0.1 — Introduction to these tutorials

BY ALEX ON MAY 27TH, 2007 | LAST MODIFIED BY ALEX ON MAY 31ST, 2017

Welcome!

Welcome to the Learn C++ tutorials! Above all else, these tutorials aim to make learning C++ easy.

Unlike many other sites and books, these tutorials don't assume you have any prior programming experience. We'll teach you everything you need to know as you progress, with *lots* of examples along the way.

Whether you're interested in learning C++ as a hobby or for professional development, you're in the right place!

Tutorial structure

The tutorials in this introductory chapter are aimed at giving you some context around what C++ is, how it came about, how programs work, and what software you need to install to create your own programs. You'll even write your own first program. Further chapters will explore different parts of the C++ language. In the first chapter (chapter 1), you'll get a broad but shallow overview of many fundamental C++ concepts, so we can start writing some simple programs. Further chapters will explore those concepts in depth, or introduce new concepts.

Each chapter has a general theme, with all of the sections underneath it being related to that theme. There is no suggested amount of time that you should spend with each lesson or chapter; progress through the material at a pace that is comfortable for you.

Goals

Before we get started, lets hit on a couple of important goals of these tutorials:

- Cover programming topics as well as C++. Traditional textbooks do a pretty good job of teaching the basics of a given programming language, but they often do not cover relevant programming topics that are incidental to the language. For example, books will omit sections on programming style, common pitfalls, debugging, good/bad programming practices, and testing. Consequently, by the time you finish the book, you understand how to program in a language, but you have a ton of bad habits that will come back to bite you later! One of the goals of these tutorials is to make sure that all of these incidental topics are covered along the way, in the sections where it naturally makes sense to discuss them. When you finish, you will not only know how to program in C++, you will know how NOT to program in C++, which is arguably as important.

- Provide a lot of examples. Most people learn as much or more from following the examples as they do from reading the text. This tutorial will endeavor to provide plenty of clear, concise examples to show how to apply the concepts you are learning. We will also avoid (as much as possible) the twin evils: the magic hand wave (also known as …), where in the interest of space part of an example is omitted, and the unexplained new concept, where a new concept that is integral to the example is introduced without any mention of what it is or how it works. Both of these tend to lead to getting stuck.

- Provide practice programs. The end of many lessons and sections will contain some exercises that you can attempt to answer on your own, along with solutions. You can compare your solution against ours to see what we did differently, or, if you get stuck, how we solved the problem. Then you can go back and refocus on the areas you need more work on.

- Most importantly: have fun. Programming can be a lot of fun, and if you're not generally having fun, you're not in the right mindset to be programming. Tired or unhappy programmers make mistakes, and debugging code tends to take much longer than writing it correctly in the first place! Often you can save yourself some time by going to bed, getting a good night's sleep, and coming back to a problem in the morning.

Getting the most out of these tutorials

As you go through these tutorials, we recommend a number of practices to maximize your learning experience:

- Type in the examples by hand and compile them yourself. Do not copy and paste them! This will help you learn where you commonly make errors, as well as becoming familiar with compiler warnings and errors. As you type in the examples, think about why each of the things you are typing in make sense.

- As you make mistakes or find bugs in your program, fix them. Try to solve your own problems before asking others for help.

- Experiment with the examples. Change numbers and text to see what happens. Modify the programs to do additional things (e.g. if a program adds two numbers, make it add three numbers). Try to find different ways to break the programs (if a program asks for user input, try a variety of different inputs). You'll learn as much from modifying the examples as you will by following them.

- Write your own short programs using the concepts you have learned. Nothing is better than practice.
- Learn to debug your programs when they don't work. This is critical to solving your own problems, and is a skill that many new programmers skip to their detriment. We'll have more information on how to do this in a future lesson.

Note: The majority of the examples in the tutorials are full programs that you can compile and run yourself. However, occasionally the examples will be "snippets" of code that are designed to quickly illustrate a concept. Because these aren't full programs, they won't compile without some additional work. You can turn these into full programs yourself, if you desire.

Is there a PDF version of this site available for offline viewing?

Unfortunately, there is not. The site is able to stay free for everyone because we're ad-sponsored -- that model simply doesn't work in PDF format. You are welcome to convert pages from this website into PDF (or any other) format for your own private use, so long as you do not distribute them.

These tutorials were written in 2007. Are they still relevant?

Yes, absolutely. C++ doesn't change very often, and these tutorials have been largely kept up to date.

What should I do if I get stuck?

If you don't understand something, read through the comments. Other readers may have encountered similar challenges. Second, try scanning through the next lesson in the series -- your question may be answered there. Third, use a search engine (we recommend Google) to see if your question (or error message) has been addressed elsewhere. Fourth, ask your question on a site that is designed for programming Q&A, like **Stack Overflow**.

If all else fails, skip the material you don't understand, and come back to it when needed. You may find that something that was hard to understand is easier with the additional knowledge and context provided by other articles.

Alright, let's get on with it!