5.8 — Break and continue

BY ALEX ON JUNE 26TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON FEBRUARY 7TH, 2020

Break

Although you have already seen the break statement in the context of switch statements, it deserves a fuller treatment since it can be used with other types of loops as well. The break statement causes a do, for, switch, or while statement to terminate.

Breaking a switch

In the context of a switch statement, a break is typically used at the end of each case to signify the case is finished (which prevents fall-through):

```
switch (ch)
{
    case '+':
        doAddition(x, y);
        break;
    case '-':
        doSubtraction(x, y);
        break;
    case '*':
        doMultiplication(x, y);
        break;
    case '/':
        doDivision(x, y);
        break;
}
```

Breaking a loop

In the context of a loop, a break statement can be used to cause the loop to terminate early:

```
#include <iostream>

int main()
{
    int sum{ 0 };

    // Allow the user to enter up to 10 numbers
    for (int count{ 0 }; count < 10; ++count)
    {
        std::cout << "Enter a number to add, or 0 to exit: ";
        int num{};
        std::cin >> num;

        // exit loop if user enters 0
        if (num == 0)
            break;

        // otherwise add number to our sum
        sum += num;
    }

    std::cout << "The sum of all the numbers you entered is " << sum << "\n";

    return 0;
}
```

This program allows the user to type up to 10 numbers, and displays the sum of all the numbers entered at the end. If the user enters 0, the break causes the loop to terminate early (before 10 numbers have been entered).

Note that break can be used to get out of an infinite loop:

```cpp
#include <iostream>

int main()
{
    while (true) // infinite loop
    {
        std::cout << "Enter 0 to exit or anything else to continue: ";
        int num{};
        std::cin >> num;

        // exit loop if user enters 0
        if (num == 0)
            break;
    }

    std::cout << "We're out!\n";

    return 0;
}
```

Break vs return

New programmers often have trouble understanding the difference between break and return. A break statement terminates the switch or loop, and execution continues at the first statement beyond the switch or loop. A return statement terminates the entire function that the loop is within, and execution continues at point where the function was called.

```cpp
#include <iostream>

int breakOrReturn()
{
    while (true) // infinite loop
    {
        std::cout << "Enter 'b' to break or 'r' to return: ";
        char ch;
        std::cin >> ch;

        if (ch == 'b')
            break; // execution will continue at the first statement beyond the loop

        if (ch == 'r')
            return 1; // return will cause the function to immediately return to the caller (i
    }

    // breaking the loop causes execution to resume here

    std::cout << "We broke out of the loop\n";

    return 0;
}

int main()
{
    int returnValue{ breakOrReturn() };
    std::cout << "Function breakOrReturn returned " << returnValue << '\n';

    return 0;
}
```

Continue

The continue statement provides a convenient way to jump to the end of the loop body for the current iteration. This is useful when we want to terminate the current iteration early.

Here’s an example of using continue:

```
for (int count{ 0 }; count  < 20; ++count)
{
    // if the number is divisible by 4, skip this iteration
    if ((count % 4) == 0)
        continue; // jump to end of loop body

    // If the number is not divisible by 4, keep going
    std::cout << count << std::endl;

    // The continue statement jumps to here
}
```

This program prints all of the numbers from 0 to 19 that aren’t divisible by 4.

In the case of a for loop, the end-statement of the for loop still executes after a continue (since this happens after the end of the loop body).

Be careful when using a continue statement with while or do-while loops. Because these loops typically increment the loop variables in the body of the loop, using continue can cause the loop to become infinite! Consider the following program:

```
int count{ 0 };
while (count < 10)
{
    if (count == 5)
        continue; // jump to end of loop body

    std::cout << count << ' ';

    ++count; // this statement is never executed after count reaches 5

    // The continue statement jumps to here
}
```

This program is intended to print every number between 0 and 9 except 5. But it actually prints:

```
0 1 2 3 4
```

and then goes into an infinite loop. When count is 5, the *if statement* evaluates to true, and the loop jumps to the bottom. The count variable is never incremented. Consequently, on the next pass, count is still 5, the *if statement* is still true, and the program continues to loop forever.

