4.10 Variable name scope

Scope of names

A declared name is only valid within a region of code known as the name's ***scope***. Ex: A variable userNum declared in main() is only valid within main(), from the declaration to main()'s end.

Most of this material declares variables at the top of main() (and if the reader has studied functions, at the top of other functions). However, a variable may be declared within other blocks too. A ***block*** is a brace-enclosed {...} sequence of statements, such as found with an if-else, for loop, or while loop. A variable name's scope extends from the declaration to the closing brace }.

PARTICIPATION ACTIVITY 4.10.1: Variable name scope extend to the end of the declaration's block.

1 2 3 2x speed

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

int main() {

   // int val1  = userNum;     // ERROR
   int userNum = 2;            // Name valid to main's "}"
   int newNum  = userNum + 1;
   int i;

   for (i = 0; i < newNum; ++i) {
      int valSquared;          // Name valid to for's "}"
      valSquared = userNum * userNum;
      cout << i << " squared: " << valSquared << endl;
   }

   // cout << "Last value: " << valSquared << endl; // ERROR

   return 0;
}
```

userNum's scope is from the declaration to main's closing brace. Using userNum before the declaration would yield an "Undeclared name" compiler error.

Feedback?

PARTICIPATION ACTIVITY 4.10.2: Variable name scope.

Refer to the animation above.

1) userNum can be used in newNum's declaration.
   - (•) True
   - ( ) False

Correct

userNum is declared above newNum's declaration, so can be used.

2) If uncommented, userNum can be used in val1's declaration.
   - ( ) True
   - (•) False

Correct

val1 is declared above userNum, so userNum is not known yet. A compiler would generate an error like "Unknown name: userNum".

3) userNum can be used within the for loop's block of statements.
   - (•) True
   - ( ) False

Correct

userNum's scope extends into blocks.

4) valSquared can be used within the for loop's block.
   - (•) True
   - ( ) False

Correct

valSquared's scope extends from the declaration to the for loop's closing brace.

5) valSquared can be used in the for loop's loop variable update, such as replacing ++i by i = i + valSquared.
   - ( ) True
   - (•) False

Correct

valSquared's scope starts at valSquared's declaration, which comes after the for loop's loop variable update.

6) valSquared can be used just before main's return statement
   - ( ) True
   - (•) False

Correct

That return statement is outside the for loop's closing brace, but valSquared's scope ends at the for loop's closing brace.

Feedback?

For loop index

Programmers commonly declare a for loop's index variable in the for loop's initialization statement. That index variable's scope covers the other parts of the for loop, up to the for loop's closing brace. The reason is clear from the for loop's equivalent while loop code shown below, noting the braces around the equivalent code.

Table 4.10.1: Index variable declared in a for loop's initialization statement.

for loop	Equivalent while loop

`for (int i = 0; i < 5; ++i) {`
`  x = x + i;`
`}`
 	
`// x = x + i;  // ERROR`
	`{`
	`  int i = 0;`
	`  while (i < 5) {`
	`  x = x + i;`
	`  ++i;`
	`  }`
	`}`
	`// x = x + i;  // ERROR`

Feedback?

The approach of declaring a for loop's index variable in the for loop's initialization statement makes clear that the variable's sole purpose is to serve as that loop's index.

This material avoids declaring index variables in for loops

This material's authors have found that declaring all variables first, then using those variables in the rest of the code, can simplify learning for students. Thus, this material avoids late declarations of variables, including declaring index variables in for loops. With that said, declaring index variables in for loops is extremely common and considered good practice by many programmers, and thus is something to consider, if one can do so without confusion.

PARTICIPATION ACTIVITY 4.10.3: For loop index declared in loop's initialization statement.

Given the following for loop, use the above equivalent while-loop to determine whether index i's scope includes the indicated region.

```
(a)
for (int i = 0; (b); (c) ) {
   (d)
}
(e)
```

1) (a)

- Yes
- (•) No

Correct

i is not valid before i's declaration.

2) (b)

- (•) Yes
- No

Correct

The loop's condition statement comes after the initialization statement where i is declared, so i can be used. An example use is: `i < 99`.

3) (c)

- (•) Yes
- No

Correct

The loop's update statement comes after the initialization statement where i is declared, and comes before the while loop's closing brace, so i is valid there. An example use is: `++i`.

4) (d)

- (•) Yes
- No

Correct

The for loop's statements are within the while loop's braces, so i is valid there. An example use is: `x = x + i;`

5) (e)

- Yes
- (•) No

Correct

The while loop is surrounded by braces, forming a block. Variable i's scope ends at the closing brace of that block. Part (e) is outside that block, so i is not valid there.

6) Suppose the above for loop is followed by a second for loop also with int i = 0 in the initialization statement. Will the compiler generate an error due to two declarations of i?

- Yes
- (•) No

Correct

The while-loop equivalent is surrounded by braces. Thus, the first loop's scope of i ends at the closing surround brace. i's declaration in the first loop is thus completely hidden outside those braces. The second loop can thus declare i again.

Feedback?

Common error

A common error is to declare a variable inside a loop whose value should persist across iterations. Below, the programmer expects the output to be 0, 1 (0+1), 3 (0+1+2), 6 (0+1+2+3), and 10 (0+1+2+3+4), but instead the output is just 0, 1, 2, 3, 4.

Figure 4.10.1: Common error: A variable declared within a loop block is (unexpectedly) re-initialized every iteration.

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

int main() {
   int i = 0;

   while (i < 5) {
      int tmpSum = 0;
      tmpSum = tmpSum + i; // Logic error: Sum is always just i
      cout << "tmpSum: " << tmpSum << endl;
      i = i + 1;
   }

   return 0;
}
```

```
tmpSum: 0
tmpSum: 1
tmpSum: 2
tmpSum: 3
tmpSum: 4
```

Feedback?

PARTICIPATION ACTIVITY 4.10.4: Common error of a variable declared within a loop block being reinitialized every iteration.

Given the following code, indicate j's value at the specified point.

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

   j = j * i;
}
```

1) At the end of iteration i = 0.

Check Show answer

Answer

0

j is initialized to 0 at the iteration's start. i is 0. So j = 0 * 0 = 0.

2) At the end of iteration i = 1.

Check Show answer

Answer

0

At the start of iteration i = 1, j is declared and initialized to 0. Thus, j = 0 * 1 = 0.

3) At the end of iteration i = 2.

Check Show answer

Answer

0

At the start of iteration i = 2, j is declared and initialized to 0. Thus, j = 0 * 2 = 0. The programmer probably should have declared j before the loop, not within the loop.

4) After the loop terminates, can j be output? Type yes or no.

Check Show answer

Answer

No

In addition to the common error of re-initializing a variable on every iteration, new programmers sometimes try to access such a variable outside a loop. New programmers may be wise to declare all variables at the top of main() (or other function), with the possible exception of the loop index variable, for convenience.

Feedback?