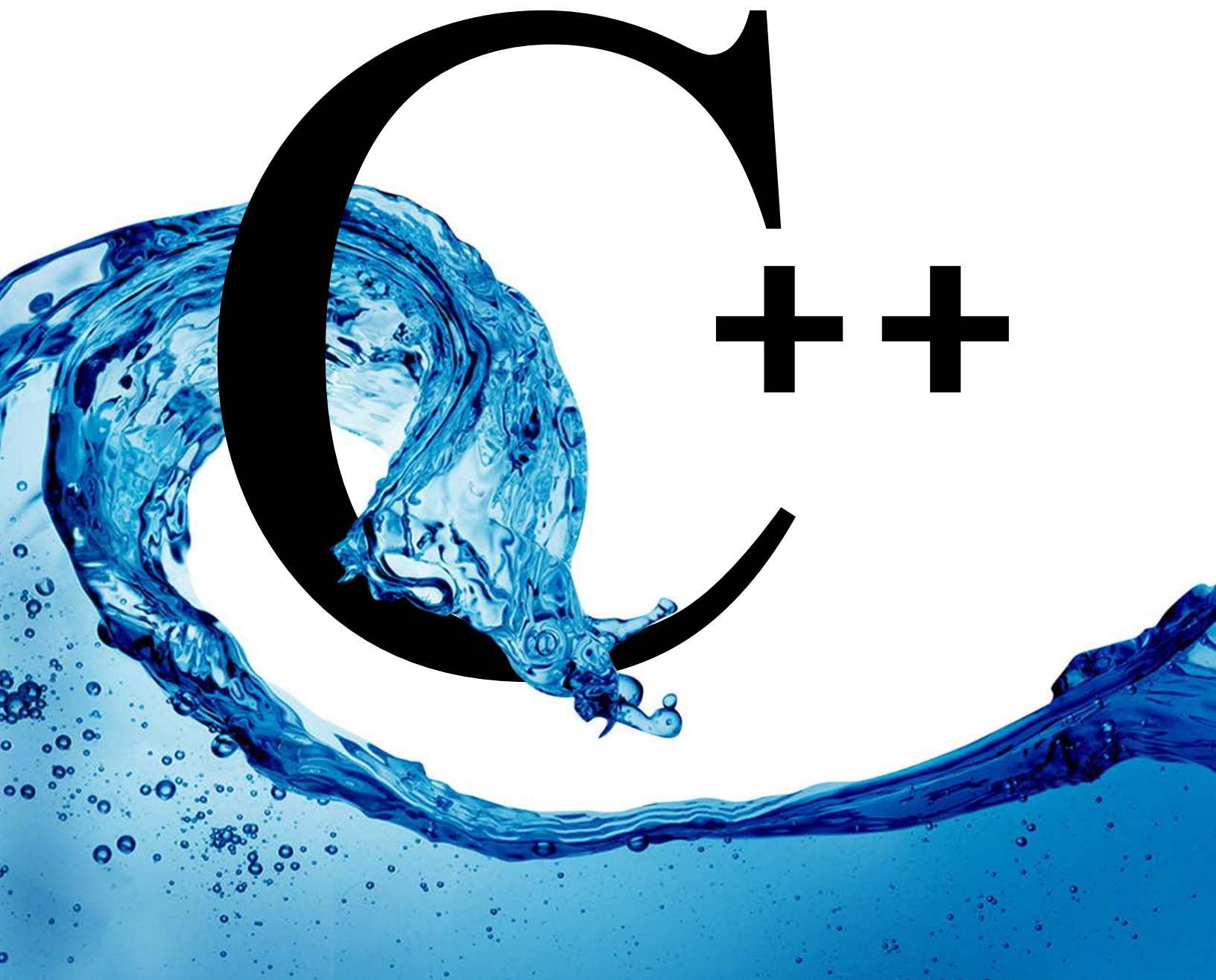


ALEX ALLAIN

CREATOR OF CPROGRAMMING.COM

# JUMPING INTO



## Contents

Part 1: Jumping into C++ .....	13
Acknowledgements.....	14
Chapter 1: Introduction and Developer Environment Setup.....	15
What is a programming language? .....	15
I've heard of a language called C, what's the difference between C and C++? .....	15
Do I need to know C to learn C++? .....	15
Do I need to know math to be a programmer? .....	15
Terminology .....	16
Programming .....	16
Executable .....	16
Editing and compiling source files .....	16
A note about sample source code .....	16
Windows .....	17
Step 1: Download Code::Blocks .....	17
Step 2: Install Code::Blocks .....	17
Step 3: Running Code::Blocks .....	17
Troubleshooting.....	23
What exactly is Code::Blocks?.....	25
Macintosh .....	25
Xcode.....	26
Installing Xcode .....	26
Running Xcode .....	26
Creating your first C++ program in Xcode.....	26
Installing Xcode 4 .....	31
Running Xcode .....	31
Creating your first C++ program in Xcode.....	32
Troubleshooting.....	37
Linux.....	39
Step 1: Installing g++ .....	40
Step 2: Running g++ .....	40
Step 3: Running your program.....	40
Step 4: Setting up a text editor .....	41
Configuring Nano .....	41

Using Nano .....	42
Chapter 2:    The Basics of C++ .....	45
Intro to the C++ language .....	45
The simplest C++ program .....	45
What happens if you don't see your program? .....	47
The basic structure of a C++ program .....	47
Commenting your programs .....	48
Thinking like a programmer and creating reusable code .....	49
A few words on the joys and pain of practice .....	49
Quiz yourself .....	50
Practice problems .....	51
Chapter 3:    User Interaction and Saving Information with Variables .....	52
Declaring variables in C++ .....	52
Using variables .....	52
What if your program exits immediately? .....	53
Changing, using and comparing variables .....	54
Shorthand for adding and subtracting one .....	54
The use and misuse of variables .....	56
Common errors when declaring variables in C++ .....	56
Case sensitivity .....	57
Naming variables .....	57
Storing strings .....	58
Okay, I get strings—but why all those other types? .....	60
Quiz yourself .....	62
Practice problems .....	63
Chapter 4:    If Statements .....	64
Basic syntax for if .....	64
Expressions .....	65
What is truth? .....	65
The bool type .....	66
Else statements .....	67
Else-if .....	67
String comparisons .....	68
More interesting conditions using Boolean operators .....	68
Boolean not .....	69

Boolean and .....	69
Boolean or .....	70
Combining expressions .....	70
Example Boolean expressions.....	71
Quiz yourself .....	72
Practice problems .....	72
Chapter 5:    Loops .....	74
While loops .....	74
A common mistake .....	74
For loops .....	75
Variable initialization .....	76
Loop condition .....	76
Variable update.....	76
Do-while loops .....	77
Controlling the flow of loops .....	78
Nested loops .....	79
Choosing the right kind of loop.....	80
For loop .....	81
While loops .....	81
Do-while loops .....	81
Quiz yourself .....	82
Practice problems .....	83
Chapter 6:    Functions.....	84
Function syntax.....	84
Local variables and global variables.....	85
Local variables.....	85
Global variables.....	87
A warning about global variables.....	88
Making functions available for use .....	88
Function definitions and declarations .....	89
An example of using a function prototype .....	89
Breaking down a program into functions .....	90
When you're repeating code again and again .....	90
When you want to make code easier to read.....	90
Naming and overloading functions.....	91

Summary of functions.....	92
Quiz yourself .....	92
Practice problems .....	93
Chapter 7: What If You Can't Figure Out What to Do?.....	94
All we need to do is check if the number has no remainder when divided by the divisor: .....	96
A brief aside about efficiency and security.....	97
What if you don't know the algorithm?.....	98
Practice Problems .....	100
Chapter 8: Switch Case and Enums.....	101
Comparison of switch case with if-else.....	103
Creating simple types using enumerations.....	103
Quiz yourself .....	105
Practice problems .....	106
Chapter 9: Randomizing Your Programs.....	107
Getting random numbers in C++.....	107
Bugs and randomness.....	110
Quiz yourself .....	110
Practice problems .....	111
Part 2: Working with Data.....	112
Chapter 10: Arrays .....	113
Some basic array syntax.....	113
Example uses for arrays .....	114
Using arrays to store orderings.....	114
Representing grids with multidimensional arrays .....	114
Using arrays.....	115
Arrays and for loops.....	115
Passing arrays to functions .....	116
Writing off the end of an array .....	117
Sorting arrays .....	118
Quiz yourself .....	122
Practice problems .....	123
Chapter 11: Structures.....	124
Associating multiple values together.....	124
Syntax.....	124
Passing structures around.....	126

Quiz yourself .....	128
Practice problems .....	129
Chapter 12: Introduction to Pointers.....	130
Forget everything you've ever heard.....	130
Ok, then—what are pointers? Why should you care?.....	130
What is memory?.....	131
Variables vs. addresses .....	131
Memory layout.....	132
Other advantages (and disadvantages) of pointers.....	134
Quiz yourself .....	136
Practice problems .....	136
Chapter 13: Using Pointers .....	138
Pointer syntax .....	138
Declaring a pointer.....	138
Pointing to something: getting the address of a variable.....	138
Using a pointer .....	139
Uninitialized pointers and NULL .....	142
Pointers and functions .....	143
References .....	145
References vs. pointers.....	146
Quiz yourself .....	147
Practice problems .....	147
Chapter 14: Dynamic Memory Allocation.....	149
Getting more memory with new .....	149
Running out of memory .....	149
References and dynamic allocation .....	150
Pointers and arrays .....	150
Multidimensional arrays .....	152
Pointer arithmetic.....	153
Understanding two-dimensional arrays .....	154
Pointers to pointers .....	155
Pointers to pointers and two-dimensional arrays .....	157
Taking stock of pointers.....	158
Quiz yourself .....	158
Practice problems .....	159

Chapter 15: Introduction to Data Structures with Linked Lists .....	161
Pointers and structures.....	163
Creating a linked list.....	164
First time through .....	165
Second time through .....	165
Traversing a linked list .....	167
Taking stock of linked lists .....	168
Arrays vs. linked lists .....	169
Quiz yourself .....	171
Practice problems .....	172
Chapter 16: Recursion.....	173
How to think about recursion .....	173
Recursion and data structures .....	175
Loops and recursion.....	177
The stack .....	179
The power of the stack .....	181
Downsides of recursion.....	181
Debugging stack overflows .....	182
Performance .....	183
Taking stock of recursion .....	184
Quiz yourself .....	184
Practice problems .....	185
Chapter 17: Binary Trees .....	186
Talking about trees .....	188
Implementing binary trees.....	188
Inserting into the tree .....	189
Searching the tree.....	192
Destroying the tree .....	192
Removing from a tree .....	194
Real world use of binary trees .....	201
Cost of building trees and maps .....	203
Quiz yourself .....	203
Practice problems .....	204
Chapter 18: The Standard Template Library.....	205
Vectors, a resizable array.....	205

Calling methods on vectors.....	206
Other features of vectors.....	207
Maps .....	208
Iterators .....	209
Checking whether a value is in a map.....	211
Taking stock of the STL.....	212
Learning more about the STL.....	213
Quiz yourself .....	213
Practice problems .....	214
Chapter 19: More about Strings .....	215
Reading in strings.....	215
String length and accessing individual elements .....	216
Searching and substrings .....	217
Passing by reference .....	218
Const propagation.....	220
Const and the STL.....	221
Quiz yourself .....	222
Practice problems .....	223
Chapter 20: Debugging with Code::Blocks.....	224
Starting out .....	225
Breaking in .....	226
Debugging crashes .....	232
Breaking into a hung program .....	235
Modifying variables.....	239
Summary .....	239
Practice problems .....	239
Problem 1: Issues with exponents .....	239
Problem 2: Trouble adding numbers .....	240
Problem 3: Bugs with Fibonacci .....	240
Problem 4: Misreading and misprinting a list .....	241
Part 3: Writing Larger Programs .....	242
Chapter 21: Breaking Programs Up Into Smaller Pieces.....	243
Understanding the C++ build process.....	243
Preprocessing.....	243
Compilation.....	245



Linking .....	245
Why separate compiling and linking? .....	245
How to split your program across multiple files .....	246
Step 1: Splitting our declarations and definitions.....	246
Step 2: Figure out which functions need to be shared .....	246
Step 3: Move shared functions into their new files .....	247
Going through an example .....	247
Other dos and don'ts of header files .....	251
Handling multiple source files in your development environment .....	251
Quiz yourself .....	254
Practice problems .....	255
Chapter 22: Introduction to Program Design .....	256
Redundant code .....	256
Assumptions about how data is stored .....	257
Design and comments.....	258
Quiz yourself .....	259
Chapter 23: Hiding the Representation of Structured Data .....	261
Using functions to hide the layout of a structure .....	261
Method declaration and call syntax.....	262
Quiz yourself .....	264
Practice problems .....	265
Chapter 24: The Class.....	266
Hiding how data is stored .....	266
Declaring an instance of a class .....	268
The responsibilities of a class.....	268
What does private really mean? .....	269
Summary .....	270
Quiz yourself .....	270
Practice problems .....	270
Chapter 25: The Lifecycle of a Class.....	271
Object construction .....	271
What happens if you don't create a constructor? .....	273
Initializing members of the class.....	274
Using the initialization list for const fields .....	275
Object destruction .....	275

Destruction on delete .....	277
Destruction when going out of scope .....	277
Destruction due to another destructor .....	278
Copying classes .....	279
The assignment operator .....	280
The copy constructor .....	282
The full list of compiler generated methods.....	284
Preventing copying entirely .....	284
Quiz yourself .....	285
Practice problems .....	286
Chapter 26: Inheritance and Polymorphism.....	287
Inheritance in C++ .....	288
Other uses and misuses of inheritance.....	291
Inheritance, object construction and object destruction .....	292
Polymorphism and object destruction.....	294
The slicing problem .....	295
Sharing code with subclasses.....	297
Protected data .....	297
Class-wide data .....	298
How is polymorphism implemented?.....	299
Quiz yourself .....	301
Practice problems .....	302
Chapter 27: Namespaces .....	303
When to write "using namespace" .....	305
When should you create a namespace? .....	305
Quiz yourself .....	306
Practice problems .....	306
Chapter 28: File I/O.....	307
File I/O basics .....	307
Reading from files .....	307
File formats .....	309
End of file .....	310
Writing files.....	311
Creating new files.....	312
File position.....	312

