6.1 — Compound statements (blocks)

BY ALEX ON JUNE 18TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 4TH, 2020

A **compound statement** (also called a **block**, or **block statement**) is a group of *zero or more statements* that is treated by the compiler as if it were a single statement.

Blocks begin with a { symbol, end with a } symbol, with the statements to be executed being placed in between. Blocks can be used anywhere 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, as the function body is a block:

```
int add(int x, int y)
{ // start block
    return x + y;
} // end block (no semicolon)

int main()
{ // start block

    // multiple statements
    int value {}; // this is initialization, not a block
    add(3, 4);

    return 0;

} // end block (no semicolon)
```

Blocks inside other blocks

Although functions can't be nested inside other functions, blocks *can be* nested inside other blocks:

```
int add(int x, int y)
{ // block
    return x + y;
} // end block

int main()
{ // outer block

    // multiple statements
    int value {};

    { // inner/nested block
        add(3, 4);
    } // end inner/nested block

    return 0;

} // end outer block
```

When blocks are nested, the enclosing block is typically called the **outer block** and the enclosed block is called the **inner block** or **nested block**.

Using blocks to execute multiple statements conditionally

One of the most common use cases for blocks is in conjunction with `if statements`. By default, an `if statement` executes a single statement if the condition evaluates to `true`. However, we can replace this single statement with a block of statements if we want multiple statements to execute when the condition evaluates to `true`.

For example:

```
#include <iostream>

int main()
{ // start of outer block
    std::cout << "Enter an integer: ";
    int value {};
    std::cin >> value;

    if (value >= 0)
    { // start of nested block
        std::cout << value << " is a positive integer (or zero)\n";
        std::cout << "Double this number is " << value * 2 << '\n';
    } // end of nested block
    else
    { // start of another nested block
        std::cout << value << " is a negative integer\n";
        std::cout << "The positive of this number is " << -value << '\n';
    } // end of another nested block

    return 0;
} // end of outer block
```

If the user 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
```

We'll talk more about `if statements`, including the use of blocks, in lesson **5.2 -- If statements**.

Block nesting levels

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

```
int main()
{ // nesting level 1
    std::cout << "Enter an integer: ";
    int value {};
    std::cin >> value;

    if (value >  0)
    { // nesting level 2
        if ((value % 2) == 0)
        { // nesting level 3
            std::cout << value << " is positive and even\n";
```

```
        }
        else
        { // also nesting level 3
            std::cout << value << " is positive and odd\n";
        }
    }

    return 0;
}
```

The **nesting level** (also called the **nesting depth**) of a function is the maximum number of blocks you can be inside at any point in the function (including the outer block). In the above function, there are 4 blocks, but the nesting level is 3 since you can never be inside more than 3 blocks at any point.

It's a good idea to keep your nesting level to 3 or less. Just as overly-long functions are good candidates for refactoring (breaking into smaller functions), overly-nested functions are also good candidates for refactoring (with the most-nested blocks becoming separate functions).

Best practice

Keep the nesting level of your functions to 3 or less. If your function has a need for more, consider refactoring.

 **6.2 -- User-defined namespaces**

 **Index**

 **O.4 -- Converting between binary and decimal**

C++ TUTORIAL | PRINT THIS POST

50 comments to 6.1 — Compound statements (blocks)

Benur21
July 29, 2019 at 7:11 am · Reply

Can I use compound statements alone, outside any if, function, etc?
Eg:

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int value;
    std::cin >> value;
    {
      std::cout << value << " is a positive integer (or zero)" << std::endl;
      std::cout << "Double this number is " << value * 2 << std::endl;
    }
    return 0;
}
```

nascardriver
July 29, 2019 at 8:14 am · Reply

Yes.
If you think you need to do this to make your code more tidy, use a function instead.

Charan
December 14, 2019 at 12:07 am · Reply

But doesn't it raise the concerns about scope of a variable as shown in one of the comments.(Copying the code from @Alireza's comment)

Eg:

```
int main()
{
   int a = 0;
   std::cout << a ;

   {
      int a = 1;
      std::cout << a;

      {
         int a = 2;
         std::cout << a;
      }
   }
   std::cout << a;
   return 0;
}
```

nascardriver
December 14, 2019 at 3:53 am · Reply

I don't understand what you're getting at. Alireza's code can be re-written with functions and without shadowing (see Alex's answer to Alireza's comment).

