Designing Loops and General Debug Pre-Defined Functions in C++

CS 16: Solving Problems with Computers I
Lecture #6

Ziad Matni
Dept. of Computer Science, UCSB

Announcements

- **Homework #5 due today**
- **Lab #3 is due on Friday AT NOON!**
- Homework Solutions are now online at:

http://cs.ucsb.edu/~zmatni/cs16/hwSolutions/

MIDTERM IS COMING!

- Material: ***Everything*** we've done**, incl. up to Th. 10/13**
  - Homework, Labs, Lectures, Textbook
- **Tuesday, 10/18** in this classroom
- **Starts at 2:00pm **SHARP****
- **I will chose where you sit!**
- Duration: **1 hour long**

- Closed book: no calculators, no phones, no computers
- Only 1 sheet (single-sided) of written notes
  - Must be no bigger than 8.5" x 11"
  - You have to turn it in with the exam
- **You will write your answers on the exam sheet itself.**

Sample Question
Multiple Choice

Complete the following C++ code that is supposed to print the numbers 2, 3, 4, 5, 6:

```
int c = 0;
while (_____________) {
 cout << c+2 << " ";
 c++; }
```

A. c < 7
B. c > 5
C. (c + 2) <= 6
D. (c + 2) != 7
E. c != 6

In this example, either C or D can be considered correct answers

Sample Question
Coding

Write C++ code that generates a random number between 3 and 7 and then prints out that number in words. For example, the program prints out “five” if the random number was 5.

- *Note:*

When I ask for code, that’s different then when I ask for an entire program.

Sample Question
Coding

Reproduce the output of this C++ code exactly.

```
int prod(1);
for (int m = 1; m <= 6; m++) {
   prod *= m;
   m += 2;
}
cout << “Total product is: ” << prod << endl;
```

Lecture Outline

- Designing Loops in C++
  - And debugging them
- Top-Down Design Concepts
- Pre-Defined Functions in C++

Designing Loops

- What do I need to know?
  - What am I doing inside the loop?
  - What are my initializing statements?
  - What are the conditions for ending the loop?

Sums and Products

- A common task is reading a list of numbers and computing the sum
  - Pseudocode for this task might be:

```
sum = 0;
repeat the following this_many times
        cin >> next;
        sum = sum + next;
end of loop
```

  - How can we best implement this code?
    - Let's look at it as a for-loop on the next slide…

for-loop for a sum

- The pseudocode from the previous slide is implemented as

```
int sum = 0;
for(int count=1; count <= this_many; count++)
   {
        cin >> next;
        sum = sum + next;
   }
```

- Note that “sum” must be initialized prior to the loop body!

Repeat "this many times"

- *ProTip:* Pseudocode containing the line
  repeat the following "this many times"
  is often implemented with a for-loop

- A for-loop is generally the choice when there is **a predetermined number of iterations**

for-loop For a Product

- Forming a **product** is very similar to the sum example seen earlier

```
int product = 1;
for(int count=1; count <= this_many; count++)
{
        cin >> next;
        product = product * next;
}
```

- Note that “product” must be initialized prior to the loop body
- Also, notice that product is initialized to 1, not 0!

Ending a Loop

- The are four common methods to terminate an input loop:
  - List headed by size
    - When we can determine the size of the list beforehand
  - Ask before iterating
    - Ask if the user wants to continue before each iteration
  - List ended with a sentinel value
    - Using a particular value to signal the end of the list
  - Running out of input
    - Using the *eof* function to indicate the end of a file

List Headed By Size

- The for-loops we have seen provide a natural implementation of the list headed by size method of ending a loop
  - Example:

```
int items;
cout << "How many items in the list?";
cin >> items;
for(int count  = 1; count <= items; count++)
   {
   int number;
   cout << "Enter number " << count;
   cin >> number;
   cout << endl;

   // statements to process the number
   }
```