Accepting command line arguments .....	315
Dealing with numeric command line arguments.....	317
Binary file I/O .....	317
Working with binary files .....	319
Converting to char* .....	319
An example of binary I/O .....	320
Storing classes in a file .....	321
Reading from a file .....	322
Quiz yourself .....	325
Practice problems .....	326
Chapter 29: Templates in C++ .....	328
Template functions .....	328
Type inference .....	329
Duck typing .....	330
Template classes .....	331
Tips for working with templates .....	332
Templates and header files .....	334
Summarizing templates .....	334
Diagnosing template error messages .....	334
Quiz yourself .....	338
Practice problems .....	339
Part 4: Miscellaneous Topics.....	340
Chapter 30: Formatting Output Using iomanip .....	341
Dealing with spacing issues.....	341
Setting the field width with setw .....	341
Changing the padding character .....	342
Permanently changing settings.....	342
Putting your knowledge of iomanip together .....	343
Printing numbers.....	344
Setting the precision of numerical output with setprecision .....	344
What do you do about money? .....	345
Output in different bases.....	345
Chapter 31: Exceptions and Error Reporting .....	347
Releasing resources during exceptions.....	348
Manual cleanup of resources in a catch block.....	349

Throwing exceptions.....	349
Throw specifications .....	351
Benefits of exceptions.....	352
Misuse of exceptions .....	352
Exceptions in summary .....	353
Chapter 32: Final Thoughts.....	355
Chapter 2 quiz solution .....	356
Chapter 3 quiz solution .....	357
Chapter 4 quiz solution .....	358
Chapter 5 quiz solution .....	359
Chapter 6 quiz solution .....	360
Chapter 8 quiz solution .....	361
Chapter 9 quiz solution .....	362
Chapter 10 quiz solution .....	363
Chapter 11 quiz solution .....	364
Chapter 12 quiz solution .....	365
Chapter 13 quiz solution .....	366
Chapter 14 quiz solution .....	367
Chapter 15 quiz solution .....	368
Chapter 16 quiz solution .....	369
Chapter 17 quiz solution .....	370
Chapter 18 quiz solution .....	371
Chapter 19 quiz solution .....	372
Chapter 21 quiz solution .....	373
Chapter 22 quiz solution .....	374
Chapter 23 quiz solution .....	375
Chapter 24 quiz solution .....	376
Chapter 25 quiz solution .....	377
Chapter 26 quiz solution .....	379
Chapter 27 quiz solution .....	381
Chapter 28 quiz solution .....	382
Chapter 29 quiz solution .....	383

# Part 1: Jumping into C++

---

Let's get ready to program! Programming, like other art forms, allows you to create—but in programming, your power is multiplied by the speed and capabilities of the computer. You can create engaging games like World of Warcraft, Bioshock, Gears of War and Mass Effect. You can create detailed and immersive simulations like The Sims. You can write programs that connect people together: web browsers like Chrome, email editors or chat clients, or websites like Facebook or Amazon.com. You can build apps that delight your users, taking advantage of new devices like iPhones or Android phones. Those things, of course, take time to become skilled enough to create. But even in the beginning you can write interesting software—programs that solve your math homework for you, simple games like Tetris that you can show your friends, tools to automate tedious chores or complex calculations that would otherwise take days or weeks by hand. Once you understand the basics of programming a computer—which this book will teach you—you'll have the ability to pick up the graphics or networking libraries you need to in order to write the kinds of programs that interest you, whether they're games, scientific simulations or something in between.

C++ is a powerful programming language that will give you a strong grounding in modern programming techniques. In fact, C++ shares concepts with many other languages, so much of what you learn will transfer to other languages that you pick up later (almost no programmer works with a single language exclusively).

C++ programmers have a flexible skill set, with the ability to work on many different projects. Most of the applications and programs you use every day were written in C++. Incredibly, every one of these applications I listed earlier was either written entirely in C++ or has significant components written in C++.<sup>1</sup>

In fact, interest in C++ continues to grow even as new programming languages such as Java and C# gain popularity. I've seen a marked increase in traffic to my site, Cprogramming.com, over the last few years. C++ continues to be the language of choice for high performance applications, creating programs that run extremely fast, often faster than Java or similar languages. C++ continues to grow as a language, with a new language specification, C++11, adding new features that make it easier and faster to use as a developer while maintaining its high-performance roots.<sup>2</sup> A strong knowledge of C++ is also valuable on the job market, and jobs that require C++ skill are often both challenging and high-paying.

Are you ready to get started? Part 1 is all about getting you set up to start writing programs and getting you using the basic building blocks of C++. Once you're done with this section, you'll be able to write real programs that you can show your friends (your close and nice friends, anyway) and you'll understand how to think like a programmer. You won't be a C++ master, but you'll be well-prepared to learn the remaining language features that you'll need to make really useful and powerful programs.

I'll also give you just enough background and terminology to stay afloat, putting off the more complicated explanations for certain things until you've got the basics.

---

<sup>1</sup> You can find these applications, and many more uses of C++, at <http://www.stroustrup.com/applications.html>

<sup>2</sup> This specification was ratified as this book neared completion, so I have not included any material from the new standard. You can find a series of articles introducing C++11 at <http://www.cprogramming.com/c++11/what-is-c++0x.html>

The other parts of this book will introduce you to increasingly advanced concepts. You'll learn how to write programs that work with large amounts of data, including taking input from files and learning how to process that data easily and efficiently (and learn numerous shortcuts along the way). You'll learn how to write larger, more complex programs without getting lost under a wave of complexity. You'll also learn about the tools that are used by professional programmers.

By the end of this book, you should be able to read and write real computer programs that do useful, interesting things. If you're interested in game programming, you'll be ready to take up the challenges specific to game programming. If you're taking, or preparing to take, a class on C++, you should have the information you need to survive and thrive. If you're a self-learner, you should have enough information to write just about any program you're interested in writing, having nearly all of the tools provided by C++ at the ready.

## Acknowledgements

I'd like to thank Alexandra Hoffer for her careful, patient editing and detailed suggestions throughout the production of this book. Without her efforts, this book would not exist. I'd also like to thank Andrei Cheremskoy, Minyang Jiang and Johannes Peter for invaluable feedback, suggestions and corrections.

## Chapter 1: Introduction and Developer Environment Setup

### What is a programming language?

When you want to control your computer, you need a way to speak to it. Unlike your dog or your cat, which have their own inscrutable languages, computers have programming languages created by people. A computer program is a piece of text—like a book, or an essay—but with its own particular structure. The language, while comprehensible to humans, is more strictly structured than a normal language, and the vocabulary is much smaller. C++ is one of these languages, and a popular one at that.

Once you have written a computer program, you need a way for the computer to run it—to interpret what you've written. This is usually called executing your program. The way you do this will depend on your programming language and environment—we'll talk more about how to execute your program soon.

There are many programming languages, each with their own different structure and vocabulary, but they are in many ways very similar. Once you have learned one, learning the next will be easier.

### I've heard of a language called C, what's the difference between C and C++?

C is a programming language originally developed for developing the Unix operating system. It is a low-level and powerful language, but it lacks many modern and useful constructs. C++ is a newer language, based on C, which adds many more modern programming language features that make it easier to program than C.

C++ maintains all the power of the C language, while providing new features to programmers that make it easier to write useful and sophisticated programs.

For example, C++ makes it easier to manage memory and adds several features to allow "object-oriented" programming and "generic" programming. We'll talk about what that really means later. For now, just know that C++ makes it easier for programmers to stop thinking about the nitty-gritty details of how the machine works and think about the problems they are trying to solve.

If you're trying to decide between learning C and C++, I strongly suggest starting with C++.

### Do I need to know C to learn C++?

No. C++ is a superset of C; anything you can do in C, you can do in C++. If you already know C, you will easily adapt to the object-oriented features of C++. If you don't know C, that's ok—there's no real advantage to learning C before C++, and you will be able to immediately take advantage of powerful C++-only features (the first among many being easier input and output).

### Do I need to know math to be a programmer?

If I had a nickel for every time someone asked me this, I'd need a calculator to count my small fortune. Fortunately, the answer is, emphatically, No! Most of programming is about design and logical reasoning, not about being able to quickly perform arithmetic, or deeply understanding algebra or calculus. The overlaps between math and programming are primarily around logical reasoning and precise thinking. Only if you want to program advanced [3D graphics engines](#), write programs to perform statistical analysis or do other specialized numerical programming will you need mathematical skill.

## Terminology

Throughout the book, I'll be defining new terms, but let's get started with some very basic concepts that you'll need to get started.

## Programming

Programming is the act of writing instructions in a way that allows a computer to understand and execute those instructions. The instructions themselves are called **source code**. That's what you'll be writing. We'll see some source code for the very first time in a few pages.

## Executable

The end result of programming is that you have an **executable** file. An executable is a file that your computer can run—if you're on Windows, you'll know these files as EXEs. A computer program like Microsoft Word is an executable. Some programs have additional files (graphics files, music files, etc.) but every program requires an executable file. To make an executable, you need a **compiler**, which is a program that turns source code into an executable. Without a compiler, you won't be able to do anything except look at your source code. Since that gets boring quickly, the very next thing we will do is set you up with a compiler.

## Editing and compiling source files

The rest of this chapter is devoted to getting you set up with a simple, easy-to-use development environment. I'll get you set up with two specific tools, a compiler and an **editor**. You've already learned why you need a compiler—to make the program do stuff. The editor is less obvious, but equally important: an editor makes it possible for you to create source code in the right format.

Source code must be written in a **plain text** format. Plain text files contain nothing but the text of the file; there is no additional information about how to format or display the content. In contrast, a file you produce using Microsoft Word (or similar products) is not a plain text file because it contains information about the fonts used, the size of the text, and how you've formatted the text. You don't see this information when you open the file in Word, but it's all there. Plain text files have just the raw text, and you can create them using the tools we're about to discuss.

The editor will also give you two other nice features, **syntax highlighting** and **auto-indentation**. Syntax highlighting just means it adds color coding so that you can easily tell apart different elements of a program. Auto-indentation means that it will help you format your code in a readable way.

If you're using Windows or a Mac, I'll get you set up with a sophisticated editor, known as an **integrated development environment** (IDE) that combines an editor with a compiler. If you're using Linux, we'll use an easy-to-use editor known as nano. I'll explain everything you need in order to get set up and working!

## A note about sample source code

This book includes extensive sample source code, all of which is made available for you to use, without restriction but also without warranty, for your own programs. The sample code is included in `sample_code.zip`, which came with this book. All sample source code files are stored in a separate folder named after the chapter in which that source file appears (e.g. files from this chapter appear in the folder `ch1`). Each source code listing in this book that has an associated file has the name (but not the chapter) of the file as a caption.



## Windows

We'll set up a tool called **Code::Blocks**, a free development environment for C++.

### Step 1: Download Code::Blocks

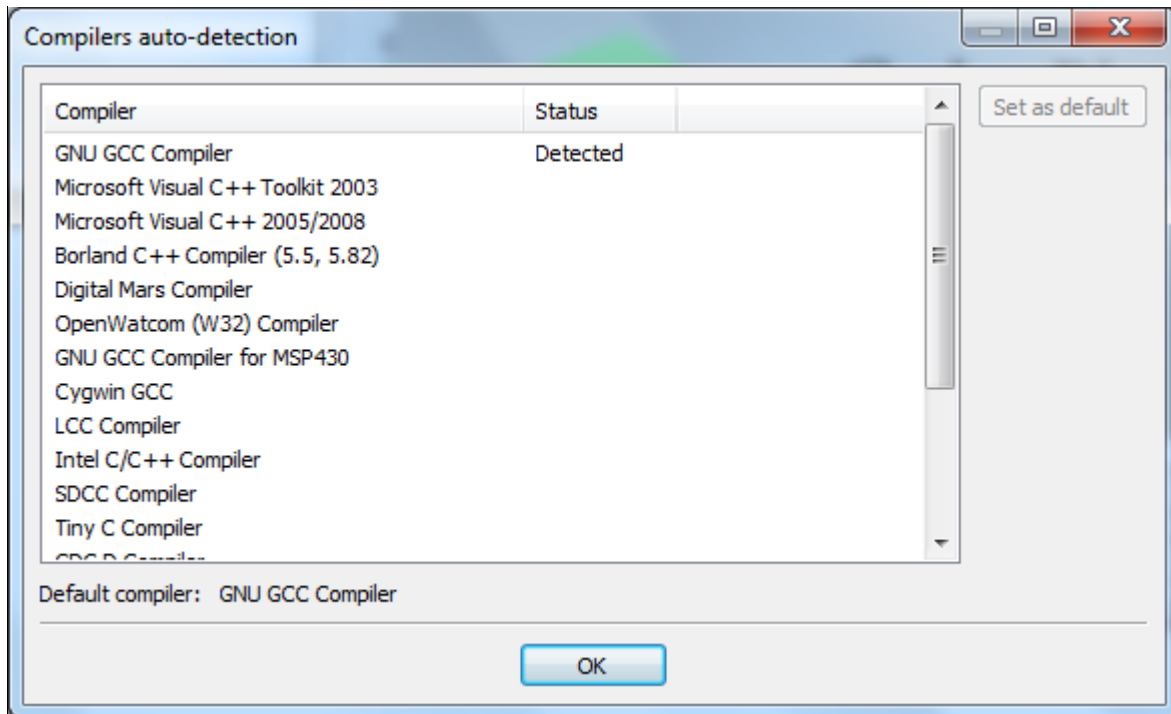
- Go to this website: <http://www.codeblocks.org/downloads>
- Follow the link to "Download the binary release" ([direct link](#))
- Go to the Windows 2000 / XP / Vista / 7 section
- Look for the file that includes mingw in the name. (The name as of this writing was codeblocks-10.05mingw-setup.exe; the number may be different).
- Save the file to your desktop. As of this writing, It is roughly 74 megabytes.