Aakash
February 22, 2019 at 8:56 am · Reply

"It is even possible to put blocks inside of blocks inside of blocks:" can be changed to
"It is even possible to put blocks inside block:"

Alex
February 22, 2019 at 4:42 pm · Reply

The subsequent example shows a 3-level nesting, which is actually a block inside a block inside a block.

Alireza
February 18, 2019 at 9:12 am · Reply

Hi,

In the following program:

```
#include <iostream>

int main()
{
    int a = 0;
    std::cout << a ;

    {
        int a = 1;
        std::cout << a;

        {
             int a = 2;
             std::cout << a;
        }
    }
    std::cout << a;
    return 0;
}
```

output:
0120

@a doesn't work similarly @a at the 2nd block and 3rd block. When should we use blocks like this way ?

Alex
February 19, 2019 at 7:56 pm · Reply

the 2nd and 3rd nested blocks shadow the outer block a, so your code actually runs identically to this program:

```
#include <iostream>

```

```
3  int main()
4  {
5      int a = 0;
6      std::cout << a ;
7
8      {
9          int a1 = 1;
10         std::cout << a1;
11
12         {
13             int a2 = 2;
14             std::cout << a2;
15         }
16     }
17     std::cout << a;
18     return 0;
19 }
```

> When should we use blocks like this way ?

Never. :) Just because you can do something doesn't mean you should.

Rehan
January 14, 2019 at 11:30 am · Reply

Nice work

Aditi
June 22, 2018 at 8:16 pm · Reply

Hi!

In the third program , what result would we get if we enter a negative integer ? Who don’t we have an else statement for negative integers ? Also , isn’t it necessary that every if statement needs to be followed by an else statement ?

Aditi
June 23, 2018 at 12:38 am · Reply

I think the reason is because as we have taken data type as int, which is already unsigned and accepts only positive integral values. But if we had already specified the data type earlier , why did we need an If statement for accepting only positive numbers? Even if we did not have an if statement for accepting positive integral values , and the user entered a negative number , wouldn’t we get an error in either case ?

nascardriver
June 23, 2018 at 5:27 am · Reply

Hi Aditi!

> what result would we get if we enter a negative integer
No output

> isn’t it necessary that every if statement needs to be followed by an else statement
No, you don't need an else-statement

> we have taken data type as int, which is already unsigned
An int is signed by default

Run the program yourself, play around with some numbers and place breakpoints to get a better understanding of what's happening.

Aditi
June 23, 2018 at 6:08 am · Reply

@nascardriver okay thank you !

Ali Dahud
February 19, 2018 at 7:23 am · Reply

Hi Alex!

could you go through my code and correct the mistakes i made?
oh and please explain this piece:

tempNum = larger;
larger = smaller;
smaller = tempNum;

```
1  // switchintegers.cpp : Defines the entrlarger point for the console application.
2  //
3
4  #include "stdafsmaller.h"
5  #include <iostream>
6
7  int userInput1()
8  {
9      int smaller;  //smaller gets created here
10     std::cout << "Enter an integer: ";
11     std::cin >> smaller;
12     return smaller;
13 }//smaller gets destrolargered here
14
15 int userInput2()
16 {
17     int larger; //larger gets created here
18     std::cout << "Enter a bigger integer: ";
19     std::cin >> larger;
20     return larger;
21 }//larger gets destrolargered here
22
23
24 int ifElse(int smaller, int larger)
25 {
26     int tempNum;
27
28     if (smaller > larger)
29     {
30
31         tempNum = larger;
32         larger = smaller;
33         smaller = tempNum;
34
35         std::cout << "Swapping the values\n";
36         std::cout << "The smaller value is: " << smaller << "\n";
37         std::cout << "The bigger value is: " << larger << "\n";
```

```
    }
    else
    {
        std::cout << "The bigger value is. " << larger << "\n";
        std::cout << "The smaller value is: " << smaller << "\n";
    }
    return tempNum;
}


int main()
{
    int smaller = userInput1();
    int larger = userInput2();
    ifElse(smaller, larger);
    return 0;
}
```

nascardriver
February 19, 2018 at 7:38 am · Reply

Hi Ali!