Here’s an example with a do-while loop using continue correctly:

```
int count{ 0 };
do
{
    if (count == 5)
        continue; // jump to end of loop body
    std::cout << count << ' ';

    // The continue statement jumps to here
} while (++count < 10); // this still executes since it's outside the loop body
```

This prints:

```
0 1 2 3 4 6 7 8 9
```

Using break and continue

Many textbooks caution readers not to use break and continue, both because it causes the execution flow to jump around and because it can make the flow of logic harder to follow. For example, a break in the middle of a complicated piece of logic could either be missed, or it may not be obvious under what conditions it should be triggered.

However, used judiciously, break and continue can help make loops more readable by keeping the number of nested blocks down and reducing the need for complicated looping logic.

For example, consider the following program:

```cpp
#include <iostream>

int main()
{
    int count{ 0 }; // count how many times the loop iterates
    bool keepLooping { true }; // controls whether the loop ends or not
    while (keepLooping)
    {
        std::cout << "Enter 'e' to exit this loop or any other character to continue: ";
        char ch{};
        std::cin >> ch;

        if (ch == 'e')
            keepLooping = false;
        else
        {
            ++count;
            std::cout << "We've iterated " << count << " times\n";
        }
    }

    return 0;
}
```

This program uses a boolean variable to control whether the loop continues or not, as well as a nested block that only runs if the user doesn’t exit.

Here’s a version that’s easier to understand, using a break statement:

```cpp
#include <iostream>

int main()
{
    int count{ 0 }; // count how many times the loop iterates
    while (true) // loop until user terminates
    {
        std::cout << "Enter 'e' to exit this loop or any other character to continue: ";
        char ch;
        std::cin >> ch;

        if (ch == 'e')
```

```
            break;

        ++count;
        std::cout << "We've iterated " << count << " times\n";
    }

    return 0;
}
```

In this version, by using a single break statement, we've avoided the use of a boolean variable (and having to understand both what its intended use is, and where it is set), an else statement, and a nested block.

Minimizing the number of variables used and keeping the number of nested blocks down both improve code understandability more than a break or continue harms it. For that reason, we believe judicious use of break or continue is acceptable.

5.9 -- Random number generation

Index

5.7 -- For statements

C++ TUTORIAL | PRINT THIS POST

92 comments to 5.8 — Break and continue

« Older Comments 1 2

comment
February 7, 2020 at 3:23 am · Reply

Not sure, if this has been mentioned already, but there are 2 issues in the "Continue" section:

Second code snippet:

```
++count; // this statement is never executed
```

That comment is misleading; the statements IS executed during the first 5 iterations of the loop.

Furthermore the output of the third code snippet should be

```
0 1 2 3 4
```

too, since the cout line is skipped in iterations 6 to 10

nascardriver
February 7, 2020 at 8:58 am · Reply

> the statements IS executed during the first 5 iterations of the loop.
Comment amended, thanks!

> Furthermore the output of the third code snippet should be
The third snippet in the "continue" section? That's correct as written. The condition is `i == 5`, not `i >= 5`.

Benur21
August 5, 2019 at 1:32 pm · Reply

Could also include exit() which is used to skip the rest of the program

nascardriver
August 6, 2019 at 2:56 am · Reply

`std::exit` shouldn't be used unless normal termination of your program is not otherwise possible. `std::exit` makes the control flow harder to follow.

Benur21
August 2, 2019 at 12:26 pm · Reply

Summarizing:

return is used to skip the current function;
break is used to skip the current loop or switch;
continue is used to skip the rest of the current iteration.

Anmol sharma
April 27, 2019 at 9:19 am · Reply

Sir, code mentioned below is not working and it is taken from your example which is mentioned above. It is working when i am using break; instead of continue. please check!

```
int count(0);
while (count < 10)
{
  if (count == 5)
    continue; // jump to end of loop body
  std::cout << count << " ";
  ++count;

  // The continue statement jumps to here
}
```

nascardriver
April 27, 2019 at 9:30 am · Reply

The text above the example explains it.
"Be careful when using a continue statement with while or do-while loops. Because these loops typically increment the loop variables in the body of the loop, using continue can cause the loop to become infinite!"