### Step 2: Install Code::Blocks

- Double click the installer.
- Hit next several times. Other setup tutorials will assume you have installed in **C:\Program Files\CodeBlocks** (the default install location), but you may install elsewhere if you like
- Do a Full Installation (select "Full: All plugins, all tools, just everything" from the "Select the type of install" dropdown menu)
- Launch Code::Blocks

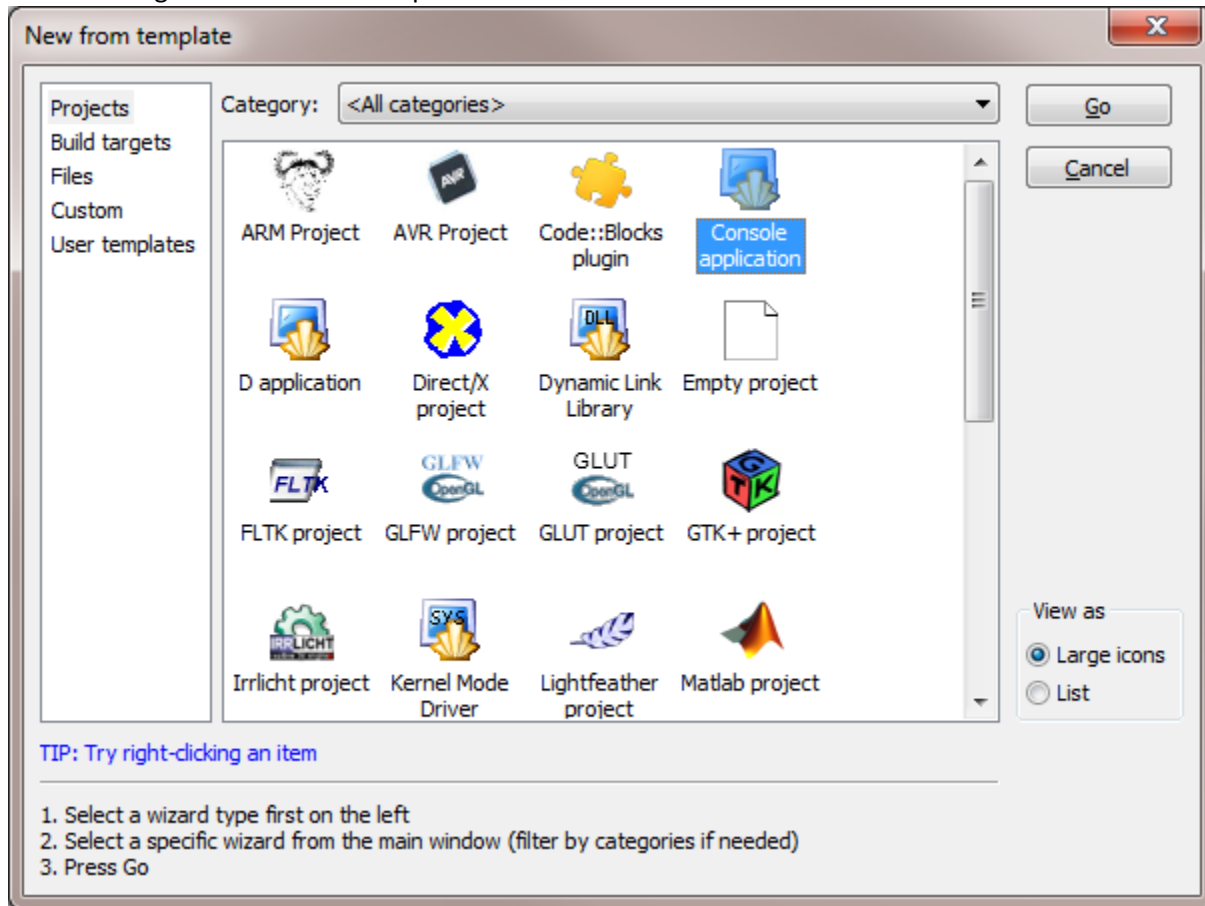
### Step 3: Running Code::Blocks

You will be prompted with a Compilers auto-detection window:



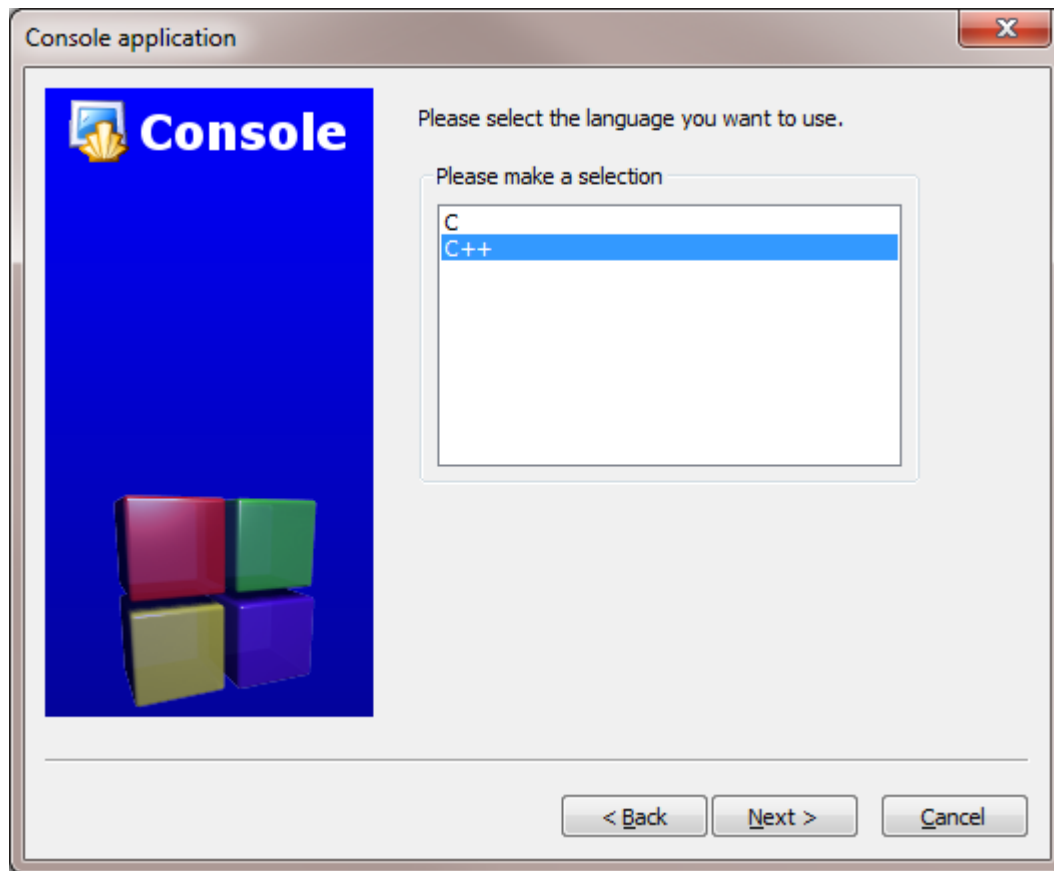
When you get the compilers auto-detection window, just hit OK. Code::Blocks may ask whether you want to associate it as the default viewer for C/C++ files—I suggest you do. Click on the File menu, and under "New", select "Project..."

The following window will come up:



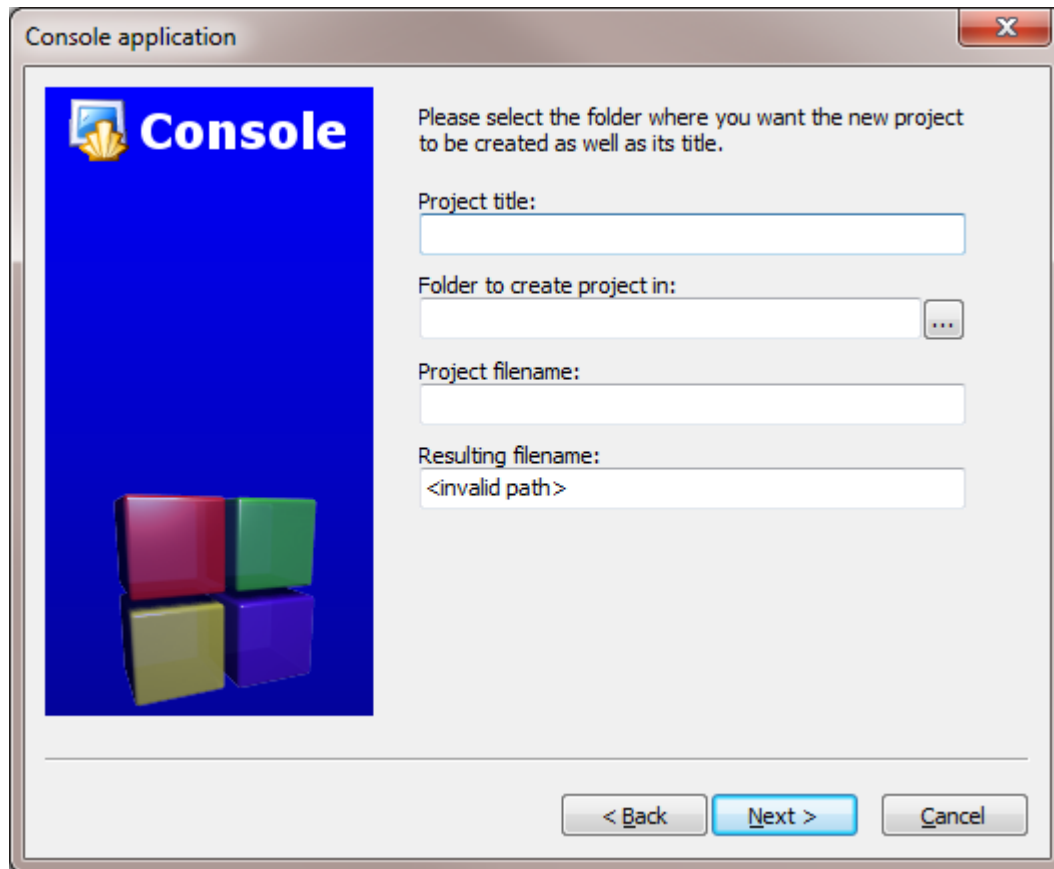
Click on "Console Application" and hit the "Go" button. All sample code from this book can be run as a console application.

Click next until you get to the language selection dialog:



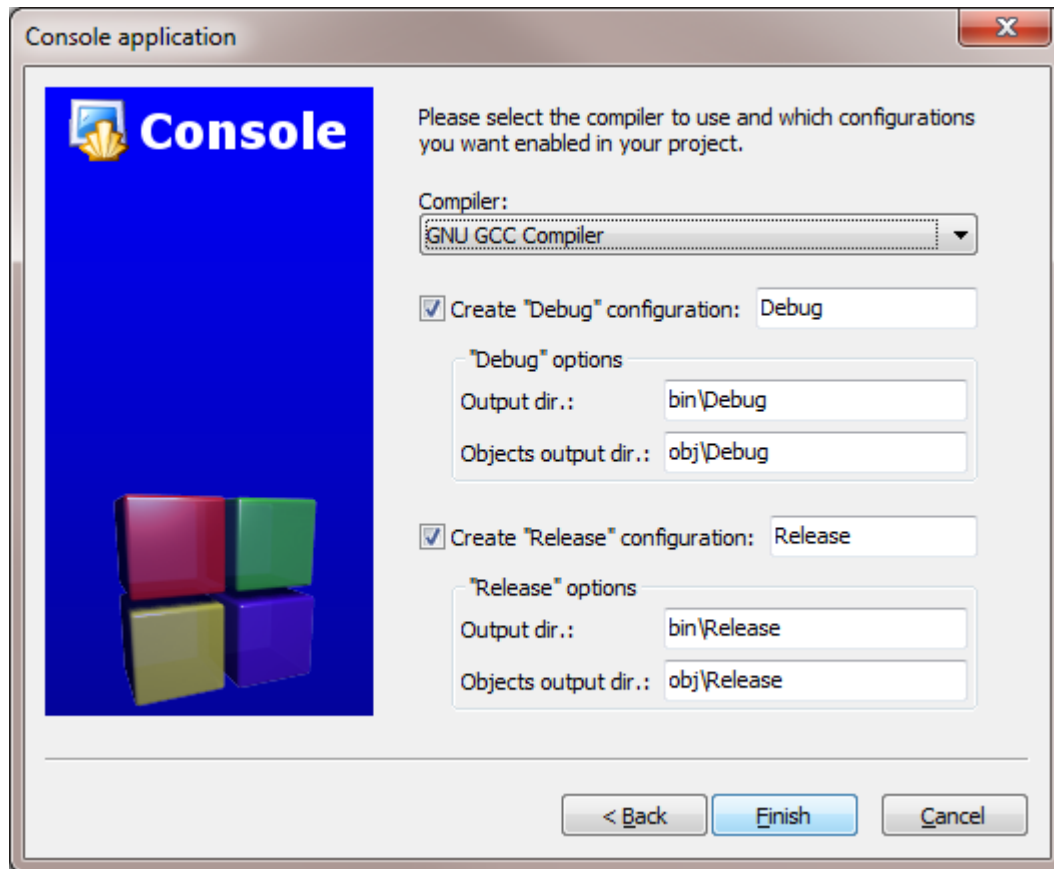
You'll be asked to choose whether you want to use C or C++. Since we're learning C++, pick C++.

