5.1 — Control flow introduction

BY ALEX ON JUNE 20TH, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

When a program is run, the CPU begins execution at the top of main(), executes some number of statements, and then terminates at the end of main(). The sequence of statements that the CPU executes is called the program's **execution path** (or path, for short). Most of the programs you have seen so far have been **straight-line programs**. Straight-line programs have **sequential flow** -- that is, they take the same path (execute the same statements) every time they are run (even if the user input changes).

However, often this is not what we desire. For example, if we ask the user to make a selection, and the user enters an invalid choice, ideally we'd like to ask the user to make another choice. This is not possible in a straight-line program. Alternatively, there are cases where we need to do something a number of times, but we don't know how many times at compile time. For example, if we wanted to print all of the integers from 0 to some number the user entered, we couldn't do that until we know what number the user entered.

Fortunately, C++ provides **control flow statements** (also called *flow control statements*), which allow the programmer to change the CPU's path through the program. There are quite a few different types of control flow statements, so we will cover them briefly here, and then in more detail throughout the rest of the chapter.

Halt

The most basic control flow statement is the **halt**, which tells the program to quit running immediately. In C++, a halt can be accomplished through use of the exit() function that is defined in the cstdlib header. The exit function takes an integer parameter that is returned to the operating system as an exit code, much like the return value of main().

Here is an example of using exit():

```
#include <cstdlib> // needed for exit()
#include <iostream>

void cleanup()
{
    // code here to do any kind of cleanup required
}

int main()
{
    std::cout << 1;
    cleanup();

    exit(0); // terminate and return 0 to operating system

    // The following statements never execute
    std::cout << 2;
    return 0;
}
```

Note that exit() works no matter what function it's called from (even a function other than main). Also note that exit() terminates the program immediately with minimal cleanup. Therefore, before calling exit(), you should consider whether any manual cleanup is required (e.g. saving the user's game to disk).

Most often, exit() is used to immediately terminate the program when some catastrophic, unrecoverable error occurs.

Jumps

The next most basic flow control statement is the jump. A **jump** unconditionally causes the CPU to jump to another statement. The *goto*, *break*, and *continue* keywords all cause different types of jumps -- we will discuss the difference between these in upcoming sections.

Function calls also cause jump-like behavior. When a function call is executed, the CPU jumps to the top of the function being called. When the called function ends, execution returns to the statement after the function call.

Conditional branches

A **conditional branch** is a statement that causes the program to change the path of execution based on the value of an expression. The most basic conditional branch is an *if statement*, which you have seen in previous examples. Consider the following program:

```
int main()
{
    // do A
    if (expression)
        // do B
    else
        // do C

    // do D
}
```

This program has two possible paths. If expression evaluates to true, the program will execute A, B, and D. If expression evaluates to false, the program will execute A, C, and D. As you can see, this program is no longer a straight-line program -- its path of execution depends on the value of expression.

The *switch* keyword also provides a mechanism for doing conditional branching. We will cover if statements and switch statements in more detail in an upcoming section.

Loops

A **loop** causes the program to repeatedly execute a series of statements until a given condition is false. For example:

```
int main()
{
    // do A
    // loop on B, 0 or more times
    // do C
}
```

This program might execute as ABC, ABBC, ABBBC, ABBBBC, or even AC. Again, you can see that this program is no longer a straight-line program -- its path of execution depends on how many times (if any) the looped portion executes.

C++ provides 3 types of loops: *while*, *do while*, and *for* loops. C++11 added support for a new kind of loop called a *for each* loop. We will discuss loops at length toward the end of this chapter, except for the *for each* loop, which we'll discuss a little later.

Exceptions

Finally, **exceptions** offer a mechanism for handling errors that occur in a function. If an error occurs in a function that the function cannot handle, the function can trigger an exception. This causes the CPU to jump to the nearest block of code that handles exceptions of that type.

Exception handling is a fairly advanced feature of C++, and is the only type of control flow statement that we won't be discussing in this section. We discuss exceptions in chapter 14.

Conclusion

Using program flow statements, you can affect the path the CPU takes through the program and control under what conditions it will terminate. Prior to this point, the number of things you could have a program do was fairly limited. Being able to control the flow of your program makes any number of interesting things possible, such as displaying a menu repeatedly until the user makes a valid choice, printing every number between x and y, or determining the factors of a number.

Once you understand program flow, the things you can do with a C++ program really open up. No longer will you be restricted to toy programs and academic exercises -- you will be able to write programs that have real applications. This is where the real fun begins. So let's get to it!

