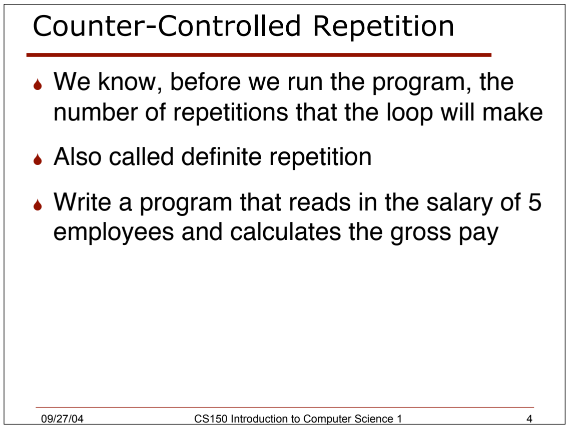
Last Time

- We
  - Nested **`if/else`** selection structures
  - **`while`** repetition structure
- Today we will



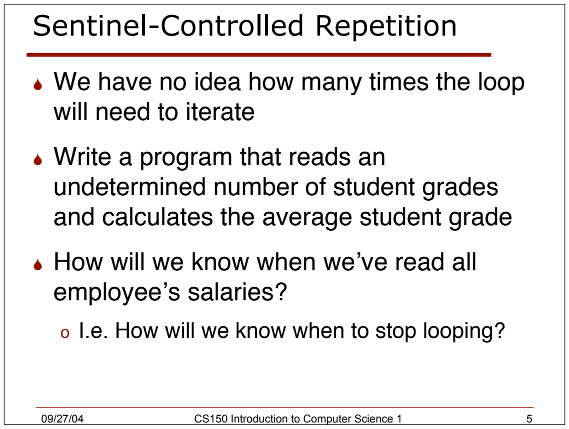
Problems

- 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



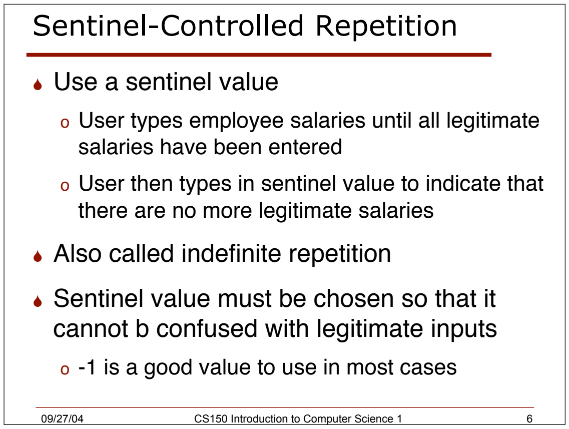
Counter-Controlled Repetition

- We know, before we run the program, the number of repetitions that the loop will make
- Also called definite repetition
- Write a program that reads in the salary of 5 employees and calculates the gross pay



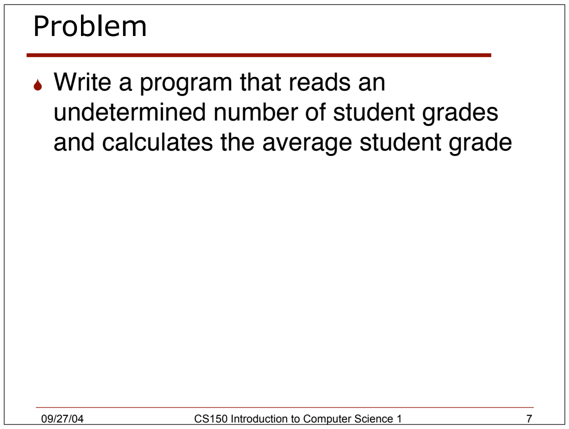
Sentinel-Controlled Repetition

- 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?



