Let's all Repeat Together

Last Time

- We covered
  - Counter and sentinel controlled loops
  - Formatting output
- Today we will
  - Type casting
  - Top-down, stepwise refinement
  - Examine different ways of writing assignments
  - Learn about the increment and decrement operators
  - Start looking at the **`for`** repetition structure

Problem

- 12.1 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;
   int 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 = static_cast< double >( total ) / gradeCounter;

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


Type Casting

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

```
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

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 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

Assignment Operators

- We've seen that C++ provides the ability to abbreviate an assignment operator in which the same variable appears on either side of the operator
- `sum = sum + num;`
- Can be abbreviated to
- `sum += num;`

Assignment Operators

- This abbreviation can be done to the following operators
  - `+ - * / %`
- Examples, where `c = 3, e = 4`
  - `c += 7`
  - `e %= 2`
  - `c *= 3`
  - `e /= 4`
  - `e -= 1`

Increment and Decrement Operators

- ++ is the unary increment operator

  x++;

  is the same as x = x + 1;

- -- is the unary decrement operator

  x--;

  is the same as x = x - 1;

Pre-increment vs. post-increment

Pre	Post
k = --x;	k =x--;
k = ++x;	k = x++;
Increment/ decrement x then assign value of x to k	Assign value of x to k, then increment or decrement x

Example

12.2: What is the output if `i = 2`?

```
cout << "Value of x is" << i;
cout << "Value of i++ is" << i++;
cout << "Value of ++i is" << ++i;
cout << "Value of --i is" << --i;
cout << "Value of i-- is" << i--;
```

Operator Precedence

`()`	`L->R`	Parentheses
`++, --, static_cast<type>()`	`L->R`	`Unary post-op`
`++, --, !, +, -`	`R->L`	Negation, Unary pre-op
`*,/,%`	`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`	`Conditional`
`=, +=, -=, *=, /=, %=`	`R->L`	Assignment

For loops

- 3 main things for loops:
  - Initialization of lcv, testing of lcv, updating lcv
- For loops provide a concise way to do this

```
for (count = 0; count < 5; count++)
  cout << count << endl;
```


General Format

```
for (initialization expression;
  loop repetition condition; update
  expression)
{
  statements;
}
```


Examples

- 12.3: Write a `for` loop that outputs odd numbers less than 10
- 12.4: Write a program that computes the factorial of a number. The factorial of a number is given by the formula
  - N! = N*(N-1)*…*2*1
    - where 0!=1, 1!=1, 2!=2, 3!=6, …

Localized Declarations

```
for (int i = 0; i < n; i++)
  cout << i << endl;
cout << i << endl;
```

`i` is declared ONLY in the loop

Equivalent Logic

12.5: Rewrite the following for loop as a while loop.

```
for (i = 5; i < 10; i+= 2)
  cout << i;
```

12.6: What does this output?

Problem

- 12.7: Write a program that will print the sum of the odd integers between 1 and 50 inclusive. Write one program using a while and the other using a for loop

Problem

- 12.8: Write a program that allows the user to enter an unknown number of integer values one at a time. When the user enters -999, you are to terminate the loop and print the following:
  - The sum of all integers inputted
  - The average of all integers inputted
  - The largest integer of all integers inputted

Summary

- In today's lecture we covered
  - Type casting
  - Top-down, stepwise refinement
  - Abbreviating assignment operators
  - Increment and decrement operators
  - `for` repetition structures
- Readings
  - P. 92 type casting
  - P. 93, p. 113 formatting output
  - P. 94 - 98 top-down, stepwise refinement
  - P. 98 Assignment operators
  - P. 99 - 102 Increment and decrement operators
  - P. 104 - 113 `for` repetition structures