Ask Before Iterating

- A while loop is used here to implement the ask before iterating method to end a loop.

```
sum = 0;
cout << "Are there numbers in the list (Y/N)?";
char ans;
cin >> ans;

while (( ans == 'Y')  || (ans == 'y'))
{
      //statements to read and process the number

   cout << "Are there more numbers(Y/N)? ";
   cin >> ans;
 }
```

List Ended With a Sentinel Value

- A while loop is typically used to end a loop using the list ended with a sentinel value method

```
cout << "Enter a list of nonnegative integers.\n"
     << "Place a negative integer after the list.\n";
sum = 0;
cin >> number;
while (number > 0)
    {
        //statements to read/process number
        cin >> number;
    }
```

  - Notice that the sentinel value is read, but not processed at the end

Running Out of Input

- The while loop is typically used to implement the running out of input method of ending a loop

```
ifstream infile;
infile.open("data.dat");
while (! infile.eof( ) )
    {
    // read and process items from the file
    // File I/O covered in Chapter 6
    }
       infile.close( );
```

General Methods To Control Loops

- Three general methods to control any loop
  - Count controlled loops
  - Ask before iterating
  - Exit on flag condition

Count Controlled Loops

- Count controlled loops are loops that determine the number of iterations *before the loop begins*
  - The list headed by size is an example of a count controlled loop for input

Exit on Flag Condition

- Loops can be ended when a particular flag condition exists
  - ***Flag***: A variable that changes value to indicate that some event has taken place
  - Examples of exit on a flag condition for input
    - List ended with a sentinel value
    - Running out of input

Exit on Flag Example

- Consider this loop to identify a student with a grade of 90 or better and think of how it's logically limited.

```
int n = 1;
grade = compute_grade(n);
   while (grade < 90)
   {
      n++;
      grade = compute_grade(n);
   }
    cout << "Student number " << n
         << " has a score of "  << grade << endl;
```

The Problem

- The loop on the previous slide might not stop at the end of the list of students if ***no*** student has a grade of 90 or higher
  - It is a good idea to use **a second flag** to ensure that there are still students to consider
  - The code on the following slide shows a better solution

The Exit On Flag Solution

- This code solves the problem of having no student grade at 90 or higher, making use of a flag (the variable *n*):

```
int n=1;
grade = compute_grade(n);
while (( grade < 90) && ( n < number_of_students))
{
        // same as before
}
if (grade > 90)
         // same output as before
else
        cout << "No student has a high score.";
```

Nested Loops

- The body of a loop may contain any kind of statement, *including another loop*
  - When loops are nested, all iterations of the inner loop are executed for each iteration of the outer loop
  - *ProTip:* Give serious consideration to making the inner loop a function call to make it easier to read your program

Example of a Nested Loop

```
int grand_total = 0, subtotal = 0, count, next, number_of_reports = 10;
for (count = 1; count <= number_of_reports; count++) {
    cout << "Enter the report of conservationist number "<< count << endl;
    cout << "Enter the number of eggs in each nest. End list with a negative number.\n"
    cin >> next;
    while (next >=0)        {
            subtotal = subtotal + next;
            cin >> next;
    } // end while loop
    cout << "Total egg count for conservationist number " << count << "is " << subtotal <<
    endl;
    grand_total = grand_total + subtotal;
} // end for loop

cout << "Total egg count for all reports = " << grand_total << endl;

return 0;
} // end program
```

Debugging Loops

- Common errors involving loops include
  - *Off-by-one* errors in which the loop executes one too many or one too few times
  - Infinite loops usually result from a mistake in the Boolean expression that controls the loop

Fixing Off By One Errors

- Check your comparison:
  should it be < or <=?
- Check that the initialization uses the correct value
- Does the loop handle the zero iterations case?

Fixing Infinite Loops

- Check the direction of inequalities:
  < or > ?
- Test for < or > rather than equality (==)

More Loop Debugging Tips

- Be sure that the mistake is *really in the loop*

- Trace the variable to observe how the variable changes
  - Tracing a variable is watching its value change during execution.
  - Best way to do this is to insert **cout** statements to have the program show you the variable at every iteration of the loop.