I've added comments addressing the issue with your code.
Try correcting it and post your progress, if you have any questions feel free to ask.

```
#include <iostream>

// This function is almost equivalent to @userInput2, DRY! (Don't repeat
// yourself)
int userInput1()
{
  // Initialize your variables, this goes for all other variables too.
  int smaller;
  std::cout << "Enter an integer: ";
  std::cin >> smaller;
  return smaller;
}

int userInput2()
{
  int larger;
  std::cout << "Enter a bigger integer: ";
  std::cin >> larger;
  return larger;
}

// Terrible function name
int ifElse(int smaller, int larger)
{
  int tempNum;

  if (smaller > larger)
  {

    tempNum = larger;
    larger = smaller;
    smaller = tempNum;

    std::cout << "Swapping the values\n";
    // This is equivalent to the code below, DRY!
```

```
36      std::cout << "The smaller value is: " << smaller << "\n";
37      std::cout << "The bigger value is: " << larger << "\n";
38    }
39    else
40    {
41      std::cout << "The bigger value is. " << larger << "\n";
42      std::cout << "The smaller value is: " << smaller << "\n";
43    }
44    return tempNum;
45  }
46
47  int main()
48  {
49    int smaller = userInput1();
50    int larger = userInput2();
51    ifElse(smaller, larger);
52    return 0;
53  }
```

> please explain this piece
It's swapping the values of @larger and @smaller. Imagine you have three baskets in front of you, one contains a red apple, one contains a green apple and one is empty.
You want the red apple to be in the green apple's basket and the green apple to be in the red apple's basket.
You are only allowed to hold one apple at a time.

PS: Please post your code in the corresponding lesson next time, this quiz is in lesson 4.1a

Ali Dahud
February 20, 2018 at 2:40 am · Reply

isn't that some kind of sorting?

nascardriver
February 20, 2018 at 7:01 am · Reply

No it's not, you only want two elements to change places.
EDIT: Well, technically, it might be sorting since you now have two elements in a specific order, but I would call it sorting when it's just two elements and not even a list/array.

Ali Dahud
February 20, 2018 at 7:06 am · Reply

i remember the bubble sorting used a method like this. or at least some kind of like this

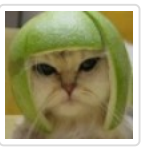
Alex
February 22, 2018 at 9:28 am · Reply

Yes, many sorting methods work by swapping elements until the list of items is sorted. Bubble sort swaps adjacent elements. Selection sort swaps once per iteration.

Generally, when we transpose the value of two elements, we call it a swap. We call it a sort when we put all the elements in a list or array into some sorted order.

Ali Dahud
February 23, 2018 at 3:57 am · Reply

Hi Alex, how many tutorials do you recommeend me on reading daily?



Alex
February 26, 2018 at 5:35 pm · Reply

I don't have a recommendation on this.

It really depends entirely on your learning capability, motivation, and how much time you want to spend experimenting with each topic on your own.

Ali Dahud
February 27, 2018 at 2:24 am · Reply

i have the time to read as much as i want and my learning capability is pretty big i guess. with each topic i guess 20 minutes-1 hour or 1,5 hour depending if it has coding quizzes in it. so how much should i read? oh and i lack motivation i guess so how should i motivate myself? for example i start watching videos, news while im supposed to learn.

Piyush
January 19, 2020 at 4:27 am · Reply

> in the ifElse function there's no need to return tempNum you can return larger either. And Its good to initialize tempNum directly above where you do swapping.
> ifElse returns something and you are not storing it. If you don't want to, change the functions type to void.

Amir
September 9, 2017 at 3:48 pm · Reply

Hi Alex. The Output of the first program has a mistake!

This is the code :

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int value;
    std::cin >> value;

    if (value >= 0)
    { // start of nested block
        std::cout << value << " is a positive integer (or zero)" << std::endl;
        std::cout << "Double this number is " << value * 2 << std::endl;
```

```
    } // end of nested block
    else
    { // start of another nested block
        std::cout << value << " is a negative integer" << std::endl;
        std::cout << "The positive of this number is " << -value << std::endl;
    } // end of another nested block

    return 0;
}
```

below it we have this output :

```
Enter an integer: 3
3 is a positive integer
Double this number is 6
```

I guess it should be like this :

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

Thank You! :))

Alex
September 9, 2017 at 9:01 pm · Reply

Thanks, fixed!

Nick
October 30, 2016 at 6:55 pm · Reply

Thank you so much for making all these.

On the 2ed code example if you put in 0 won't it fail when it gets to