5.2 -- If statements

Index

S.7.x -- Chapter 7 summary and quiz

C++ TUTORIAL | PRINT THIS POST

47 comments to 5.1 — Control flow introduction

Nikola
January 25, 2020 at 11:14 pm · Reply

Hey guys,

Quick question. Are you sure that including cstdlib is necessary to be able to call exit(0)?
I just tried compiling this code and forgot to put #inlcude <cstdlib> and the program compiled and ran correctly. I'm using GNU GCC on Ubuntu 18.04. Is this behavior some kind of GCC extension or something else?

Best regards,
Nikola

nascardriver
January 26, 2020 at 2:05 am · Reply

Yes, you need <cstdlib>. If you code works without it, you're including a header that itself includes <cstdlib>. It's not guaranteed that your code works with other compilers. If you use something, include its header.

Nikola
January 26, 2020 at 3:18 am · Reply

Damn... I must be missing something then. Would you be so kind to help me resolve this, nascardriver?

This is my main.cpp:
<code>
#include <iostream> // Allows usage of the console output and input

#include "chapter_L_5_1_example_0_version.hpp"

void cleanup(void)
{
}

// Main program function.
int main(int argc, char **argv)
{
  std::cout << "Version: "
    << VERSION_chapter_L_5_1_example_0_MAJOR << "."
    << VERSION_chapter_L_5_1_example_0_MINOR << "."
    << VERSION_chapter_L_5_1_example_0_PATCH << "\n\n";

  cleanup();

  exit(0);

  std::cout << "This will never execute!\n";

  // Signal successful program termination.
  return 0;
}
</code>

and finally, this is the chapter_L_5_1_example_0_version.hpp:
<code>
#ifndef CHAPTER_L_5_1_EXAMPLE_0_VERSION_HPP
#define CHAPTER_L_5_1_EXAMPLE_0_VERSION_HPP

// Version definition for the target_0172
#define VERSION_chapter_L_5_1_example_0_MAJOR 0
#define VERSION_chapter_L_5_1_example_0_MINOR 0
#define VERSION_chapter_L_5_1_example_0_PATCH 0

#endif // CHAPTER_L_5_1_EXAMPLE_0_VERSION_HPP
</code>

This is as simple as it gets... Or at least I think so.

Thank you for the quick reply, nascardriver.

P.S.
Would you mind telling me how to format the code here in the comment section?

nascardriver
January 26, 2020 at 3:37 am · Reply

Resolve what exactly? Your system's implementation of iostream includes cstdlib somehow. That's not standard behavior, but you can't change it.

> Would you mind telling me how to format the code here in the comment section?
Square brackets [], not the triangle ones <>

Nikola
January 27, 2020 at 4:30 am · Reply

Then it must have something to do with Ubuntu Linux and the implementation of GCC. Thank you for the information.

Chase
May 14, 2019 at 4:48 pm · Reply

"When a program is run, the CPU begins execution at the top of main(), executes some number of statements, and then terminates at the end of main()."

The tense of "run" seems off. Perhaps you meant "When a program is ran" or "When a program runs".

iii
April 12, 2019 at 4:43 am · Reply

```
#include <cstdlib> // needed for exit()
#include <iostream>

void cleanup
{
    // code here to do any kind of cleanup required
}
```

don't you need a parenthesis after cleanup? compile error occurred

Alex
April 14, 2019 at 7:50 am · Reply

Yup, thanks. Fixed.

Nigel Booth
July 31, 2018 at 11:24 pm · Reply

in C# exceptions can be caught and handled thus:

```
using namespace example

try
{
    function1()
    {
        // some stuff here
    }

    function2()
    {
        // some stuff here
    }
}

catch

{
```

```
19      // code to deal with error
20  }
```

is it similar in C++ or are the try... catch... keywords just in C#

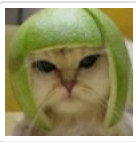
nascardriver
August 1, 2018 at 6:47 am · Reply

Hi Nigel!

Exceptions are covered in chapter 14. They're not as commonly used in cpp as in C# or Java.

Michaek Wycombe
April 12, 2018 at 3:34 pm · Reply

Oops; exceptions are in Ch14 not Ch15 as referenced in this chapter



Alex
April 13, 2018 at 10:40 pm · Reply

Thanks. Fixed!

