Week 6: Intro to Loops

Gaddis: 5.2-6

CS 1428
Fall 2015
Jill Seaman

Control Flow
(order of execution)

• So far, control flow in our programs has included:
  ‣ sequential processing (1st statement, then 2nd statement...)
  ‣ branching (conditionally skip some statements).

• Chapter 5 introduces loops, which allow us to conditionally repeat execution of some statements.
  ‣ while loop
  ‣ do-while loop
  ‣ for loop

5.2 The while loop

• As long as the relational expression is true, repeat the statement

while (expression)
  statement

while syntax and semantics

• The while statement is used to repeat statements:

• How it works:
  ‣ expression is evaluated:
    ‣ If it is true, then statement is executed, then it starts over (and expression is evaluated again).
    ‣ If it is false, then statement is skipped (and the loop is done).
### 5.3 Using `while` for input validation

- Inspect user input values to make sure they are valid.
- If not valid, ask user to re-enter value:

  ```
  int number;
  cout << “Enter a number between 1 and 10: “;
  cin >> number;
  while (number < 1 || number > 10) {
    cout << “Please enter a number between 1 and 10: “;
    cin >> number;
  }
  // Do something with number here
  ```

### 5.4 Counters

- **Counter**: a variable that is incremented (or decremented) each time a loop repeats.
- Used to keep track of the number of iterations (how many times the loop has repeated).
- Must be initialized before entering loop!!!
Counters

- Example (how many times does the user enter an invalid number?):

```cpp
int number;
int count = 0;
cout << "Enter a number between 1 and 10: ";
cin >> number;
while (number < 1 || number > 10) {
    count = count + 1;
    cout << "Please enter a number between 1 and 10: ";
    cin >> number;
}
cout << count << " invalid numbers were entered." << endl;
// Do something with number here
```

- Example, using the counter to control how many times the loop iterates:

```cpp
int num = 1; // counter variable
while (num <= 8) {
    cout << num << "           " << (num * num) << endl;
    num = num + 1; // increment the counter
}
```

- Output:

<table>
<thead>
<tr>
<th>Number</th>
<th>Number Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
</tbody>
</table>

5.5 The do-while loop

- Execute the statement(s), then repeat as long as the relational expression is true.

```
do
    statement
while (expression);
```

- do-while syntax and semantics

  - The do-while loop has the test expression at the end:

  - How it works:
    - statement is executed.
    - expression is evaluated:
    - If it is true, then it starts over (and statement is executed again).
    - If (when) it is false, the loop is done.

  - statement always executes at least once.
do-while example

• Example:

```
int number = 1;
do {
cout << "Student" << number << endl;
    number = number + 1;
} while (number <= 3);
cout << "Done" << endl;
```

• Output

```
Student1
Student2
Student3
Done
```

do-while with menu

```
char choice;
do {
cout << "A: Make a reservation." << endl;
cout << "B: View flight status." << endl;
cout << "C: Check-in for a flight." << endl;
cout << "D: Quit the program." << endl;
cout << "Enter your choice: ";
cin >> choice;
switch (choice) {
case 'A':  // code to make a reservation
    break;
case 'B':  // code to view flight status
    break;
case 'C':  // code to process check-in
    break;
}
} while(choice != 'D');
```

Different ways to control the loop

• Conditional loop: body executes as long as a certain condition is true
  ▸ input validation: loops as long as input is invalid

• Count-controlled loop: body executes a specific number of times using a counter
  ▸ actual count may be a literal, or stored in a variable.

• Count-controlled loop follows a pattern:
  ▸ initialize counter to zero (or other start value).
  ▸ test counter to make sure it is less than count.
  ▸ update counter during each iteration.

5.6 The for loop

• The for statement is used to easily implement a count-controlled loop.

```
for (expr1; expr2; expr3)
    statement
```

• How it works:
  1. expr1 is executed (initialization)
  2. expr2 is evaluated (test)
  3. If it is true, then statement is executed, then expr3 is executed (update), then go to step 2.
  4. If (when) it is false, then statement is skipped (and the loop is done).
The for loop flow chart

for (expr1; expr2; expr3)
  statement

expr1

expr2 → True

statement

expr3

expr2 → False

The for loop and the while loop

• The for statement

for (expr1; expr2; expr3)
  statement

• is equivalent to the following code using a while statement:

expr1;       // initialize
while (expr2) { // test
  statement
  expr3;      // update
}

for loop example

• Example:

```cpp
int number;
for (number = 1; number <= 3; number++)
{
  cout << "Student" << number << endl;
}
cout << "Done" << endl;
```

Output

Student1
Student2
Student3
Done

Counters: redo

• Example, using the counter to control how many times the loop iterates:

```cpp
cout << "Number Number Squared" << endl;
cout << "------ --------------" << endl;
int num = 1;     // counter variable
while (num <= 8) {
  cout << num << " " << (num * num) << endl;
  num = num + 1; // increment the counter
}
cout << "Number Number Squared" << endl;
cout << "------ --------------" << endl;
```

• Rewritten using a for loop:

```cpp
cout << "Number Number Squared" << endl;
cout << "------ --------------" << endl;
int num;
for (num = 1; num <= 8; num++)
{
  cout << num << " " << (num * num) << endl;
}
```
Define variable in init-expr

- You may define the loop counter variable inside the for loop's initialization expression:

```c++
for (int x = 10; x > 0; x=x-2)
    cout << x << endl;
```

- Do NOT try to access x outside the loop (the scope of x is the for loop statement ONLY)
- What is the output of the for loop?

User-controlled count

- You may use a value input by the user to control the number of iterations:

```c++
int maxCount;
cout << “How many squares do you want?” << endl;
cin >> maxCount;
cout << “Number Number Squared” << endl;
cout << “------ ---------------” << endl;
for (int num = 1; num <= maxCount; num++)
    cout << num << “ “ << (num * num) << endl;
```

- How many times does the loop iterate?

The exprs in the for are optional

- You may omit any of the three exprs in the for loop header

```c++
int value, incr;
cout << “Enter the starting value: “;
cin >> value;
for ( ; value <= 100; )
{
    cout << “Please enter the next increment amount: “;
cin >> incr;
    value = value + incr;
    cout << value << endl;
}
```

- Style: use a while loop for something like this.
- When expr2 is missing, it is true by default.