```
cout << "Double this number is " << value * 2 << endl;
```

Alex
October 30, 2016 at 10:22 pm · Reply

No, why would it? 0 * 2 = 0.

Nick
October 31, 2016 at 7:24 am · Reply

Oh I see now it is division by zero.

Jack Oz
April 29, 2016 at 12:16 am · Reply

Alex thank you so much your time, you help many people. Are you familiar with socket programming in C++?

Best regards,
Jack

Alex
April 29, 2016 at 4:35 pm · Reply

I've used sockets in the distant past but I've probably forgotten more than I know about them. :(

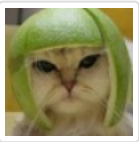
Piyush
January 19, 2020 at 4:32 am · Reply

Thats what happened to me not with sockets but core C++ :(thats why I am revising all the updated lessons.

Rajeg
December 31, 2015 at 11:21 am · Reply

Hey Alex,

maybe you should add a special case for 0 on your second program, now it says that 0 is a negative integer.



Alex
January 1, 2016 at 12:40 pm · Reply

Good catch. I updated the conditional on the first if statement to handle this properly.

Jim
October 12, 2015 at 3:45 pm · Reply

Alex,

On line 18 of the second program you have:
cout << "The positive of this number is " << -value << endl;.

Since you entered -4 for the input, how does the program know to print out 4 (positive)? when no math is ever done?

Does this somehow say take value and multiply it times -1 ?
In math we learned (-4 * -1)= 4.

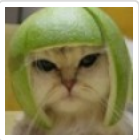
Piyush
January 19, 2020 at 4:34 am · Reply

-4 * -1 = 4
and -(-4) also equals 4 the program uses second math.

Mr D
September 2, 2015 at 2:12 pm · Reply

In the last code example above, you're missing a semi-colon at the end of the line(4):

```
1 cout << "Enter an integer: "
```

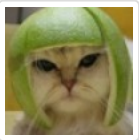


Alex
September 2, 2015 at 7:02 pm · Reply

Thanks, fixed!

techsavvy....aye
June 29, 2015 at 3:26 am · Reply

A minor note in the last program
it should be just value instead of nValue shouldn't it?
In block inside of block inside of block!
:-)



Alex
June 29, 2015 at 1:15 pm · Reply

Yep, fixed. Thanks for pointing that out.

Todd
June 19, 2015 at 2:50 pm · Reply

Typo.

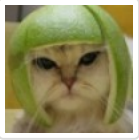
I would change the title,

"4.1 — Blocks (compound statements) and local variables"

to the title referenced on your main page

"4.1 — Blocks (compound statements)"

since you don't talk about local variables here.



Alex
June 20, 2015 at 7:58 pm · Reply

Good call. I used to talk about local variables here, but moved it into section 1.

Martin
February 23, 2015 at 9:01 am · Reply

Blockception.

Ignacio
August 29, 2016 at 6:35 am · Reply

best comment so far!!! lol

Pan
February 3, 2015 at 8:28 pm · Reply

Great!

Zidane
January 11, 2014 at 8:14 am · Reply

The best tutorials I've ever seen :P

posentel
October 24, 2011 at 8:12 pm · Reply

It may be helpful to point out that since a block replaces a single statement (that ends with a semicolon), the block supercedes the use of a semicolon. No semicolon is necessary to end a block.

Fluke
November 27, 2009 at 7:56 am · Reply

Now i see a benefit from declare_variable_when_needed instead of all at the top of the function. Cheers :)

davidv
September 8, 2008 at 10:34 am · Reply

Hi Alex,

Fabulous work, first of all.

I have a question: how does "using namespace std" work with the nesting? It seems that once written in a block, it is valid for the sub-blocks as well. If it is so, wouldn't it be easier to simply declare it at the beginning rather than several times inside functions?
Thanks.

Alex
September 10, 2008 at 11:30 pm · Reply

As you note, using statements also use normal scoping rules -- if used inside a block, it applies to the block and all subblocks.

learning c++
February 7, 2008 at 9:13 am · Reply

Hey Alex when you read this would like to suggest adding something like a review at the end of every section just quickly reminding people like for example on the variables section 1 it would say all the variable types with quick meanings and examples maybe.