Eri
March 5, 2018 at 3:14 pm · Reply

If you had to roughly say, is all the material on this site about a semesters worth at a college or university?

nascardriver
March 6, 2018 at 1:51 am · Reply

Hi Eri!

This depends on the country, university, module and requirements to take that module.
If the module doesn't have any requirements students without any experience in programming should be able to pass it. This means, you'll probably learn C (Basic data types, loops, if, functions) and write a couple of algorithms.
On person mentioned his university getting all to way to chapter 12, he didn't let me know the country/university unfortunately.
If you finish all 18 chapters everything that you university could possibly ask you to do should've been covered. You might want to do chapter(s 17 and) 18 early, since those seem likely to be necessary earlier in university.

Alex
March 6, 2018 at 3:30 pm · Reply

My best guess is probably more like a semester and a half.

Eri
March 6, 2018 at 8:53 pm · Reply

Very interesting, im gonna ask my local colleges and universities. ill keep you posted.

Thanks.

WiseFool
March 10, 2018 at 11:55 am · Reply

Not only that, but it cleverly weaves in a quite a few topics that would often be part of a separate Computer Science class instead. (Great website!! Thank you so much, Alex! Keep it up!)

rahul
February 17, 2018 at 5:34 am · Reply

Good Evening sir,
I am talking about the inbuilt getpass function(a function of conio.h header file) which reads a password from the console

how to declare this func??

Alex
February 19, 2018 at 10:58 am · Reply

See **https://code-reference.com/c/conio.h/getpass**.

For what it's worth, conio.h is a C-style, DOS specific header, so you should generally avoid anything from there if you have the choice, as it likely won't compile anywhere else.

rahul
February 15, 2018 at 6:31 am · Reply

good evening sir
I want to ask a question

sir how can we declare getpass function?

Alex
February 15, 2018 at 5:57 pm · Reply

```
1 void getpass()
2 {
3     std::cout << "What is this function supposed to do?";
4 }
```

Raghav
September 19, 2019 at 9:07 pm · Reply

He wants to declare a function not to define and u have written code to define that function instead of declaring.
so the code will be:
void getpass();
And this should be written above main function.

Zero Cool
September 17, 2017 at 2:01 pm · Reply

I think you should change "chapter 15" for "chapter 14" in the following text "We discuss exceptions in chapter 15."

Keep up with the good work Alex!

Zero Cool
September 17, 2017 at 1:48 pm · Reply

Hi Alex. I have a question for you:
I know you use Microsoft Visual Studio for compile your examples. But when you do an application do you use the same software? Or do you use just a text editor and compile with the console?

And do you use Windows, MacOS or Linux for development and why?

I'm just curious about what is the best software and OS for developers.

Thanks again for all your work in this website.

Brian Gaucher
January 6, 2019 at 6:22 pm · Reply

Disclaimer: This answer is subjective and based on my own opinion.
I started out programming on Windows using IDEs (Like PyCharm or IntelliJ), I found that I became dependant on the IDE auto-complete and highlighting too much and didn't fully understand the syntax.

A little while later, I switched to Linux (I started with dual-boot), I tried Code-Blocks, and I might try again later, I enjoyed it's simplicity, and performance compared to do-it-all IDE. But currently I use the linux terminal for my programming. I use Vim for text-editing, and g++ for compiling. Not having under-line, automatic compiling, and auto-complete forces me to understand the basics more. If you can't get the basics, give up on the advance. IDEs tend make beginners lazy with the basics, then we struggle with the advance. So I suggest use the most simple tools possible (feel free to use a graphical text editor bro), then later, use IDEs in large projects to save time finding small errors.

Merlin
May 7, 2019 at 8:05 pm · Reply

I think you should use the tool, you feel comfortable with. Visual Studio is pretty easy and it's got all what a developer needs at the beginning. But there are many other IDEs which are worth testing as well! :)

Weejtoen
August 5, 2016 at 11:09 am · Reply

Thank you very much for the nice tutorial. I have been trying to learn C++ for a while now, but never found the right tutorial, untill this website!

Cathrine
June 3, 2016 at 1:16 am · Reply

Hello Hello Alex. Am from Africa in country called Kenya.Am a student and i want to be a programmer and since started learning using this site its much easier to understand C++ and am

looking forward to finish learning C++and learn JAVA with your site. God Bless.

Jim
November 13, 2015 at 6:39 am · Reply

Alex,
This may not be the proper place to ask this, but suppose I want to pause the program for a while. Lets say 10 to 20 seconds more or less. Is there a pause function or some way to do this without just using cout << "Press any key to continue";, in C++.