Dustin
January 5, 2019 at 10:18 am · Reply

When I created this small bit of code to practice using continue, VS2017 required me to initialize "int count" outside of the for loop. Why? I originally wrote it in the init-statement, but I would get an error that "count" wasn't defined. **Edit: I can't seem to get the code tags to work. Not sure what I'm missing...

```
#include <iostream>

int main()
{
    int count{ 0 };
    for (; count <= 50; ++count)
    {
        if ((count % 2) == 0)
            continue;
        else
            std::cout << count << std::endl;
    }
}
```

nascardriver
January 6, 2019 at 5:11 am · Reply

Hi Dustin!

```
[-CODE]
int main()
{
  ...
}
[-/CODE]
```
without the -

The following code is working. If it doesn't work for you, please share the exact error message, including the line number(s).

```
#include <iostream>

int main()
{
  for (int count{ 0 }; count <= 50; ++count)
```

```
    {
      if ((count % 2) == 0)
        continue;
      else
        std::cout << count << '\n';
    }

    return 0;
}
```

Dustin
January 6, 2019 at 3:51 pm · Reply

smh... I don't think I actually initialized it when I included it in the init statement. You're right. It's working.

On an unrelated note, but still having to do with VS2017, while I'm going through these tutorials, I'm leaving VS up and creating small programs to experiment with what I'm learning. The other day, to mess with my son, I wrote a program in this VS2017 program window I always have open that asked the user for their name and if it matched my son's name, it would print out that he stunk. Otherwise, it would print out that the user didn't stink. (I know, it's juvenile). After I messed with him, I altered the code to say I stunk if the name input matched my name. What was odd is that even though I had deleted the code that told the program that if the string input matched my son's name the program was still recognizing when you entered his name and out put that he stunk. The program was still acting like the code was there!

Have you ever had something like this happen where VS remembered code that you had deleted and executed it? I had saved the new code and I tried "cleaning" the project. It was like that comparison bit of code was still hanging out in the memory or something. Thoughts?

nascardriver
January 7, 2019 at 6:19 am · Reply

If old code gets stuck:
- Make sure you restarted to program
- Make sure you saved the changes (This happens too often)
- Make sure you're building the project (VS can debug/run without re-building)
- Clean the project (Delete everything that's not a code or project file)

Some systems cache dynamic libraries. If you're developing one, disable/clean the cache or rename the library.

Dustin
January 7, 2019 at 10:50 am · Reply

Thank you!

Anson
December 14, 2018 at 3:59 am · Reply

Actually, break statement should be omitted in this way without using any variables.
This was what my teacher told me.

```
#include <iostream>
```

```
2
3   int main()
4   {
5       int count(0); // count how many times the loop iterates
6       do { // loop until user terminates
7           std::cout << "Enter 'e' to exit this loop or any other character to continue: ";
8           char ch;
9           std::cin >> ch;
10          if (ch != 'e') {
11              ++count;
12              std::cout << "We've iterated " << count << " times\n";
13          }
14      } while (ch != 'e');
15
16      return 0;
17  }
```

nascardriver
December 14, 2018 at 6:53 am · Reply

Hi Anson!

Your teacher is wrong. Your code will compare @ch to 'e' once in line 10 and once in line 14, without @ch changing in-between. This can introduce a significant performance overhead, depending on the loop-condition. On top of that, your sample has duplicate code, which can cause additional problems when updating the code. Both of Alex' snippets should be preferred over your teacher's solution.

Your teacher has a valid point in not liking @break, because it can make understanding the control flow more difficult. You should not let this affect the efficiency or stability of your code!

Kumar
July 20, 2018 at 11:50 pm · Reply

In the "Break vs return" segment, there is a typo:

```
std::cout << "Function breakOrContinue returned " << returnValue << '\n';
```

It should be

```
std::cout << "Function breakOrReturn returned " << returnValue << '\n';
```

The output looks weird but the function name printed would be inline with the function name in the code.

« Older Comments 1 2