After clicking "Next", Code::Blocks will then prompt you with where you'd like to save the console application:



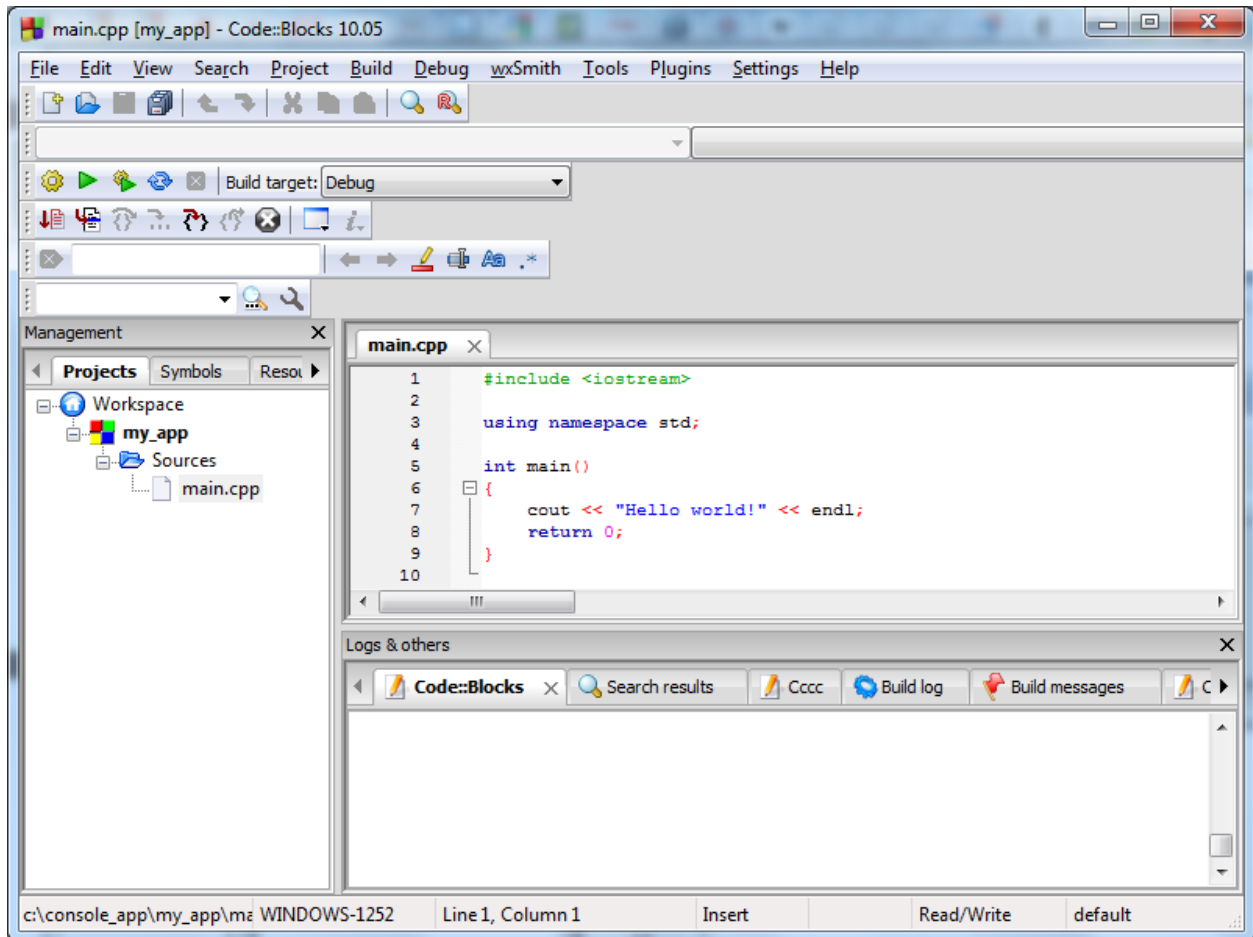
I'd recommend you put it in its own folder, as it may create several files (this is especially true if you create other types of projects). You will need to give your project a name; anything will be fine.

Clicking "Next" again will prompt you to set up your compiler:



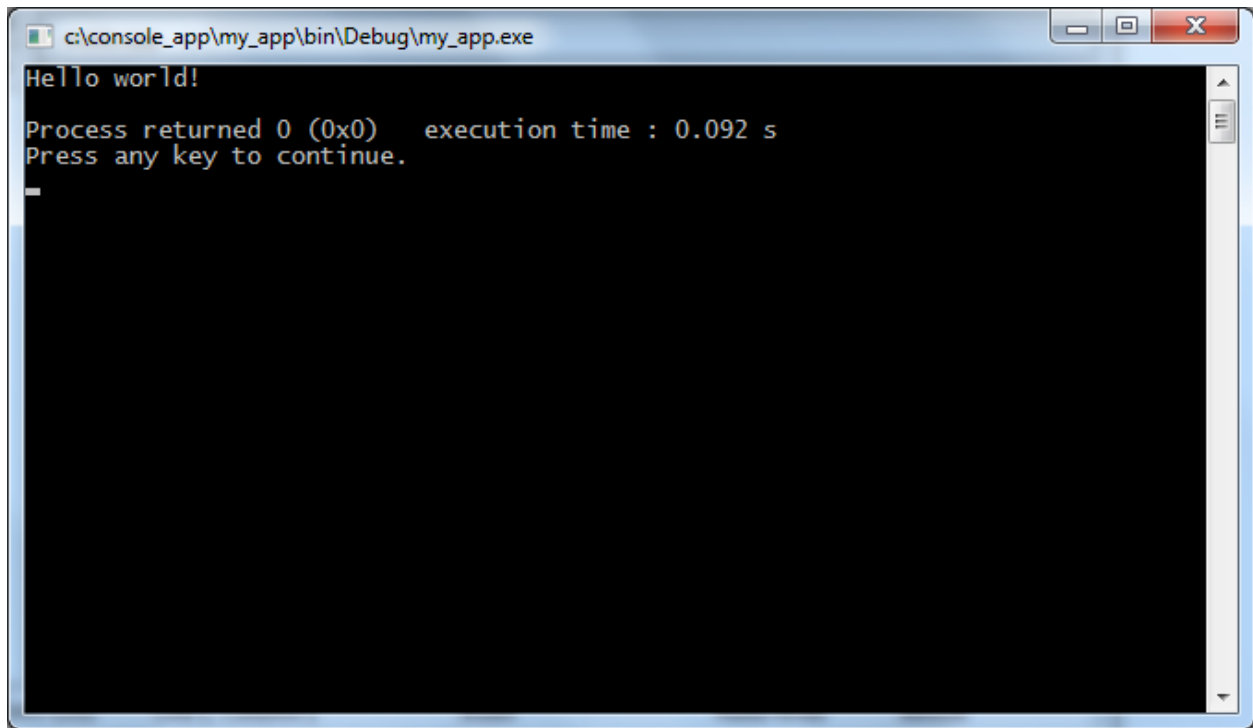
You don't need to do anything here. Just accept the defaults by hitting "Finish".

You can now open the main.cpp file on the left:



(You may need to expand the contents of the "Sources" folder if you don't see main.cpp.)

At this point, you will have your main.cpp file, which you can modify if you like. Notice the file extension: .cpp is the standard extension for C++ source files—not .txt—even though cpp files are plain text. For now, it just says "Hello World!", so we can run it as is. Hit F9, which will first compile it and then run it. (You can also go to the Build|Build and Run menu option.)



You now have a running program! You can simply edit main.cpp and then hit F9 to compile it and run it again.

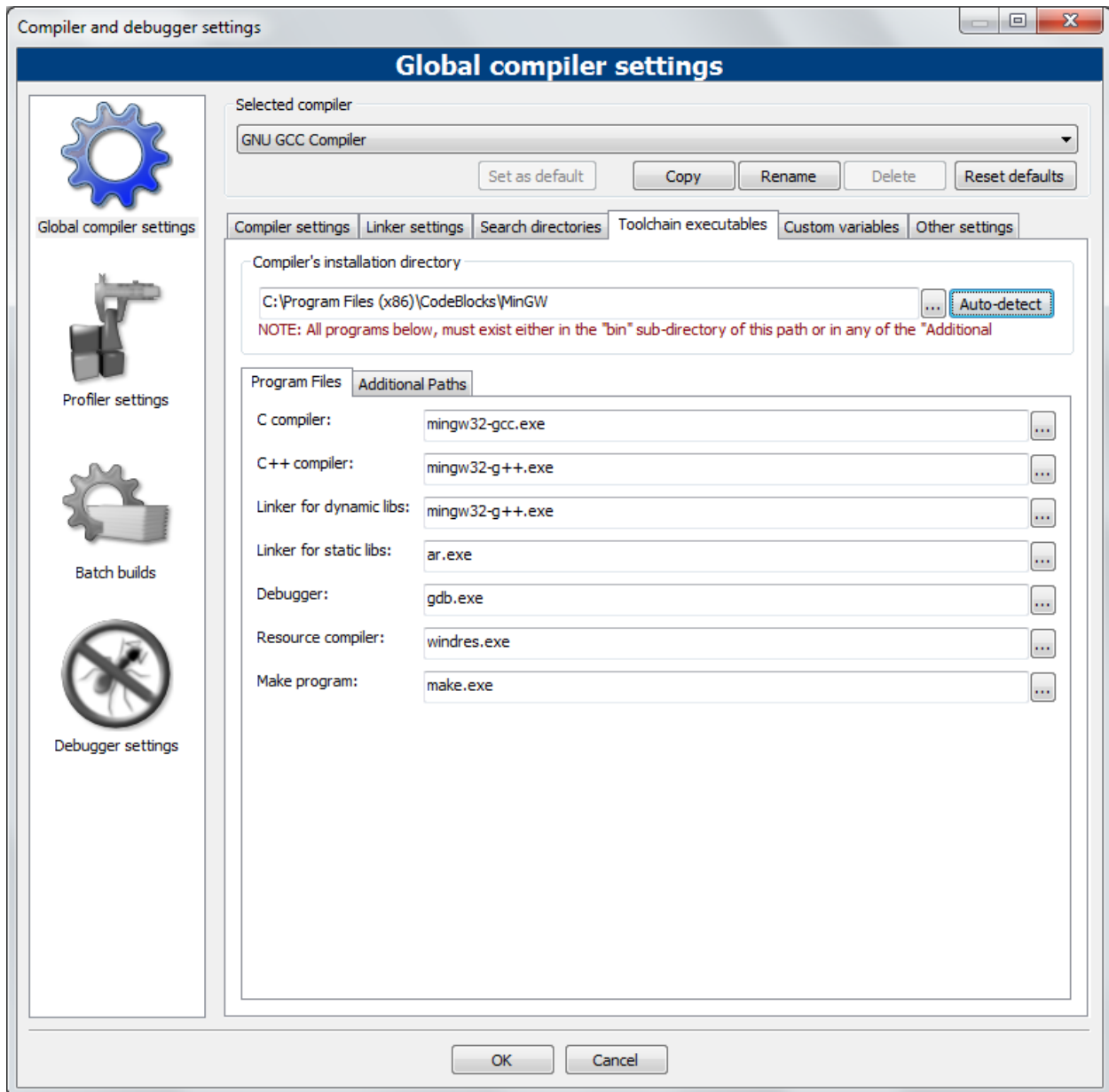
### Troubleshooting

If for some reason you don't get a running program, it probably means that there were compiler errors or that the environment wasn't set up correctly.

### Environment Setup

The most common error people see if things don't work is a message like "'CB01 – Debug" uses an invalid compiler. Probably the toolchain path within the compiler options is not setup correctly?! Skipping..."

First, make sure that you downloaded the right version of Code::Blocks, the one that included MinGW. If that doesn't solve the problem, it is likely a problem with compiler auto-detection. To check your current "auto-detected" state, go to "Settings|Compiler and Debugger...". Then on the left, choose "Global Compiler Settings" (it has a gear icon) and on the right, select the "Toolchain executables" tab. This tab has an "Auto-detect" button that you can use. That might fix the problem—if it doesn't, you can manually fill out the form. Here's a screenshot demonstrating what things look like on my system. Change the path marked "Compiler's installation directory" if you installed to a different location, and make sure everything else is filled in as shown.

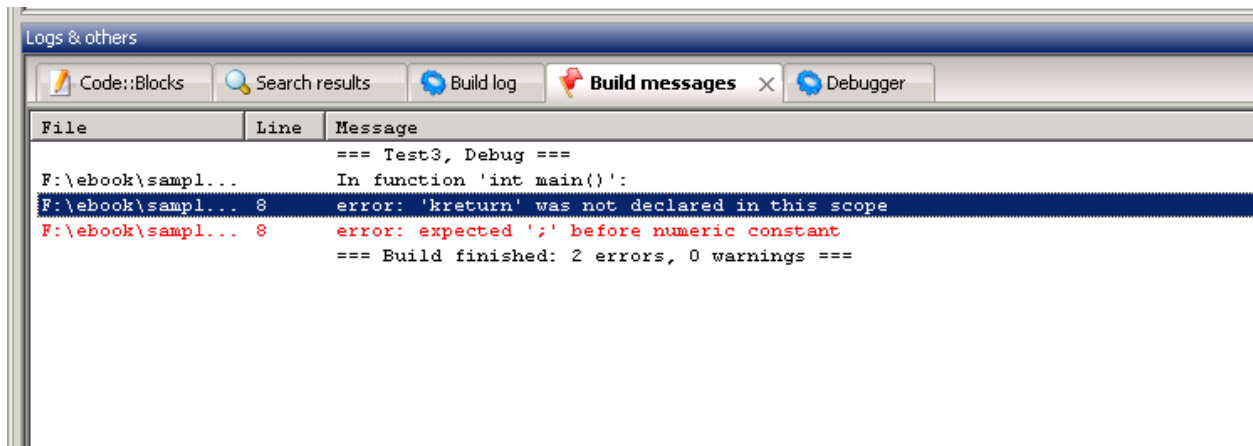


Once you've done that, try pressing F9 again to see if you get a running program.

### Compiler Errors

Compiler errors could happen if you've modified the main.cpp file in a way that confuses the compiler. To figure out what is wrong, take a look at the "Build messages" or "Build log" windows. The "Build messages" window will show you just compiler errors, the "Build log" will show you other issues too. Here's what it will look like if you have an error:





In this case, it shows you the name of the file, the line number, and then a brief string of text explaining the error. Here, I changed the line `return 0;` to be `kreturn 0;` and that is not valid C++, so I got an error.

Whenever you are programming, you will find it useful to check this window when your program doesn't compile in order to figure out what happened.

Throughout this book, you will see lots of sample code. For each one, you can either create a new console application or modify the source file of your original program. I'd recommend making a new console application for each program so that you can make changes to the sample code and save it for later review.

### What exactly is Code::Blocks?

Earlier, I introduced the idea of an integrated development environment. Code::Blocks is an integrated development environment because it makes it easy to write source code and build your program from the same application. One thing you should be aware of is that Code::Blocks itself is not a compiler. When you downloaded Code::Blocks, the installation package you chose *also* included a compiler, in this case GCC from [MinGW](http://www.mingw.org/), which is a free compiler for Windows. Code::Blocks handles all the messy details of setting up and calling the compiler, which is doing the real work.

### Macintosh

This section covers only setting up development on an OS X system.<sup>3</sup>

OS X already comes with a powerful Unix-based shell environment that you can use, so many of the tools that are covered in the Linux section of this book are available to you. However, you may also want to try out Apple's Xcode development environment. Regardless of whether you choose to use the Xcode environment itself, installing Xcode is a prerequisite to using the standard Linux tools as well.

While using the Xcode environment itself is not required for developing C++ programs on the Mac, if you want to venture into Mac UI programming, then you should learn to use Xcode.

<sup>3</sup> If you're using Mac OS 9 or earlier, and are unable to upgrade, you can try the Macintosh Programmer's Workshop, available directly from Apple: <http://developer.apple.com/tools/mpw-tools/> Since OS 9 is so old, I cannot walk you through the setup.

