Block Scope

After viewing the lecture about stack memory, you should be already have an intuitive idea about how **block scope** works. The idea is that certain blocks of code, signified by **{ }**, create an inner stack on top of the previous stack, which can hide the pre-existing values. The lifetime of stack variables inside a block is called the variable's **scope**. Variables created on the stack inside the inner block are only in existence for the duration of the block. When the block ends, the inner stack is removed, and the pre-existing values in the outer scope are available again (because they are still in scope!).

Here is an example:

```
#include <iostream>
int main() {

  // You can actually make an inner scope block anywhere in a function.
  // Let's do it to show how scope works.

  // In the initial, outer scope:
  int x = 2;
  std::cout << "Outer scope value of x (should be 2): " << x << std::endl;

  // Create an inner scope and make a new variable with the name "x".
  // This is not an error! We can redeclare x because of the inner scope.
  // Also, make an extra variable named "y".
  {
    int x = 3;
    int y = 4;
    std::cout << "Inner scope vaue of x (should be 3): " << x << std::endl;
    std::cout << "Inner scope vaue of y (should be 4): " << y << std::endl;
  }

  // Now that the inner block has closed, the inner x and y are gone.
  // The original x variable is still on the stack, and it has its old value:
  std::cout << "Outer scope value of x (should be 2): " << x << std::endl;

  // We can't refer to y here, because it doesn't exist in this scope at all!
  // If you un-comment this line, there will be a compile error.
  // std::cout << "This line causes an error because y doesn't exist: " << y <<

  return 0;
}
```

Run

Reset

Some keywords like **if** can have a block in { } afterwards, which does create an inner block scope for the duration of the conditional block:

```cpp
#include <iostream>
int main() {
  int x = 2;
  std::cout << "Outer scope value of x (should be 2): " << x << std::endl;

  if (true) {
    int x = 3;
    std::cout << "Inner scope vaue of x (should be 3): " << x << std::endl;
  }

  std::cout << "Outer scope value of x (should be 2): " << x << std::endl;

  return 0;
}
```

Run

Reset

Loops

There are several kinds of loops in C++ that allow you to process data iteratively. We'll just show you a few types here.

for loops

The **for** loop is a common loop type that lets you specify an iteration variable, a range for that variable, and an increment instruction. The syntax is:

for (***declaration*** **;** ***condition*** **;** ***increment operation*** **) {** ***loop body*** **}**

Be careful about whether you are redeclaring the iteration variable at block scope or not! Here's an example:

```
#include <iostream>
int main() {

  // outer scope version of "x" to show the for-loop block scoping
  int x = -1;

  std::cout << "[Outside the loop] x is now (should be -1): " << x << std::endl;

  // The for loop lets us declare a variable in the first part of the
  // loop statement, which will belong to the inner block scope:
  for (int x = 0; x <= 2; x++) {
    std::cout << "[Inside the loop] x is now: " << x << std::endl;
  }

  // The outer scope x is still -1
  std::cout << "[Outside the loop] x is now (should be -1): " << x << std::endl;

  // This version doesn't redeclare x, so it just inherits access to the
  // same x variable from the outer scope. This modifies the outer x directly!
  for (x = 0; x <= 2; x++) {
    std::cout << "[Inside the loop] x is now: " << x << std::endl;
  }

  // In the last iteration where the condition x<=2 was true, x had the value 2.
  // After that iteration, x was incremented one more time and became 3.
  // Then the condition was false and the loop body did not execute.
  // Afterwards, the outer scope x is still 3 because the loop modified it.
  std::cout << "[Outside the loop] x is now (should be 3): " << x << std::endl;

  return 0;
}
```

Run

Reset

while loops

Another loop is the simple **while** loop, which has this syntax:

while (*condition*) { *loop body* }

```
#include <iostream>
int main() {
  int x = 0;
  std::cout << "This should show 0, 1, 2, 3:" << std::endl;
  while (x <= 3) {
    std::cout << "x is now: " << x << std::endl;
    x++; // increment x by 1
  }
  return 0;
}
```

Run

Reset

Preview: Modern "range-based" for loops

You might see another very common type of **for** loop in recent C++ code, that has this syntax:

for (***temporary variable declaration*** **:** ***container*** **) {** ***loop body*** **}**

We'll wait to talk more about that in a later reading lesson.