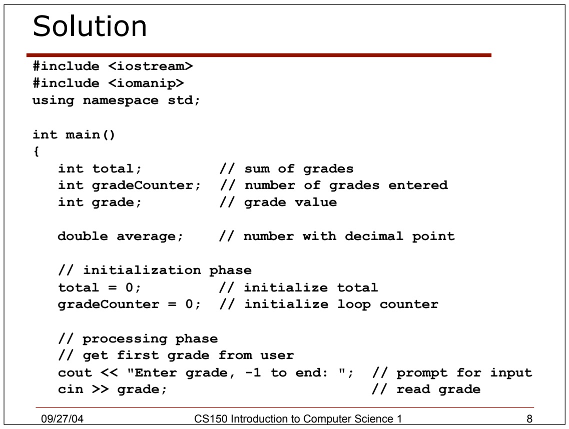
Sentinel-Controlled Repetition

- 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



Problem

- Write a program that reads an undetermined number of student grades and calculates the average student grade



Solution

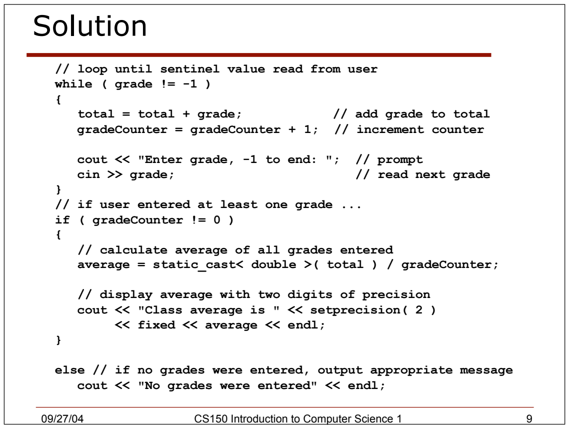
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int total;          // sum of grades
   int gradeCounter;   // number of grades entered
   int grade;          // grade value

   double average;     // number with decimal point

   // initialization phase
   total = 0;          // initialize total
   gradeCounter = 0;   // initialize loop counter

   // processing phase
   // get first grade from user
   cout << "Enter grade, -1 to end: ";  // prompt for input
   cin >> grade;                        // read grade
```



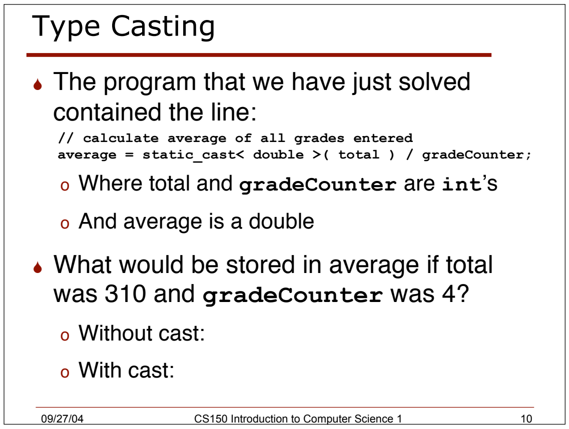
Solution

```
   // loop until sentinel value read from user
   while ( grade != -1 )
   {
      total = total + grade;            // add grade to total
      gradeCounter = gradeCounter + 1;  // increment counter

      cout << "Enter grade, -1 to end: ";  // prompt
      cin >> grade;                        // read next grade
   }
   // if user entered at least one grade ...
   if ( gradeCounter != 0 )
   {
      // calculate average of all grades entered
      average = static_cast< double >( total ) / gradeCounter;

      // display average with two digits of precision
      cout << "Class average is " << setprecision( 2 )
           << fixed << average << endl;
   }