## Xcode

Xcode comes for free as part of Mac OS X, but by default, Xcode is not actually installed. You can either find Xcode on your Mac OS X DVD, or download the latest version. The download that includes documentation is very large, so you should try to find Xcode on your Mac OS X CD if you have a slow network connection. Note that even the basic compilers, such as gcc and g++, which you normally have installed by default on a Linux environment, are not installed by default on Mac OS X; to get them, you must download Xcode Developer Tools.

## Installing Xcode

To download Xcode:

- Register as an Apple developer at <http://developer.apple.com/programs/register/>
- Registering as an Apple developer is free. The Apple website may make it seem like you have to pay, but the link above should take you directly to the free signup page. You will have to fill out some basic personal information as part of signing up.
- If you are using Lion or a later OS version (10.7.x or later) go to <https://developer.apple.com/downloads/index.action> and search for Xcode. You should see the latest version of Xcode; click on it. A number of links will show up, pick the one that's just called "Xcode" followed by the version number. You can also download Xcode in the App store instead.
- If you are using Snow Leopard (OS 10.6.x), the version of Xcode above will not work for you. You will need to go to <https://developer.apple.com/downloads/index.action> and search for Xcode 3.2.6. One result will show up; click on it and then click the download link, Xcode 3.2.6 and iOS SDK 4.3.

Xcode comes as a standard disk image file that you can open. Open this disk image, and run the file Xcode.mpkg.

The installation process will ask you to agree to a licensing agreement, and then present you with a list of components to install. The default components should be fine. Go ahead and accept all the defaults and run the rest of the installer.

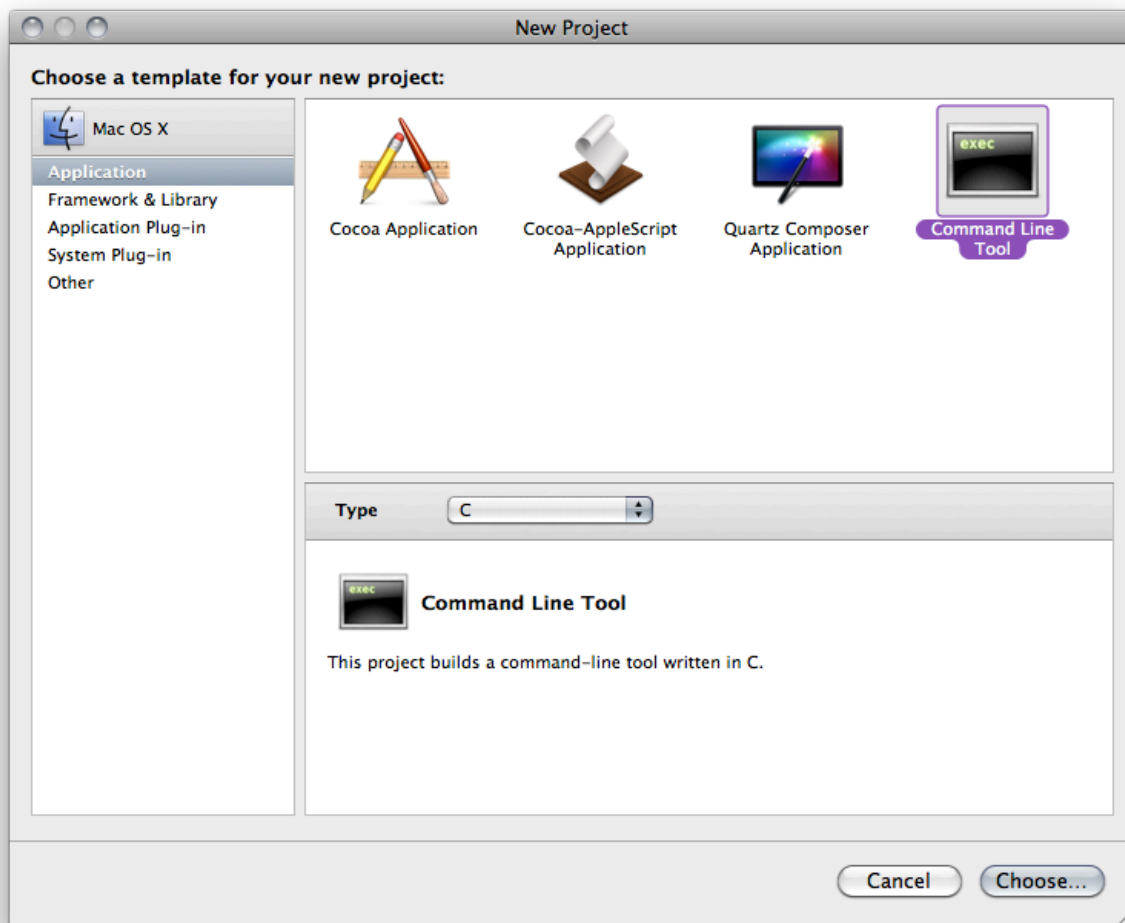
## Running Xcode

Once you've run the installer, you can find Xcode in Developer|Applications|Xcode. Go ahead and run the Xcode application. Xcode comes with extensive documentation, and you may wish to take some time and go through the "Getting Started with Xcode" tutorial. However, the rest of this section will not assume that you have read any other documentation.

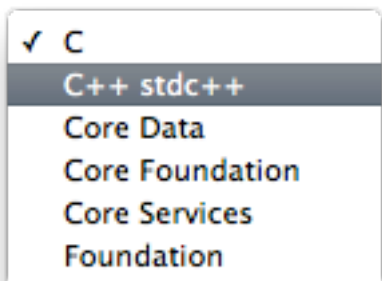
## Creating your first C++ program in Xcode

So let's get started—from the main Xcode window that comes up when you start Xcode, choose "Create a new Xcode project". (You can also go to "File|New Project..." or press Shift-⌘-N).

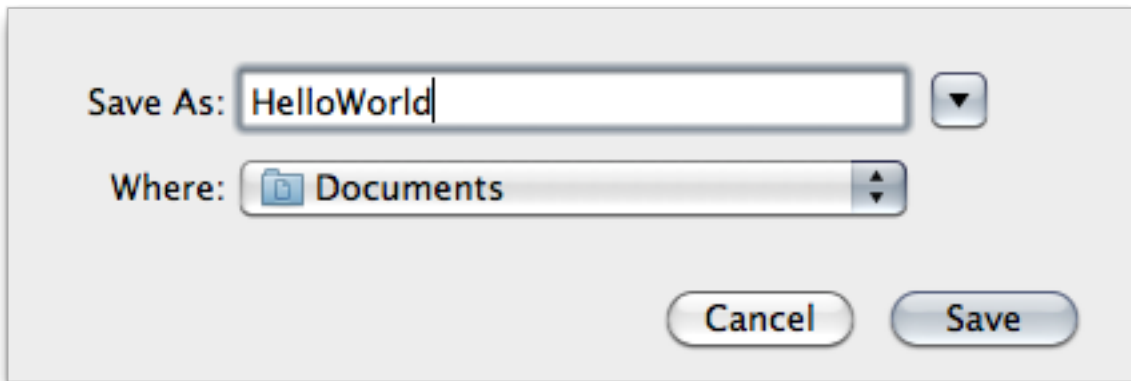
Choose “Application” from the left sidebar under “Mac OS X”, and then choose “Command Line Tool”. (You may also see “Application” under iOS—you don’t want that right now.)



You will also need to change the “Type” of the project from C to C++ stdc++.

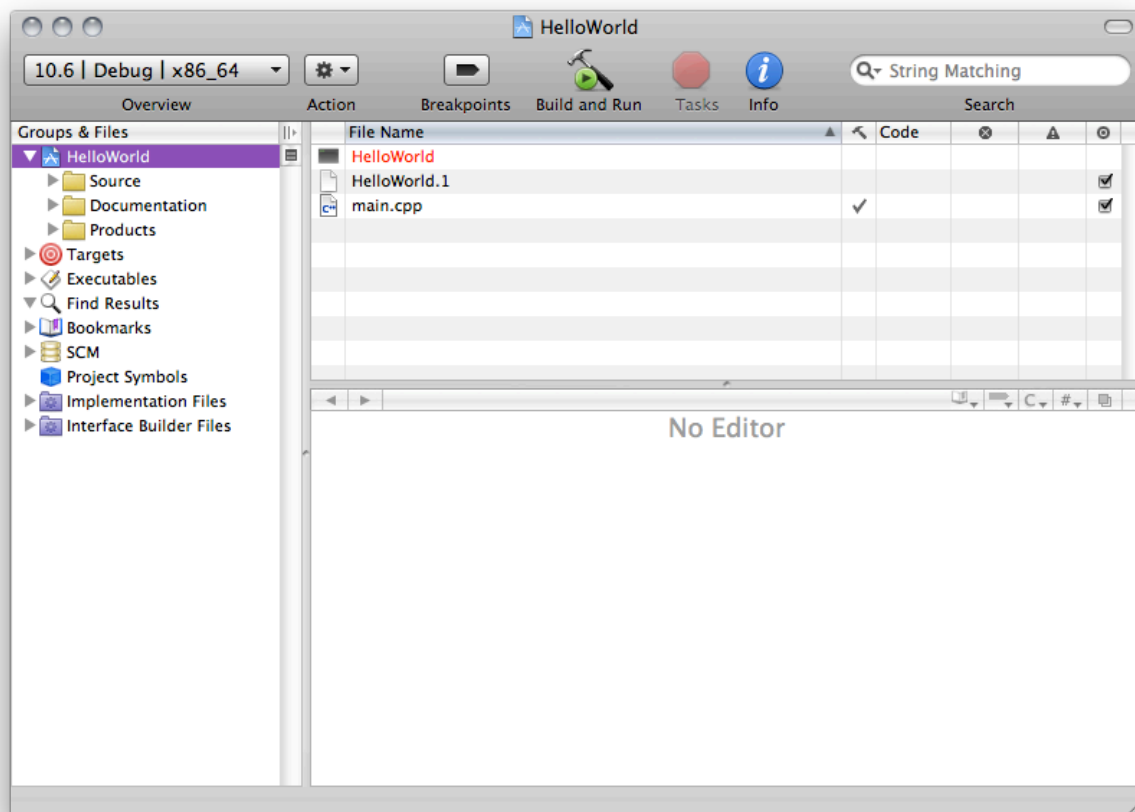


Once you’ve done that, press “Choose...” and select a name and a location for your new project. This will create a new directory under the location that you choose, with the same name as the name of your project. For this sample program, I will use the project name HelloWorld.



Then press Save.

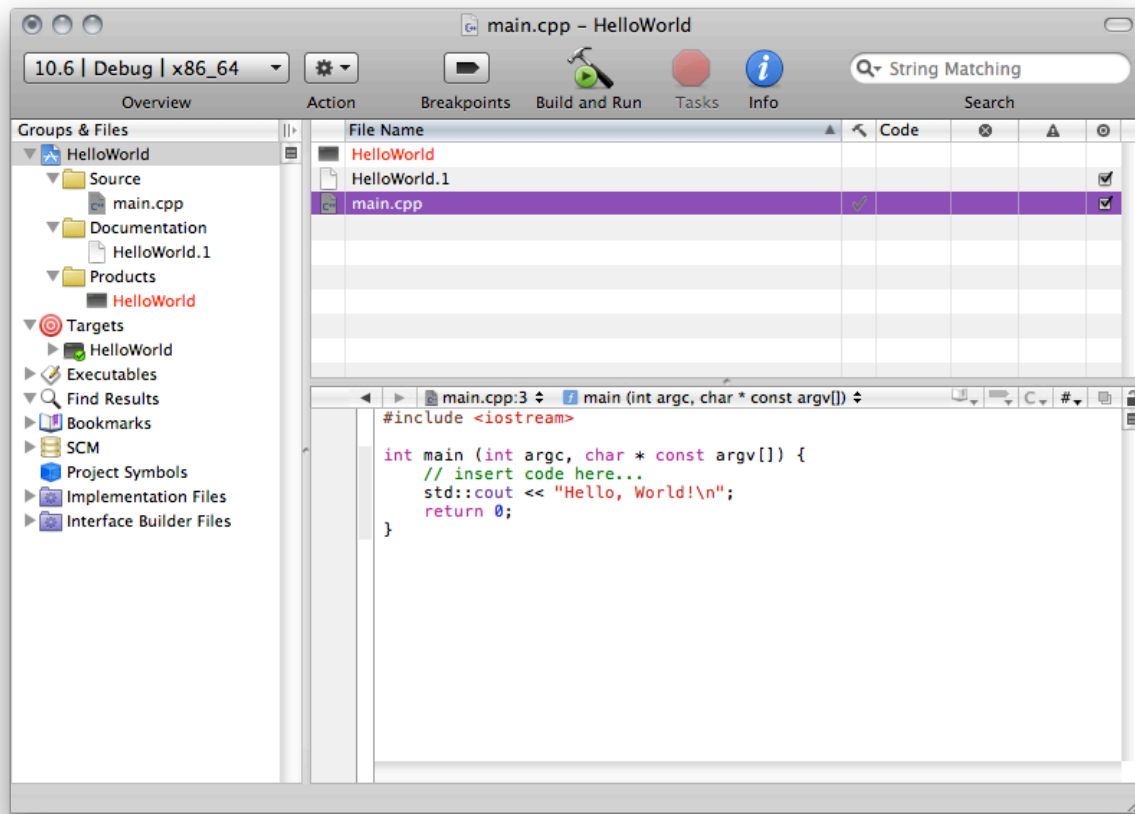
After pressing save, a new window will come up that looks like this:



This view shows you quite a few things. The right sidebar gives you access to Source, Documentation and Products. The “Source” folder contains the actual C++ files associated with your project, the “Documentation” folder contains any documentation you have—usually the source for a “man page”. You can ignore it for now. The “Products” folder stores the result of compiling your program. You can

also see the contents of these folders displayed in the top middle window.

