4.1 — Blocks (compound statements)

BY ALEX ON JUNE 18TH, 2007 | LAST MODIFIED BY ALEX ON SEPTEMBER 9TH, 2017

Blocks (compound statements)

A **block** of statements, also called a **compound statement**, is a group of statements that is treated by the compiler as if it were a single statement. Blocks begin with a { symbol, end with a } symbol, and the statements to be executed are placed in between. Blocks can be used any place where a single statement is allowed. No semicolon is needed at the end of a block.

You have already seen an example of blocks when writing functions:

```
int add(int x, int y)
2
     { // start a block
3
       return x + y;
4
     } // end a block
5
6
    int main()
7
    { // start a block
8
9
    // multiple statements
10
         int value(0);
     add(3, 4);
11
12
13
     return 0;
14
15 } // end a block (no semicolon)
```

Blocks can be nested inside of other blocks. As you have seen, the *if statement* executes a single statement if the condition is true. However, because blocks can be used anywhere a single statement can, we can instead use a nested block of statements to make the *if statement* execute multiple statements if the condition is true!

```
1
     #include <iostream>
2
3
     int main()
4
5
         std::cout << "Enter an integer: ";</pre>
6
         int value;
7
         std::cin >> value;
8
9
         if (value >= 0)
10
         { // start of nested block
              std::cout << value << " is a positive integer (or zero)" << std::endl;</pre>
11
             std::cout << "Double this number is " << value * 2 << std::endl;</pre>
12
         } // end of nested block
13
14
         else
15
          { // start of another nested block
16
              std::cout << value << " is a negative integer" << std::endl;</pre>
17
              std::cout << "The positive of this number is " << -value << std::endl;</pre>
         } // end of another nested block
18
19
20
        return 0;
21
    }
```

If the users enters the number 3, this program prints:

```
Enter an integer: 3
3 is a positive integer (or zero)
Double this number is 6
```

If the user enters the number -4, this program prints:

```
Enter an integer: -4
-4 is a negative integer
The positive of this number is 4
```

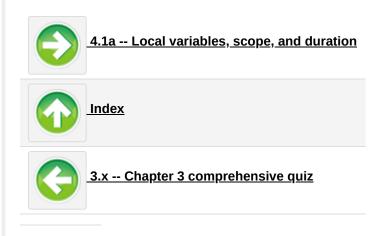
It is even possible to put blocks inside of blocks inside of blocks:

```
1
      int main()
2
3
          std::cout << "Enter an integer: ";</pre>
4
          int value;
5
          std::cin >> value;
6
7
          if (value > 0)
8
9
              if ((value \% 2) == 0)
11
                   std::cout << value << " is positive and even" << std::endl;</pre>
               }
12
13
              else
14
               {
15
                   std::cout << value << " is positive and odd" << std::endl;</pre>
16
17
          }
18
19
        return 0;
20
```

There is no practical limit to how many nested blocks you can have. However, it is generally a good idea to try to keep the number of nested blocks to at most 3 (maybe 4) blocks deep. If your function has a need for more, it's probably time to break your function into multiple smaller functions!

Summary

Blocks allow multiple statements to be used wherever a single statement can normally be used. They are extremely useful when you need a set of statements to execute together.



Share this:



38 comments to 4.1 — Blocks (compound statements)

Aditi <u>June 22, 2018 at 8:16 pm · Reply</u>