4.5 More for loop examples

Example: Finding the max

Analyzing data is a common programming task. A common data analysis task is to find the maximum value in a list of values. A loop can achieve that task by updating a max-seen-so-far variable on each iteration.

Figure 4.5.1: Finding the max in a list.

```
#include <iostream>
using namespace std;

// Outputs max of list of integers
// First value indicates list size
// Ex: 4 -1 9 0 3  yields 9

int main() {
   int maxSoFar;
   int currValue;
   int numValues;
   int i;

   cin >> numValues;

   for (i = 0; i < numValues; ++i) {
      cin >> currValue;

      if (i == 0) { // First iteration
         maxSoFar = currValue;
      }
      else if (currValue > maxSoFar) {
         maxSoFar = currValue;
      }
   }

   if (numValues > 0) {
      cout << "Max: " << maxSoFar << endl;
   }

   return 0;
}
```

```
4 -1 9 0 3
Max: 9

...

5 -15 -90 -2 -60 -30
Max: -2
```

Feedback?

PARTICIPATION ACTIVITY 4.5.1: Finding the max.

Consider the example above.

1) Before entering the loop, what is the maximum value seen so far from the list of integers?

- 0
- -1
- [✓ No such value]

✓

Correct

Before the loop, no list item has been input yet, so no max value has been seen.

2) The loop's first iteration gets the list's first integer into variable currValue. Is that the maximum value seen so far?

- [✓ Yes]
- No

✓

Correct

The first value is the max value seen so far, as well as being the min value seen so far. The if-else statement's i == 0 branch detects that this iteration is the first, and simply assigns maxSoFar = currValue.

3) For each iteration after the first iteration, the comparison _____ is checked.

- maxSoFar > currValue
- [✓ currValue > maxSoFar]
- currValue == maxSoFar

✓

Correct

After the first iteration, i == 0 will be false, so the else if expression is checked. If the current value is greater than the max seen so far, then that current value is the new max seen so far, so the else if branch executes maxSoFar = currValue.

Feedback?

Beyond iterating N times

The three parts of a for loop may be adjusted to do more than just iterate N times. For example, a for loop can output various sequences. The following outputs multiples of 5 from 10 to 50.

Figure 4.5.2: Outputting multiples of 5 from 10 to 50.

```
#include <iostream>
using namespace std;

// Outputs 10 15 20 25 30 35 40 45 50

int main() {
   int i;

   for (i = 10; i <= 50; i = i + 5) {
      cout << i << " ";
   }

   cout << endl;

   return 0;
}
```

```
10 15 20 25 30 35 40 45 50
```

Feedback?

PARTICIPATION ACTIVITY | 4.5.2: For loops beyond iterating N times.

Type the output of the for loop. Whitespace matters, including after the last item.

Example:

```
for (i = 1; i <= 5; ++i) {
  (Put i to output, followed by a space)
}
```

Outputs:
1 2 3 4 5

1)
```
for (i = 0; i < 5; ++i) {
  (Put i to output, followed by a
space)
}
```

Check Show answer

2)
```
for (i = 1; i <= 5; ++i) {
  (Put i to output, followed by a
space)
}
```

Check Show answer

3)
```
for (i = 0; i < 10; i = i + 2) {
  (Put i to output, followed by a
space)
}
```

Check Show answer

4)
```
for (i = -3; i <= 3; ++i) {
  (Put i to output, followed by a
space)
}
```

Check Show answer

5)
```
for (i = 5; i >= 0; --i) {
  (Put i to output, followed by a
space)
}
```

Check Show answer

6)
```
for (i = 0; i < 5; ++i) {
  (Put 2 * i to output, followed by a
space)
}
```

Check Show answer

Feedback?

Example: Outputting a table of temperatures

Programs are sometimes used to auto-generate data tables. The following program generates a table of Celsius and Fahrenheit temperature values, in increments of 5 C. The for loop counts from -10 to 40 in increments of 5, and names the loop variable currC rather than i to be more descriptive.

Figure 4.5.3: Auto-generate a data table: Celsius to Fahrenheit.

```
#include <iostream>
using namespace std;

int main() {
   int currC;
   double equivalentF;

   for (currC = -10; currC <= 40; currC += 5) {
      equivalentF =  (currC * 9.0 / 5.0) + 32.0;

      cout << currC << " C is ";
      cout << equivalentF << " F";
      cout << endl;
   }

   return 0;
}
```

```
-10 C is 14 F
-5 C is 23 F
0 C is 32 F
5 C is 41 F
10 C is 50 F
15 C is 59 F
20 C is 68 F
25 C is 77 F
30 C is 86 F
35 C is 95 F
40 C is 104 F
```

Feedback?

PARTICIPATION ACTIVITY 4.5.3: For loop generating a table of temperature values.

Consider the example above.