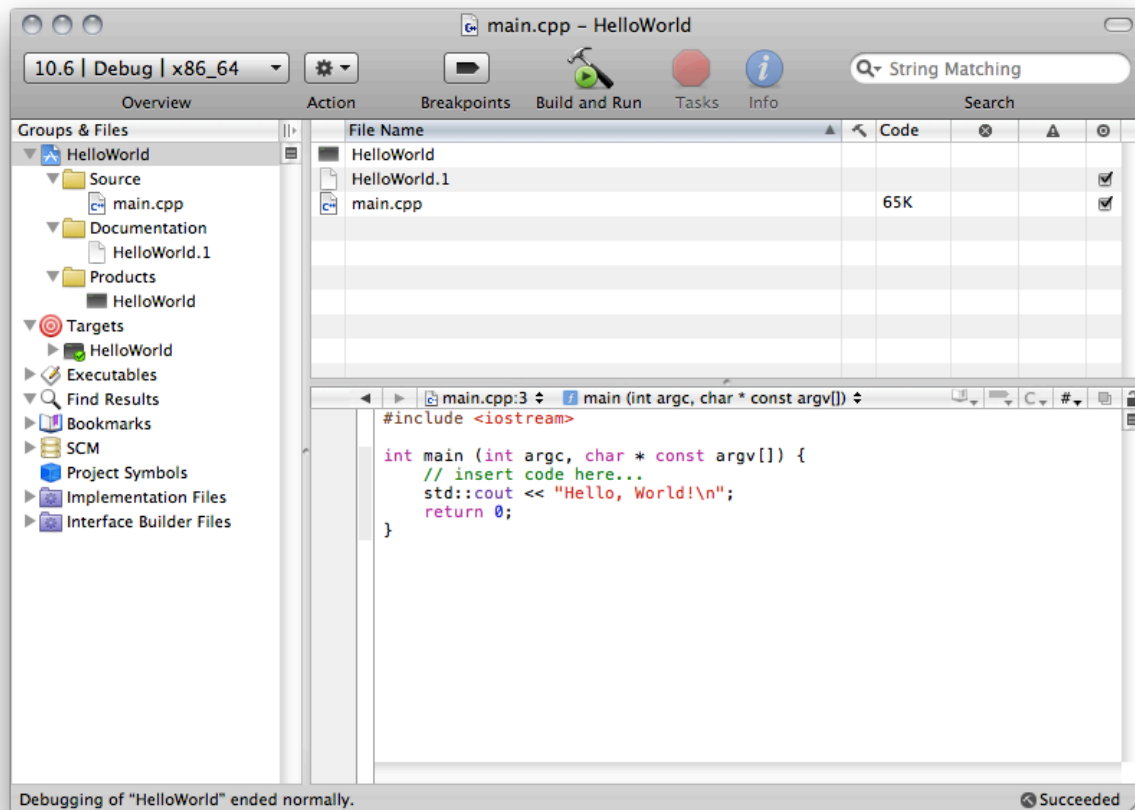
For now, let's work on the source file itself. Go ahead and select "main.cpp" either from the top middle window or from the Source folder on the left. (Notice the file extension: .cpp is the standard extension for C++ source files—not .txt—even though cpp files are plain text.) If you single-click you will bring up the source in the window that currently reads "No Editor". You can then start typing directly into the file.



You can also double-click on the file in order to bring up a larger editor window, if you want more space.

By default Xcode provides a small sample program that you can start with. Let's compile and then run this sample program. First click on the "Build and Run" button on the toolbar.

When you press this button, the program will compile, meaning that the executable file will be created. In Xcode 3, you won't actually see anything run. In order to do that, you need to double-click on the "HelloWorld" executable. You'll notice that it used to be colored red, but after doing the build it should be colored black:



Go ahead and double click it to run your first program!

You should see some output that looks something like this (I've covered the username for the privacy of the person who lent me their Macintosh for this screenshot):



```
Terminal — 80x24
[redacted]-macbook-pro:~ [redacted]$ /Users/[redacted]/Documents/HelloWorld/buil
d/Debug/HelloWorld ; exit;
Hello, World!
logout

[Process completed]
```

And there you go—you've run your first program!

From here on out, whenever you have a sample program you want to run, you can use the project we just created, or you can create a new project for it. In either case, when you want to add your own code, you can start by modifying the sample program that Xcode creates in `main.cpp`.

### Installing Xcode 4

To download Xcode 4, you can simply search for it in the Mac App Store and install it. It is about 4.5 GB. The download from the Mac App Store will put an "Install Xcode" icon into your Dock. Run this to start the install process.

The installation process will ask you to agree to a licensing agreement, and then present you with a list of components to install. The default components should be fine. Go ahead and accept all the defaults and run the rest of the installer.

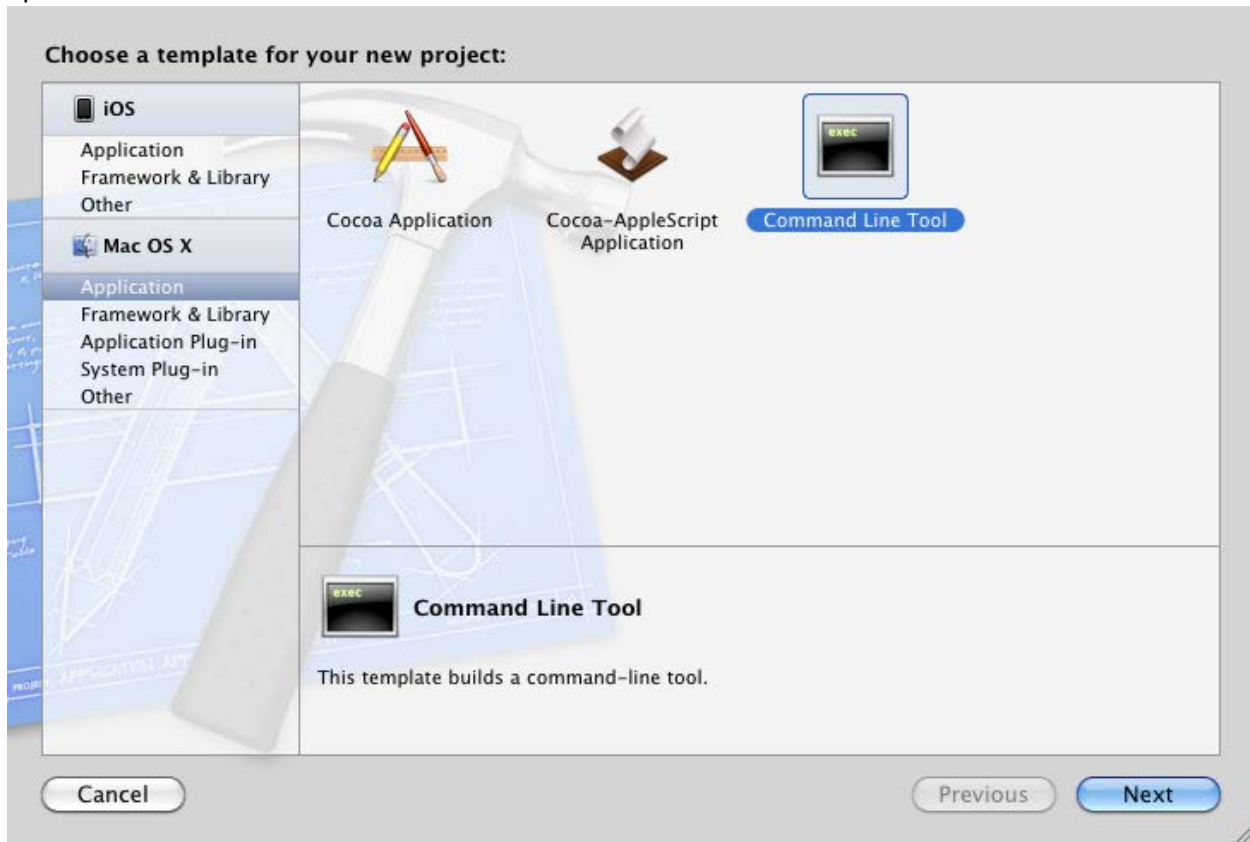
### Running Xcode

Once you've run the installer, you can find Xcode in Developer|Applications|Xcode. Go ahead and run the Xcode application. Xcode comes with extensive documentation, and you may wish to take some

time and go through the “Xcode Quick Start Guide”, which you can reach from the “Learn about using Xcode” link on the startup screen. However, the rest of this section will not assume that you have read any other documentation.

### Creating your first C++ program in Xcode

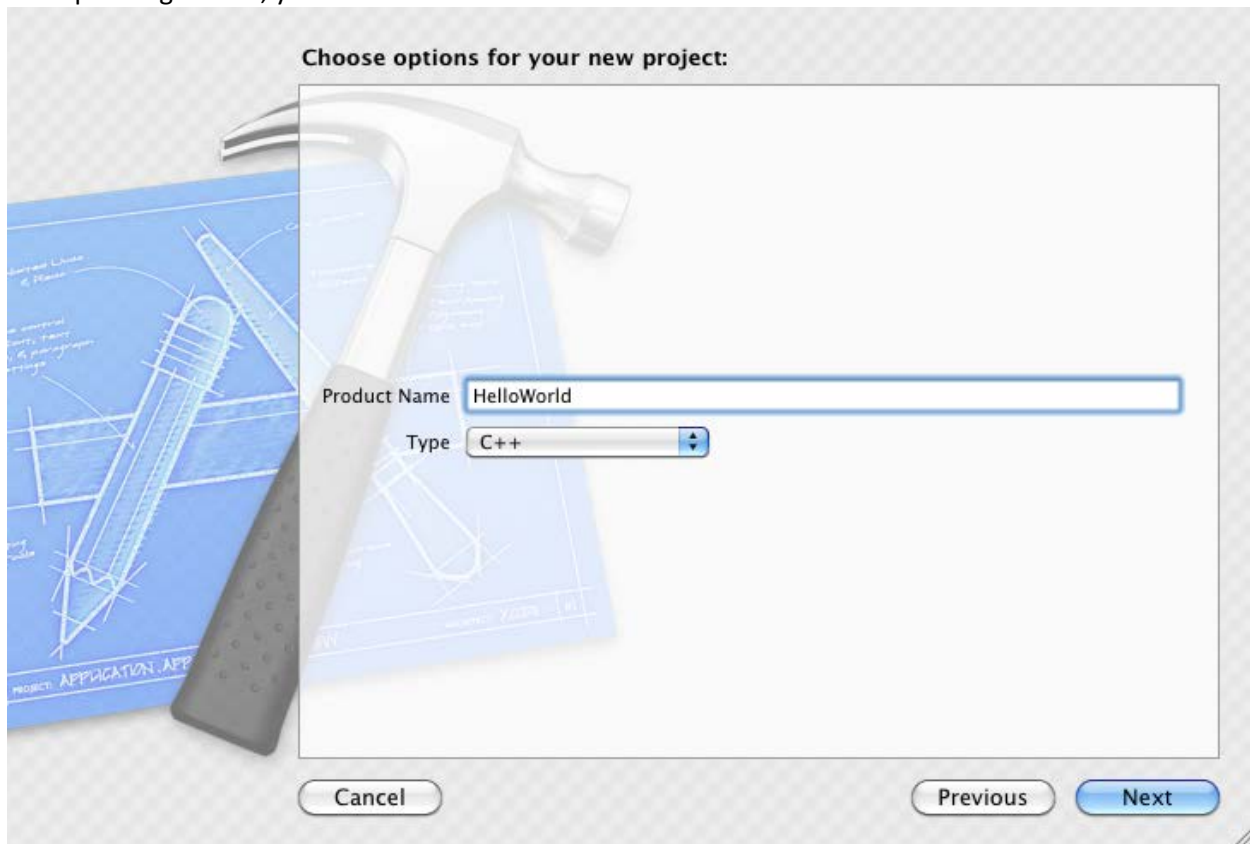
So let’s get started—from the main Xcode window that comes up when you start Xcode, choose “Create a new Xcode project”. (You can also go to “File|New|New Project...” or press Shift-⌘-N). This will bring up a screen that looks like this.



Choose “Application” from the left sidebar under “Mac OS X”, and then choose “Command Line Tool”. (You may also see “Application” under iOS—you don’t want that right now.) Then press “Next”.

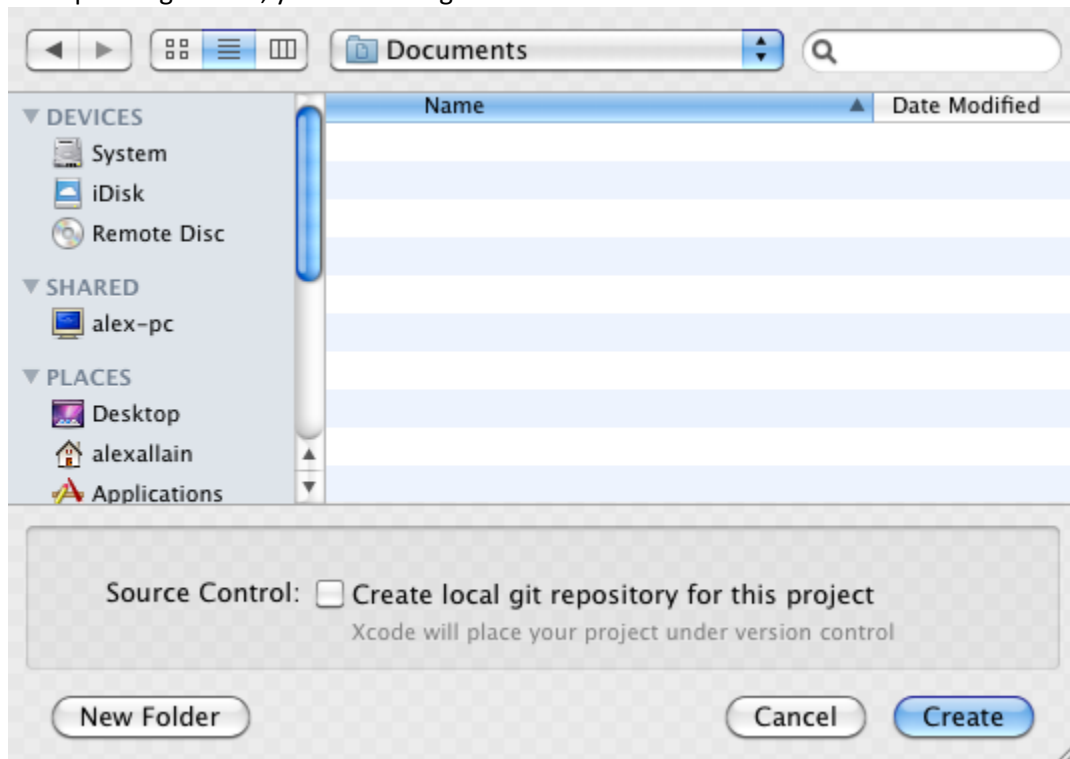


After pressing “Next”, you will see this screen:



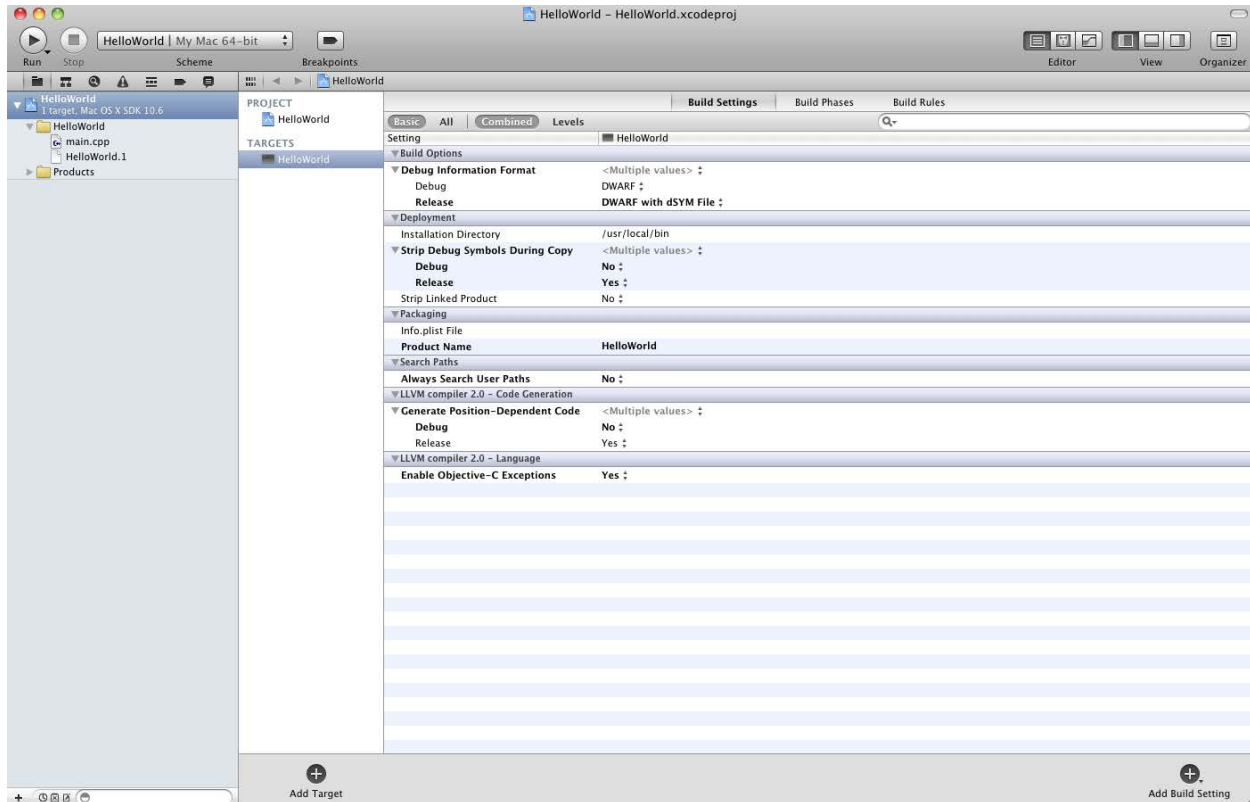
I’ve already filled it out with a product name, “HelloWorld”, and I’ve chosen the Type to be C++ (it defaults to C). Do that, and then press “Next” again.

After pressing “Next”, you’ll be brought to this screen:



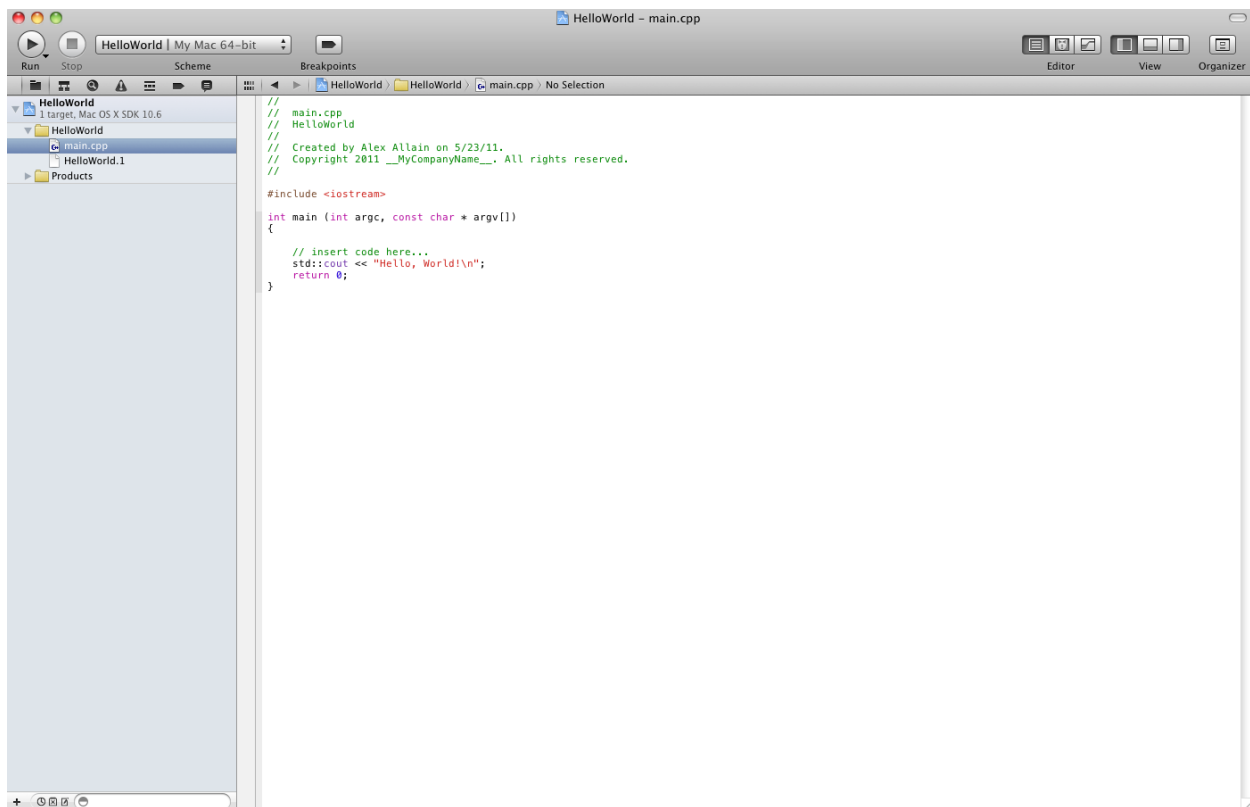
If “Create local git repository for this project” is checked, you can uncheck it. Git is a “source control” system that allows you to keep multiple versions of your project, but git is outside the scope of this book. You should also choose a location for your project—I put this one in Documents. Once you’ve made these choices, press “Create”.

After pressing “Create”, a new window will come up that looks like this:



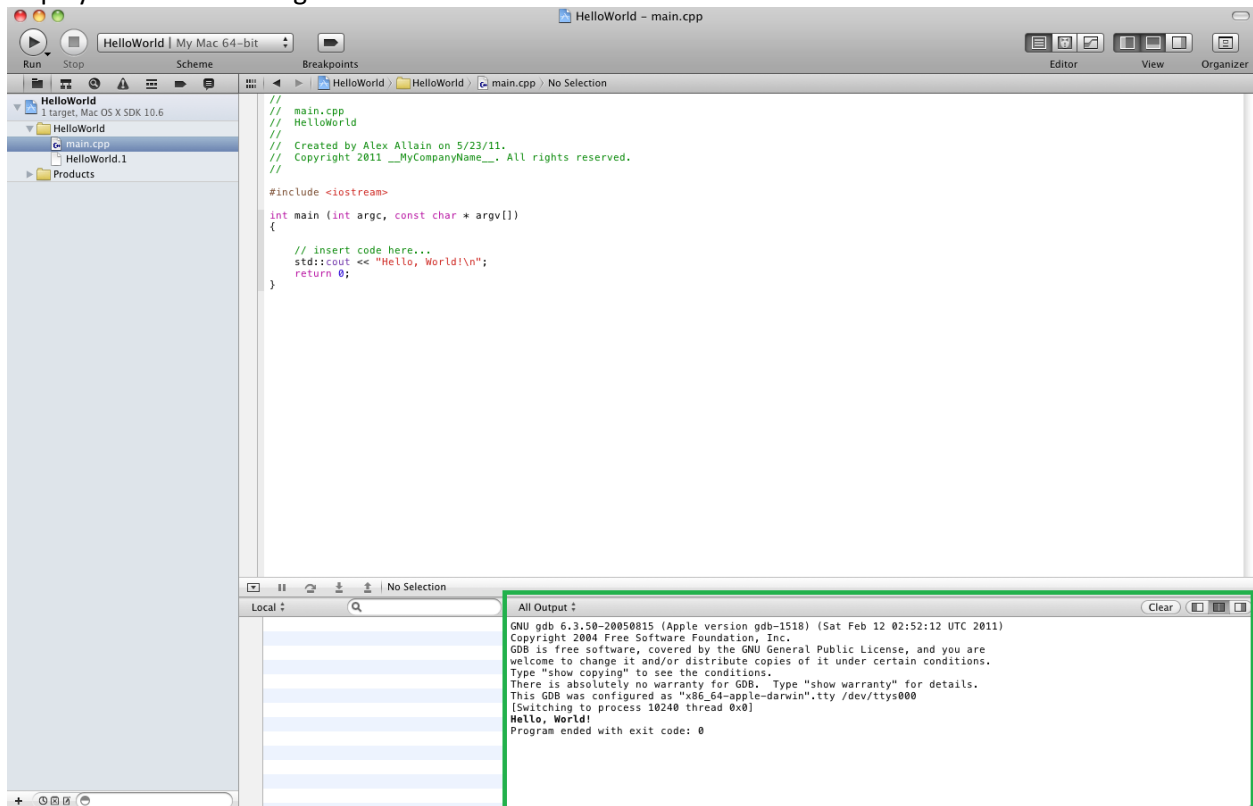
This view shows you quite a few things. The right sidebar gives you access source code and Products. The source code is under the directory named after your project, in this case “HelloWorld”. Most of the rest of this screen is displaying compiler configuration, which we don’t need to do anything with right now.

Let’s work on the source file itself. Go ahead and select “main.cpp” in the folder on the left sidebar. (Notice the file extension: .cpp is the standard extension for C++ source files—not .txt—even though cpp files are plain text.) If you single-click you will bring up the source in the main window. You can then start typing directly into the file.



You can also double-click on the file in order to bring up an editor window that can be moved around.

By default Xcode provides a small sample program that you can start with. Let's compile and then run this sample program. All you need to do is click the "Run" button on the toolbar! The output will be displayed in the lower right:



And there you go—you've run your first program!

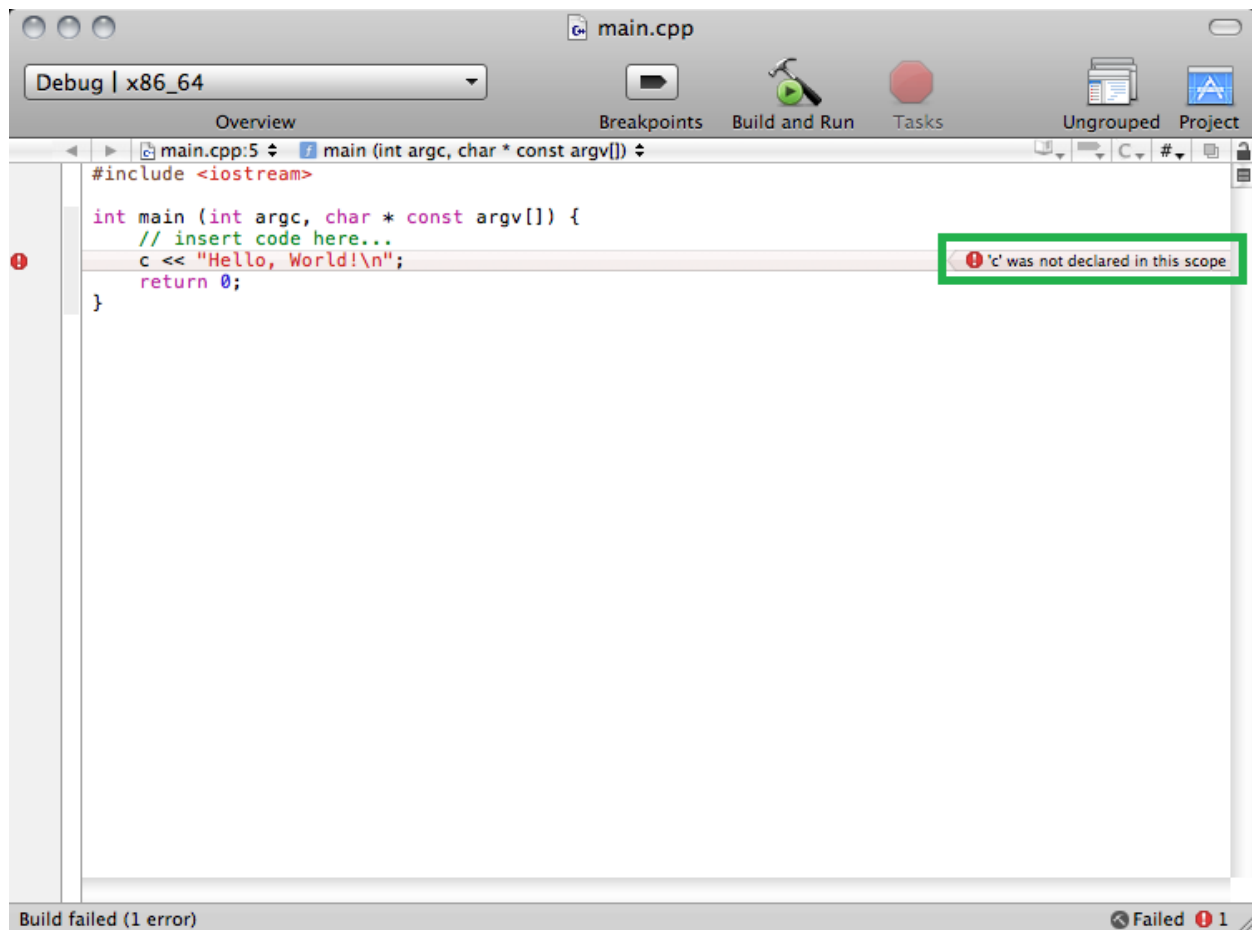
From here on out, whenever you have a sample program you want to run, you can either use the project we just created, or you can create a new project for it. In either case, when you want to add your own code, you can start by modifying the sample program that Xcode creates in main.cpp.

