

12.1 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

12.1.1: Variable name scope extend to the end of the declaration's block.



Animation captions:

1. `userNum`'s scope is from the declaration to `main`'s closing brace. Using `userNum` before the declaration would yield an "Undeclared name" compiler error.
2. A declaration can appear within any block (statements in braces `{...}`). `valSquared` is valid from the declaration to the closing brace.
3. `valSquared` is not valid outside that block.

PARTICIPATION ACTIVITY

12.1.2: Variable name scope.



Refer to the animation above.

- 1) `userNum` can be used in `newNum`'s declaration.

- ☐ True
☐ False

- 2) `userNum` can be used in `val1`'s declaration.

- ☐ True
☐ False

- 3) `userNum` can be used within the for loop's block of statements.



- ☐ True
- ☐ False

4) valSquared can be used within the for loop's block.

- ☐ True
- ☐ False

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

- ☐ True
- ☐ False

6) valSquared can be used just before main's return statement.

- ☐ True
- ☐ False

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

for loop	Equivalent while loop
<pre>for (int i = 0; i < 5; ++i) { x = x + i; } // x = x + i; // ERROR</pre>	<pre>{ int i = 0; while (i < 5) { x = x + i; ++i; } } // x = x + i; // ERROR</pre>

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.

12.1.3: For loop index declared in loop's initialization statement.

ACTIVITY

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

2) (b)

☐ Yes☐ No

3) (c)

☐ Yes☐ No

4) (d)

☐ Yes☐ No

5) (e)

☐ Yes☐ No

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

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

```
#include <stdio.h>

int main(void) {
    int i = 0;
    while (i < 5) {
        int tmpSum = 0;
        tmpSum = tmpSum + i; // Logic error: Sum is always just i
        printf("tmpSum: %d\n", tmpSum);
        i = i + 1;
    }

    return 0;
}
```

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

PARTICIPATION ACTIVITY

12.1.4: 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

2) At the end of iteration i = 1.

Check

Show answer

3) At the end of iteration i = 2.

Check

Show answer

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

[Check](#)[Show answer](#)

Ahram Kim

AhramKim@u.boisestate.edu

12.2 zyBooks built-in programming window

Aug. 27th, 2017 18:15

PARTICIPATION ACTIVITY

12.2.1: Programming window.

[Load default template...](#)

```
1 /* Enter your program here */  
2 |
```

Pre-enter any input for program, the

Run

Ahram Kim

AhramKim@u.boisestate.edu

BOISESTATECS253Fall2017

Aug. 27th, 2017 18:15