1) What is the loop variable's name?

Check Show answer

2) What are the values of currC for the first four iterations?
Type as: 1 9 2 6

Check Show answer

3) What is the loop expression? (The expression checked for whether to enter the loop body).

Check Show answer

Feedback?

Loop style issues

Starting with 0

Programmers in C, C++, Java, and other languages have generally standardized on looping N times by starting with i = 0 and checking for i < N, rather than by using i = 1 and i <= N. One reason is due to other constructs (arrays / vectors), often used with loops, start with 0. Another is simply that a choice was made.

The ++ operators

The ++ operator can appear as ++i (***prefix form***) or as i++ (***postfix form***). ++i increments i first and then evaluates the result, while i++ evaluates the result first and then increments i. The distinction is relevant in a statement like x = ++i vs. x = i++; if i is 5, the first yields x = 6, the second x = 5.

Some consider ++i safer for beginners in case they type i = ++i, which typically works as expected (whereas i = i++ does not), so this material uses ++i throughout. The -- operator also has prefix and postfix versions. Incidentally, the C++ programming language gets its name from the ++ operator, suggesting C++ is an increment or improvement over its C language predecessor.

In-loop declaration of i

Variables can be declared throughout code, so many programmers use: `for (int i = 0; i < N; ++i)`. But, the teaching experience of this material's authors suggests such declarations may confuse learners who may declare variables within loops, repeatedly re-declaring variables, etc. This material avoids the in-loop declaration approach. The authors hope to make the learning less error-prone, and have confidence that programmers can easily pick up on the common in-loop declaration approach later.

PARTICIPATION ACTIVITY 4.5.4: Miscellaneous for loop and ++ topics.

1) Do these loops iterate the same number of times?

```
for (i = 0; i < 5; ++i) {
   ...
}

for (i = 1; i <= 5; ++i) {
   ...
}
```

○ Yes

○ No

2) Does this for loop iterate 5 times?

```
for (i = 0; i < 5; i++) {
   ...
}
```

○ Yes

○ No

3) Is the following valid code?

```
for (int i = 0; i < 5; i++) {
   ...
}
```

○ Yes

○ No

Feedback?

Common errors / good practice

A common error is to also have a ++i; statement in the loop body, causing the loop variable to be updated twice per iteration.

Figure 4.5.4: Common error: loop variable updated twice.

```
// Loop variable updated twice per iteration
for (i = 0; i < 5; ++i) {
   // Loop body
   ++i; // Oops
}
```

Feedback?

While the initialization and update parts of a for loop can include multiple statements separated by a comma, good practice is to use a single statement for each part. Good practice also is to use a for loop's parts to count the necessary loop iterations, with nothing added or omitted. The following loop examples should be avoided, if possible.

Figure 4.5.5: Avoid these for loop variations.

```
// initialExpression not related to counting iterations; move r = rand() before loop
for (i = 0, r = rand(); i < 5; ++i) {
   // Loop body
}

// updateExpression not related to counting iterations; move r = r + 2 into loop body
for (i = 0; i < 5; ++i, r = r + 2) {
   // Loop body
}
```

Feedback?

PARTICIPATION ACTIVITY 4.5.5: For loop: Common errors / good practice.

1) Putting ++i at the end of a for loop body, in addition to in the updateExpression part, yields a syntax error.

- ○ True
- ○ False

2) The above two for loop variations each yield a syntax error.

- ○ True
- ○ False

Feedback?

CHALLENGE ACTIVITY 4.5.1: For loops.

✓ 1 ✓ 2 ✓ 3 ✓ 4 ✓ 5

Jump to level 1

Write a for loop that prints from initialNumber to lastNumber. Ex: initialNumber = -3 and lastNumber = 1 outputs:

```
-3 -2 -1 0 1
```

```
#include <iostream>
using namespace std;

int main() {
   int initialNumber;
   int lastNumber;
   int i;

   cin >> initialNumber;
   cin >> lastNumber;

   for(int i = initialNumber; i <=lastNumber; i++/* Your code goes here */) {
      cout << i << " ";
   }

   return 0;
```

1 2 3 4 5

Check Next **Done**. Click any level to practice more. Completion is preserved.

✓ The loop starts with the value in initialNumber due to i = initialNumber. The loop ends with the value in lastNumber due to i <= lastNumber. If initialNumber is assigned a value greater than lastNumber the loop condition is immediately false and the loop is never entered.

✓ 1: Compare output

Input
```
-3 1
```
Your output
```
-3 -2 -1 0 1
```

✓ 2: Compare output

Input
```
-8 4
```
Your output
```
-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4
```

✓ 3: Compare output

Input
```
6 2
```
Your output *Your program produced no output*

✓ 4: Compare output

Input
```
-7 -3
```
Your output
```
-7 -6 -5 -4 -3
```

Feedback?