Let's all Repeat Together

Last Time
- We
  - Looked at complex examples that use the if selection structure
  - Covered the if/else selection structure
  - Learned about the ?: conditional operator
- Today we will
  - Start looking at examples of repetition structures

Repetition Structures
- All of the C++ programs that we have seen so far are executed only once before the program terminates
- However, it is often the case that programmers would like to specify that an action continue repeating while a condition is true
- This is achieved by using repetition structures, also called loops

An Example of Repetition
- An example of where we might need to use repetition is if we are calculating the average grade of a class of students
- We would need to continue reading in student grades until we have covered all students
- In pseudocode this might be:
  
  While there are more students in the class
  Ask for student grade

C++ Example
- Write the segment of C++ code that will sum five numbers entered by the user

```cpp
int sum, counter, num;
sum = 0;
counter = 0;
while(counter < 5)
{
  cout << "Enter a number: ";
  cin >> num;
  sum = sum + num;
  counter = counter + 1;
}
cout << "The sum of your numbers is: " << sum;
```

while Repetition Structure
- Initialize LCV
  ```cpp```
  ```
  int sum, counter, num; 
  sum = 0; 
  counter = 1; 
  ```
  ```cpp```
- Loop Control Variable
- Change the value of count
  ```cpp```
  ```
  while( counter <= 5 )
  {
    cout << "Enter a number: ";
    cin >> num;
    sum = sum + num;
    counter = counter + 1;
  }
  ```
  ```cpp```
- Change the value of count
  ```cpp```
  ```
  cout << "The sum of your numbers is: " << sum;
  ```
  ```cpp```

while Repetition Structure
**while loops**

- The syntax for **while** loops is
  
  ```
  while (condition is true)
  statement;
  ```

- The syntax for **while** loops is
  
  ```
  while (condition is true)
  {
  statement1;
  statement2;
  ...
  }
  ```

**Key Ingredients of while loops**

- **Initialize**
  
  MUST initialize loop control variable

- **Test**
  
  The value of the loop control variable is tested during each iteration of loop

- **Update**
  
  Loop control variable is changed during each loop iteration

*If any one of these is missing or incorrect, your loop won’t run properly—not at all, too many/few times or infinitely.*

**Problems**

- **08.1:** Write a while loop that outputs each integer from 1 to 5
- **08.2:** What’s the output for x = 2? 3? 5?

  ```
  cout << "Enter an integer";
  cin >> x;
  product = x;
  count = 0;
  while (count < 4)
  {
  cout << product << endl;
  product *= x;
  count += 1;
  }
  ```

**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

  - 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

- **08.3:** Write a program that interactively reads in the salary of 5 employees and calculates their gross pay. Overtime is to be computed as any hours over 40 gets paid time and one-half.
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 be 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.

The answer is on the following slides.

Solution

```cpp
#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
    double average;    // number with decimal point

    // processing phase
    while ( grade != -1 )
    {
        total = total + grade;            // add grade to total
        gradeCounter = gradeCounter + 1;  // increment counter
        cout << "Enter grade, -1 to end: ";  // prompt for input
        cin >> grade;                        // read grade
        cout << "Enter grade, -1 to end: ";  // prompt
        cin >> grade;                        // read next 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;
    }
    return 0;
}
```

Type Casting

- The program that we have just solved contained the line:
  ```cpp
  average = static_cast<double>( total ) / gradeCounter;
  ```
- Without cast:
  ```cpp
  average = total / gradeCounter;
  ```
- What would be stored in average if total was 310 and gradeCounter was 4?
  - Without cast:
    - 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

- `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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>1</td>
<td>L→R</td>
</tr>
<tr>
<td>static_cast&lt;type&gt;()</td>
<td>2</td>
<td>L→R</td>
</tr>
<tr>
<td>!, +, -</td>
<td>3</td>
<td>R→L</td>
</tr>
<tr>
<td>*, /, %</td>
<td>4</td>
<td>L→R</td>
</tr>
<tr>
<td>+, -</td>
<td>5</td>
<td>L→R</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>6</td>
<td>L→R</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>7</td>
<td>L→R</td>
</tr>
<tr>
<td>==, !=, &amp;&amp;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>9</td>
<td>R→L</td>
</tr>
</tbody>
</table>

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

```
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

Summary

- In today’s lecture we covered
  - Counter and sentinel-controlled repetitions
  - Type casting
  - Formatting output
  - Top-down, stepwise refinement
- Readings
  - P. 81 - 83: while repetition structure
  - P. 83 - 94 counter and sentinel loops
  - P. 92 type casting
  - P. 93, p. 113 formatting output
  - P. 94 - 98 top-down, stepwise refinement