   else // if no grades were entered, output appropriate message
      cout << "No grades were entered" << endl;
```

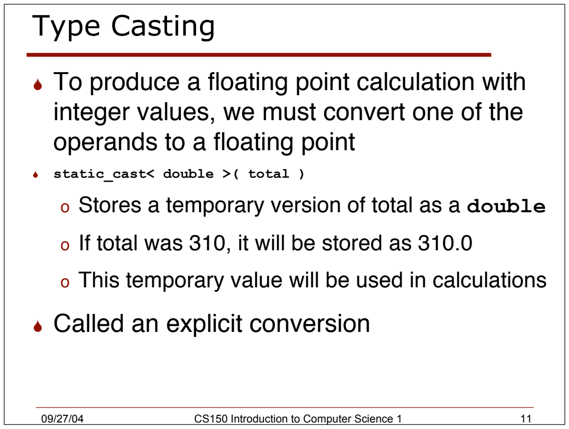


Type Casting

- The program that we have just solved contained the line:

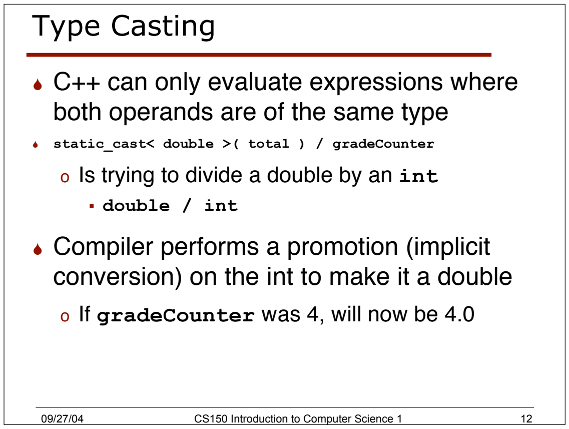
```
// calculate average of all grades entered
average = static_cast< double >( total ) / gradeCounter;
```

  - Where total and **`gradeCounter`** are **`int`**'s
  - And average is a double
- What would be stored in average if total was 310 and **`gradeCounter`** was 4?
  - Without cast:
  - With cast:



Type Casting

- To produce a floating point calculation with integer values, we must convert one of the operands to a floating point
- `static_cast< double >( total )`
  - Stores a temporary version of total as a **`double`**
  - If total was 310, it will be stored as 310.0
  - This temporary value will be used in calculations
- Called an explicit conversion



Type Casting

- C++ can only evaluate expressions where both operands are of the same type
- `static_cast< double >( total ) / gradeCounter`
  - Is trying to divide a double by an **`int`**
    - **`double / int`**
- Compiler performs a promotion (implicit conversion) on the int to make it a double
  - If **`gradeCounter`** was 4, will now be 4.0

Type Casting

- `average = static_cast< double >( total ) / gradeCounter;`
- If total was originally 310 and gradeCounter was 4, compiler will
  - 310.0 / 4.0
  - Results in 77.5
- If average is a double, then 77.5 is stored
- If average is an int then the fractional part will be truncated

static_cast

- It's a unary operator
- The syntax:
  - static_cast<data type>(variable)

Operator Precedence & Associativity

`()`	`L->R`	Parentheses
`static_cast<type>()`	`L->R`	`Unary`
`!, +, -`	`R->L`	Negation, Unary +, -
`*,/,%`	`L->R`	Mult, div, mod
`+, -`	`L->R`	Add, Subtract
`<<, >>`	`L->R`	Insertion/extraction
`<, <=, >, >=`	`L->R`	Relational
`==, !=`	`L->R`	Equality
`&&`	`L->R`	And
`||`	`L->R`	Or
`=`	`R->L`	Assignment

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;
```

A Note on Stepwise Refinement

- P. 87 - 89 in your book describe the process of top-down stepwise refinement
- This is a really useful process for solving a problem
- It describes how to start from the top-most description of the problem and refining it until you have a detailed description of the process
- Be sure to read about it!

Top-Down, Stepwise Refinement

- There is a description of how to solve a complete problem using top-down, stepwise refinement on p. 94 - 98
- The solution to this problem requires that an if selection structure be embedded within a while repetition structure
- You used a similar process when you solved the 3n+1 problem in the lab

Summary

- In today’s lecture we covered
  - Counter and sentinel-controlled repetitions
  - Type casting
  - Formatting output
  - Top-down, stepwise refinement
- Readings
  - P. 83 - 94 counter and sentinel loops
  - P. 92 type casting
  - P. 93, p. 113 formatting output
  - P. 94 - 98 top-down, stepwise refinement