C++ Lecture 4

* Simple Looping, Loop Control, Loop Options, Applications
* CIS 251 • Shelby-Hoover Campus

Looping

* In any of the branching structures, the Boolean (or controlling) expression is only evaluated one time; execution of the program continues with the statements after the branching structure
* Each loop structure also involves a Boolean expression
  + The Boolean expression may be evaluated before the first time the loop’s statements are executed
  + After each repetition of the loop’s statements, the program returns to the Boolean expression to see if the condition remains true
  + Repetition continues until the Boolean expression is false, at which point the program proceeds to the statements after the loop structure

Simple Looping

* The simplest loop form involves evaluating a condition before executing a statement or group of statements; it reevaluates the condition after each execution to determine whether to repeat
  + The statements executed each time the condition is true are known as the **loop body**
  + Each repetition of the loop body is known as an **iteration**
* Syntax: the **while statement**  
    
  while (Boolean\_expression)  
  {  
   statements\_executed\_as\_long\_as\_true;  
  }

Comparing if and while

* Both can have a single or compound statement as the body executing if / while the Boolean expression is true; curly braces are required for compound statements
* Neither has a semicolon at the end of its header
* A while loop never has an else statement (once the while loop terminates, the program proceeds to the code that comes after the loop)
* After a while loop body executes, control of the program returns to the loop’s Boolean expression to determine if another iteration is needed; the Boolean expression of an if statement won’t be evaluated again

Infinite Loops

* The loop body should contain a statement that modifies a variable involved in the Boolean expression
* Failure to modify one of these variables may result in an infinite loop if the Boolean expression never evaluates to false:  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int greetings = 0;  
   while (greetings < 10)  
   cout << "Hello!" << endl;  
    
   return 0;  
  }

Count-Controlled Loops

* In a count-controlled loop, a loop control variable is assigned some starting value before the loop, compared to a stopping value, and modified inside the loop body:  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int count = 1;  
    
   while (count < 6)  
   {  
   cout << "Perform step " << count << endl;  
   count++;  
   }  
    
   cout << "Program complete." << endl;  
    
   return 0;  
  }
* The loop control variable may have any name (count is frequently used but not required)
* The statement to modify the loop control variable may be anything (addition, subtraction, etc.)

Sentinel-Controlled Loops

* In a sentinel-controlled loop, the loop control variable is modified by getting more input from the user (or a file):  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int quantity;  
   double cost;  
    
   cout << "Enter the first order quantity: ";  
   cin >> quantity;  
    
   while (quantity > 0)  
   {  
   cost = quantity \* 1.99;  
   cout << "Order cost: $" << cost << endl;  
    
   cout << "Enter another order (enter 0 or less to exit): ";  
   cin >> quantity;  
   }  
    
   cout << "Thanks for using the order cost calculator." << endl;  
    
   return 0;  
  }

The do-while Loop

* The while loop is known as a **pretest** loop
  + The controlling question is asked before the first iteration
  + If the question is false the first time it is evaluated, there will be zero iterations of the loop body
* A program may require a **posttest** loop that executes one iteration of the loop body before evaluating the question
* In C++, the posttest loop is **do-while**:  
    
  do  
  {  
   statements\_to\_repeat;  
  } while (Boolean\_expression);

do-while Characteristics

* In do-while, the Boolean expression is placed after the loop body and followed with a semicolon (this is the only selection or loop structure that has a semicolon after the closing parenthesis of the Boolean expression)
* A do-while loop can be used when the repetition depends on the user’s response:  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   char ans;  
    
   do  
   {  
   cout << "Hello" << endl;  
   cout << "Want another greeting?" << endl;  
   cout << "(Y = Yes, N = No): ";  
   cin >> ans;  
   } while ((ans == 'Y') || (ans == 'y'));  
    
   cout << "Goodbye" << endl;  
    
   return 0;  
  }

The for Loop

* There are three actions required in every loop
  + Provide a starting value for the loop control variable before evaluation
  + Test / evaluate the loop control variable
  + Update the loop control variable
* A third type of loop, the **for loop**, places all three actions in the header, with semicolons between the actions:  
    
  for (start\_action; test; update)  
  {  
   statements\_to\_repeat;  
  }

for Loops in Execution

* The first portion of the for loop header executes only at the very beginning of the loop execution
* The Boolean expression is evaluated before proceeding to the loop body
* The update step is performed after the loop body executes; then, the Boolean expression is reevaluated
* for loops are especially useful as count-controlled loops (e.g., labeling each line of output with a number):  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int line;  
    
   for (line = 1; line <= 10; line++)  
   cout << line << ". " << endl;  
    