Jim
November 13, 2015 at 7:14 am · Reply

Alex,
Continuing from the above. Edit didn't work for some reason. I'm looking for two methods to do this. A function to pause the program waiting for the user to hit any key to continue and another that times out after a given time.

Alex
November 14, 2015 at 12:02 pm · Reply

A method to pause the program until the user hits a key is shown in lesson 0.7.

Alex
November 14, 2015 at 12:02 pm · Reply

Prior to C++11, there's no way to do this in standard C++. Different operating systems have OS-specific functionality to do this -- for example, on Windows, windows.h contains function Sleep(), which takes a parameter indicating how many milliseconds to sleep for.

In C++11, you can do this:

```
#include <chrono>
#include <thread>

std::this_thread::sleep_for(std::chrono::milliseconds(x));
```

papaz
October 20, 2015 at 2:48 am · Reply

hey man, thanks for the series, amazing work.
can you advice on a more detailed exercises we can use?

techsavvy....aye
July 15, 2015 at 10:37 pm · Reply

Typo, in "Loops" for is printed twice.

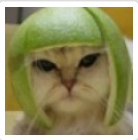
Todd
July 1, 2015 at 11:37 am · Reply

I like the fact that you've introduced each of these powerful concepts with a hint to their potential before diving in to the syntax.

The following independent clause confused me:

"control jumps to the nearest block of code that has declared it is willing to catch exceptions of that type."

Is "control jumps" a noun that "is willing to catch exceptions", or is "jumps" being used as a verb (in which case "it is" is confusing and grammatically incorrect). I would recommend rewriting this clause. I can't tell what you're trying to say here.



Alex
July 1, 2015 at 12:06 pm · Reply

I rewrote a few of the sentences to:

Finally, **exceptions** offer a mechanism for handling errors that occur in a function. If an error occurs in a function that the function cannot handle, the function can trigger an exception. This causes the CPU to jump to the nearest block of code that handles exceptions of that type.

Is that clearer?

Todd
July 2, 2015 at 9:11 am · Reply

Much clearer - thank you!

neelam kumari
March 10, 2015 at 1:58 am · Reply

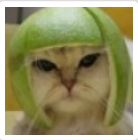
awsummmmmmmmm

FrostByte
February 22, 2015 at 6:06 am · Reply

"Unlike more modern programming languages, such as C# or D, C++ does not provide a foreach keyword."

This is not true. C++11 provides for-each loops! At least it should be noted there.

Source: http://en.wikipedia.org/wiki/Foreach_loop#C.2B.2B



Alex
May 22, 2015 at 2:18 pm · Reply

Yup! The tutorials are being updated to account for this.

Mike
March 15, 2010 at 9:59 am · Reply

Can we use break;'s instead of exit?

zingmars
May 28, 2011 at 10:03 am · Reply

break; only breaks a loop in which it is in, while exit() closes the program.

Uri
April 15, 2009 at 5:07 pm · Reply

Hi Alex!!

Just onted to say WOW!!

Your C++ tutorial is AMAZING!!! this clears soooooooo many things for me...very easy to understand - without missing anything!

So thanks alot man!

Jeff
March 13, 2008 at 8:19 am · Reply

Alex, what is the difference, if any, between using a "halt":

```
1 if (bIsError1)
2     exit(-1);
```

As opposed to using a standard "return" type of statement:

```
1 if (bIsError2)
2     return -1;
```

I'm sure I have written and compiled code before that has multiple return statements, to be executed under various conditions, within the same main(). Is there a reason to favor one type of statement over the other?

UPDATE: I may have partially answered my own question. If exit() is called within a function other than main() does the program terminate outright rather than returning control to main()? Side-effects? Any other reasons to use, or not use, exit()?

Alex
March 13, 2008 at 1:18 pm · Reply

exit() will cause the program to terminate immediately, regardless of where it is called.

One thing you'll need to be careful of when using exit is to make sure you do any necessary cleanup before the program exits. Most of the time, no explicit cleanup is needed. However, it's probably a good idea to close any open file handles, network sockets, etc... before exiting. Note that destructors to in-scope variables will not be called before the program exits.

Peter Baum
March 12, 2018 at 5:23 pm · Reply

You might want to consider adding this information about exit() in the lesson where it will be more easily found.

Alex
March 13, 2018 at 12:26 pm · Reply

Good idea. Done. Thanks for the suggestion.