Debugging Example

- The following code is supposed to conclude with the variable product containing the product of the numbers 2 through 5
  - i.e. 2 x 3 x 4 x 5, which, of course, is 120.

- What could go wrong?! ☺

DEMO!
Using variable tracing

```
int next = 2, product = 1;
while (next < 5)
    {
        next++;
        product = product * next;
    }
```

Loop Testing Guidelines

- Every time a program is changed, it should be retested
  - Changing one part may require a change to another

- Every loop should at least be tested using input to cause:
  - Zero iterations of the loop body
  - One iteration of the loop body
  - One less than the maximum number of iterations
  - The maximum number of iterations

Starting Over

- Sometimes it is more efficient to throw out a buggy program and start over!
  - The new program will be easier to read
  - The new program is less likely to be as buggy
  - You may develop a working program faster than if you repair the bad code
    - The lessons learned in the buggy code will help you design a better program faster

Top Down Design Concept

- In general, to write a program
  1. Develop the algorithm that the program will use
  2. Translate the algorithm into the programming language

- Top Down Design (also called *stepwise refinement*)
  1. Break the algorithm into subtasks
  2. Break each subtask into smaller subtasks
  3. Eventually the smaller subtasks are trivial to implement in the programming language

Predefined Functions in C++

Predefined Functions

- C++ comes with "built-in" libraries of predefined functions
- Example: sqrt function (found in the library *cmath*)
  - Computes and returns the square root of a number

```
the_root = sqrt(9.0);
```

  - The number 9 is called *the argument*
  - After calculation, the variable **the_root** will contain 3.0
- Can variable **the_root** be either int or double?

Function Calls

- **sqrt(9.0)** is a function call
  - It invokes a pre-defined function
  - The argument (9), can also be a variable or an expression

- A function call can be used like any expression

```
bonus =  sqrt(sales) / 10;
```

```
cout << "The side of a square with area " << area
     << " is "
     << sqrt(area);
```

Function Call Syntax

Function_Name (*Argument_List*)

- Argument_List is a comma separated list:

  (Argument_1, Argument_2, … , Argument_Last)

- Example:

```
side = sqrt(area);

cout << "2.5 to the power 3.0 is "
     << pow(2.5, 3.0);
```

Function Libraries

- Predefined functions are found in libraries
- The library must be “included” in a program to make the functions available
- An include directive tells the compiler which library header file to include.
- To include the math library containing sqrt(), pow() & others:

```
#include <cmath>
```

- Newer standard libraries, such as **cmath**, also require the directive

```
using namespace std;
```

Other Predefined Functions

- abs(x) --- **int** value = abs(-8);
  - Returns absolute value of argument x
  - Return value is of type **int**
  - Argument is of type int
  - Found in the library **cmath**

- fabs(x) --- **double** value = fabs(-8.0);
  - Also returns absolute value of argument x
  - Return value is of type **double**
  - Argument is of type double
  - Found in the library **cmath**

Random Number Generation

- Not true-random, but pseudo-random numbers.

- First, seed the random number generator only once

```
#include <cstdlib>
#include <ctime>
srand(time(0));
```

  - **time()** is a pre-defined function in the **ctime** library
    - It gives the current system time
  - It's used here because it generates a *distinctive enough seed*, so that **rand()** generates a "good enough" random number.

- Secondly, use the rand() function, which returns a random integer that is greater than or equal to 0 and less than RAND_MAX (a library-dependent value, but is at least 32767)

```
int r = rand();
```

Random Numbers

- Use **%** and **+** to scale to the number range you want

- For example to get a random number bounded from 1 to 6 to simulate rolling a six-sided die:

```
int die = (rand() % 6) + 1;
```

TO DOs

- Readings
  - The rest of Chapter 4, of textbook
- Homework #6
  - Due on Thursday, 10/13 and submit in class
- Lab #3
  - Due Friday, 10/14, at noon

</LECTURE>