   return 0;  
  }

for Loop Options

* The first and third sections of a for loop header may contain more than one statement, separated by commas instead of semicolons (the semicolons separate the sections):  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int a, b;  
    
   for (a = 1, b = 10; a <= b; a++, b--)  
   cout << a << " times " << b << " equals " << (a \* b) << endl;  
    
   return 0;  
  }
* A program may also omit the statements for these sections from the for loop header if their actions are accomplished elsewhere (but the semicolons that separate the sections must be included):  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int a = 1, b = 10;  
    
   for ( ; a <= b; )  
   {  
   cout << a << " times " << b << " equals " << (a \* b) << endl;  
   a++;  
   b--;  
   }  
    
   return 0;  
  }

More for Loop Details

* The Boolean expression may be composed of several individual Boolean expressions joined by &&, ||
* The loop control variable(s) may be declared in the first section of a for loop header, though these variables only exist as long as the loop executes (they are local to the loop):  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   for (int val = 10; val > 0; val--)  
   cout << val << endl;  
    
   // val no longer exists here  
    
   return 0;  
  }
* Be careful not to write the update action both in the header and in the loop body, or it will execute twice

Loop Commonalities

* In every loop, the program must provide a starting value for the loop control variable before reaching a Boolean expression, test that variable in the Boolean expression, and update it such that the test eventually evaluates to false
* Only the do-while loop has a semicolon after the closing parenthesis surrounding the Boolean expression (while and for do not)
* Only the for loop has semicolons inside the parentheses (two)
* Two statements can be used to alter loop execution (these are generally avoided because they make the loop more difficult to follow):
  + break; // causes the loop to terminate immediately
  + continue; /\* ends the current iteration, proceeds to the next step after the iteration (evaluating the Boolean expression or executing the update statement) \*/

Loop Usage: Statistics

* A program uses a loop to compute a sum or product of a set of input values
  + A variable storing a sum should be initialized to 0
  + A variable storing a product should be initialized to 1
  + Each time the user enters a value, the loop adds the input to the sum or multiplies the product by it
* The average of a set of values is the sum divided by the count

Sum Loop Example

* Calculating the sum (and average) of ten values:  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int sum = 0, value;  
   double average;  
    
   cout << "Enter ten values:" << endl;  
    
   for (int a = 1; a <= 10; a++)  
   {  
   cin >> value;  
   sum += value; // sum = sum + value  
   }  
    
   average = sum / 10.0; // real number division  
    
   cout.setf(ios::fixed); // number formatting  
   cout.setf(ios::showpoint);  
   cout.precision(1);  
    
   cout << "Sum of the values: " << sum << endl;  
   cout << "Average value: " << average << endl;  
    
   return 0;  
  }

Loop Usage: Termination

* A count-controlled loop terminates when a counter variable reaches its limit (a program may ask the user for this limit prior to the loop, in which it is a **list headed by size**)
* If the user does not know how many values are to be entered, another approach is to write an **ask before iterating** loop, in which the program asks at the end of each iteration whether or not another iteration is required
* A variable that indicates that an event has taken place is called a **flag**; the Boolean expression may test the value of the flag to see if it indicates that the loop must terminate (the loop **exits on a flag condition**)

Loop Usage: Nesting

* When a program nests one loop inside another, the inner loop must go through all of its iterations before a single iteration of the outer loop terminates
* Example: a program to calculate a total for seven days in each of four weeks:  
    
  #include <iostream>  
  using namespace std;  
    
  int main()  
  {  
   int grandTotal = 0, weekTotal, dailyAmount, week, day;  
    
   for (week = 1; week <= 4; week++)  
   {  
   weekTotal = 0;  
    
   cout << "Enter amounts for week " << week << endl;  
    
   for (day = 1; day <= 7; day++)  
   {  
   cout << "Day #" << day << ": ";  
   cin >> dailyAmount;  
   weekTotal += dailyAmount;  
   }  
    
   cout << "Total for Week #" << week << ": " << weekTotal  
   << endl << endl; // a blank line between weeks  
    
   grandTotal = grandTotal + weekTotal;  
   }  
    
   cout << "Grand Total: " << grandTotal << endl;  
    
   return 0;  
  }

Loop Errors

* A common mistake that occurs when writing loops is an **off-by-one error**, in which the loop has one too few or one too many iterations
  + Counter starting at 0 vs. 1
  + Comparison being < vs. <=
* If a loop is not functioning properly, it may be helpful to add extra temporary output statements to see the current values of crucial variables during different iterations (remove these from the final version of the program)
* Tracking the changes in the value of a variable is called **tracing** the variable