## Troubleshooting

[This section uses screenshots from Xcode 3. I have noted where Xcode 3 and Xcode 4 are different.]

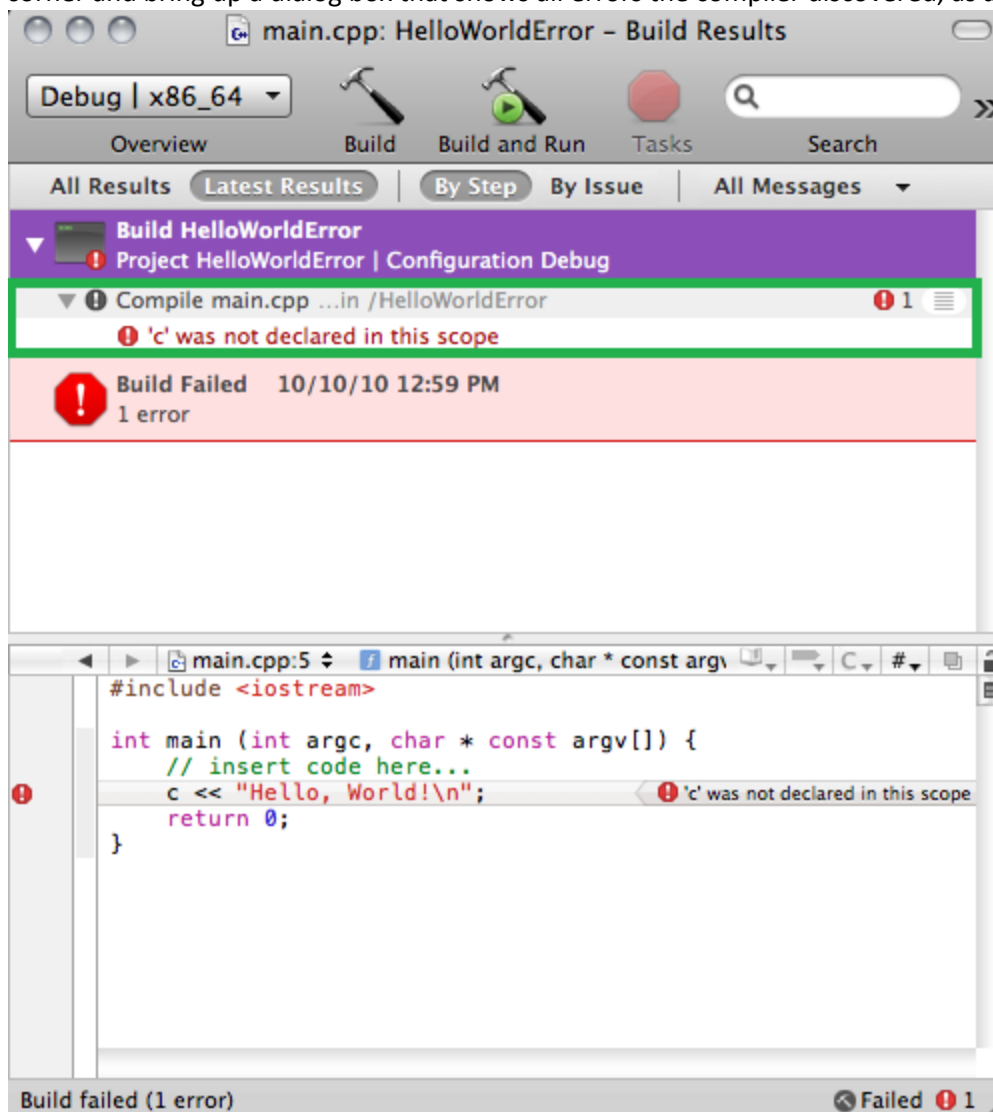
It's possible that your program will fail to compile for some reason, usually because of a compiler error (for example, perhaps a typo in the sample program or a real error in your own program). If this happens, then the compiler will display one or more compiler error messages.

Xcode displays compiler error messages directly alongside the source code, at the line where the error occurred. In the below example, I modified the original program so that instead of `std::cout`, it has simply `c`.



In the rectangle, you can see the compiler error—that Xcode doesn't know what 'c' is. You can also see a message that the build failed, in the lower left corner, and again in the lower right corner, along with a count of the number of errors (1, in this case). (In Xcode 4, the icon is similar, but it appears in the upper-right corner.)

If you want to see a full list of errors, in Xcode 3 you can click on the hammer icon in the lower-right corner and bring up a dialog box that shows all errors the compiler discovered, as a list:



Again I've highlighted the place where you can see the actual error, and if you click on it, it will show a small editor window where you can see the error in the code itself.

In Xcode 4, the right-hand panel where the source files were located is replaced with compiler errors if the build fails.

Once you fix the error, you can simply press the "Build and Run" button again to try again.

## Linux

If you are running on Linux, you almost certainly already have a C++ compiler installed. Typically, Linux users use the C++ compiler `g++`, which is part of the GNU Compiler Collection (GCC).

### Step 1: Installing g++

To check if you have g++ installed, bring up a terminal window. Type in g++ and hit enter. If you have your compiler already installed, you should see:

```
g++: no input files
```

If you see a phrase like this one:

```
command not found
```

then you will probably need to install g++. Installing g++ will depend on your particular Linux distribution's package management software. If you are running Ubuntu, for example, you may need to simply type:

```
aptitude install g++
```

Other Linux distributions may have similarly easy package management or may require additional steps. Read the documentation from your Linux distro for more.

### Step 2: Running g++

Running g++ is relatively easy. Let's create your very first program right now. Create a simple file with a .cpp extension that contains exactly this text:

```
#include <iostream>

int main ()
{
    std::cout << "Hello, world" << std::endl;
}
```

**Sample Code 1: hello.cpp**

Save this file as `hello.cpp`, and remember the directory where you put it. (Notice the file extension: .cpp is the standard extension for C++ source files—not .txt—even though cpp files are plain text.)

Go back to the terminal window, and change to the directory where you saved the file.

Type:

```
g++ hello.cpp -o hello
```

Then hit enter.

The `-o` option to g++ provides a name for the output file. If you don't use it, the name defaults to `a.out`.

### Step 3: Running your program

In this case, we gave the file the name `hello`, so you can now run your new program by typing

```
./hello
```

And you should see the output



```
Hello, world
```

And there is your first program, saying hi to the brave new world.

### Troubleshooting

It's possible that your program may fail to compile for some reason, usually because of a compiler error (for example, if you entered the sample program with a typo). If this happens, then the compiler will display one or more compile error messages.

For example, if you put an `x` before `cout` in the sample program, the compiler would come back with these errors:

```
gcc_ex2.cc: In function 'int main ()':  
gcc_ex2.cc:5: error: 'xcout' is not a member of 'std'
```

Each error shows you the file name, a line number, and an error message. Here, the issue is that the compiler doesn't know anything about `xcout` since it should just be `cout`.

### Step 4: Setting up a text editor

If you're using Linux, you will also want to find a good text editor to use. Linux has some very high end text editors available, such as [Vim](#) and [Emacs](#) (I use Vim when I'm working on Linux). But they are relatively difficult to learn, and require a real time investment. In the long run, it's worth it, but you may not want to take the time when you're also starting to learn to program. If you are already familiar with either of these tools—feel free to continue using them.

If you don't already have a favorite editor, you may want to try a text editor like `nano`. [Nano](#) is a comparatively simple text editor, but it does have certain valuable features like syntax highlighting and automatic indentation (so that you don't have to keep pressing tab all the time when you go to a new line in your program—sounds trivial, but you really do want it). Nano is based on an editor called pico, which is a very simple editor to learn to use but that lacks many features needed for programming. You may even have used pico if you've used the mail program Pine. If not, that's ok, no prior experience is necessary to start working with nano.

You may already have nano—to find out, type `nano` in a terminal window. It may launch automatically. If not, and you get variant of

```
command not found
```

then you will need to install nano—you should follow the instructions for getting nano using your Linux distribution's package manager. I've written this section with version 2.2.4 of nano in mind, but later versions should be fine.

### Configuring Nano

In order to take advantage of some features of nano, you will need to set up a nano configuration file. The configuration file for nano is called `.nanorc` and like most Linux configuration files, your user-specific configuration resides in your home directory (`~/ .nanorc`).

If this file already exists, you can simply edit it—otherwise, you should create it. (If you have no

experience at all using text editors on Linux, you can use nano to do this configuration—read below if you need help with the basics of nano!)

To configure nano properly, use the sample `.nanorc` file that comes with this book. It will provide you will nice syntax highlighting and auto-indentation, which will make editing source code much easier.

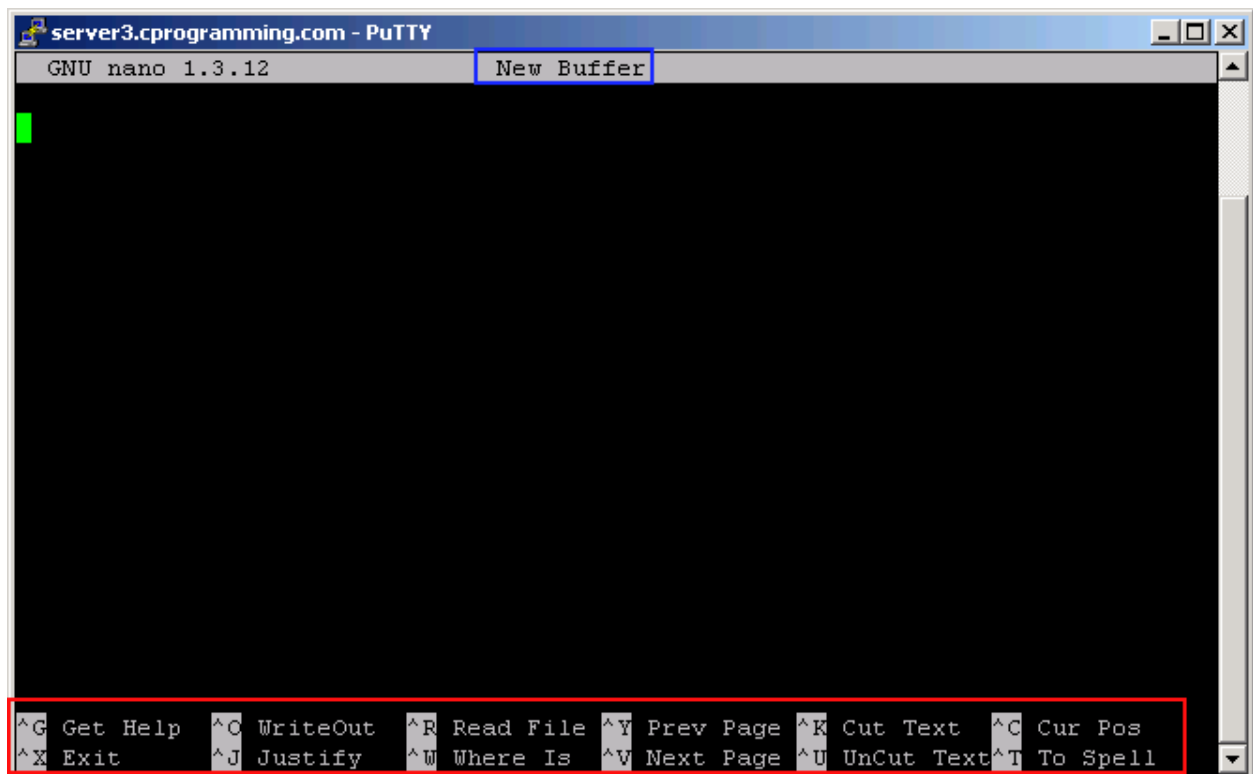
## Using Nano

You can run nano without providing an argument if you wish to create a new file, or you can specify a filename at the command line to start editing that file:

```
nano hello.cpp
```

If the file doesn't exist, nano will start editing a new buffer associated with the file. It will not, however, create the file on disk until you actually save your changes.

Here's an example of what nano should look like when you run it.



In the rectangle at the top is the title of the current file being edited or "New Buffer" if you ran nano without providing a file.

In the rectangle at the bottom are a bunch of keyboard commands. Any time you see the ^ character in front of a letter, that means you need to press the Control key on your keyboard in combination with the letter—for example, exit is shown as ^X, so to exit you press Ctrl-X. The capitalization is not important.

If you're coming from a Windows world, you may not be familiar with some of the terminology used by

nano, so let's look at some basic nano operations.

### Editing text

When you launch nano, you can either bring up a new file or open an existing file. At this point, you can simply begin typing into the file—in this respect nano is very similar to Notepad on Windows. If you want to use copy and paste, though, the terms are different—Cut Text (Ctrl-K) and UnCut Text (Ctrl-U). These commands default to cutting a single line of text if you haven't selected any text.

You can also search for text in a file using "Where Is" by pressing Ctrl-W. This brings up a new set of options, but the simplest of them is to simply type in the string you're looking for and hit enter.

You can navigate a page at a time using Prev Page (Ctrl-Y) and Next Page (Ctrl-V). Notice that the keyboard shortcuts have little in common with Windows.

The only major feature that nano lacks, that most other text editors have, is that nano currently (in version 2.2) has only experimental support for undo/redo functionality. All undo/redo functionality is disabled by default.

You can use nano to do a file wide search/replace by pressing Alt-R—you'll first be prompted with text to find, and then the text to replace it with.

### Saving files

Saving a file is called, in nano parlance, WriteOut (Ctrl-O).

```
File Name to Write:           
^G Get Help      ^T To Files      M-M Mac Format   M-P Prepend
^C Cancel        M-D DOS Format   M-A Append       M-B Backup File
```

When you invoke WriteOut, you will always be prompted for the name of the file to write to, even if you have a file already open. If you are already editing a file, the name of the file will be shown by default, so you can just hit Enter and it will save the file. If you want to save to a new location, can type in the name of the file to save, or you can use the To Files menu option (Ctrl-T) to select a file to write to. Cancel (Ctrl-C) speaks for itself—most commands will have the option of cancelling them—but unlike a Windows machine, the default cancel button is Ctrl-C rather than Escape. Don't worry about the other options that are available for now—you shouldn't need to use them most of the time.

### Opening files

If you want to actually open a file for editing, you use Read File (Ctrl-R). Read File brings up a new set of menu options.

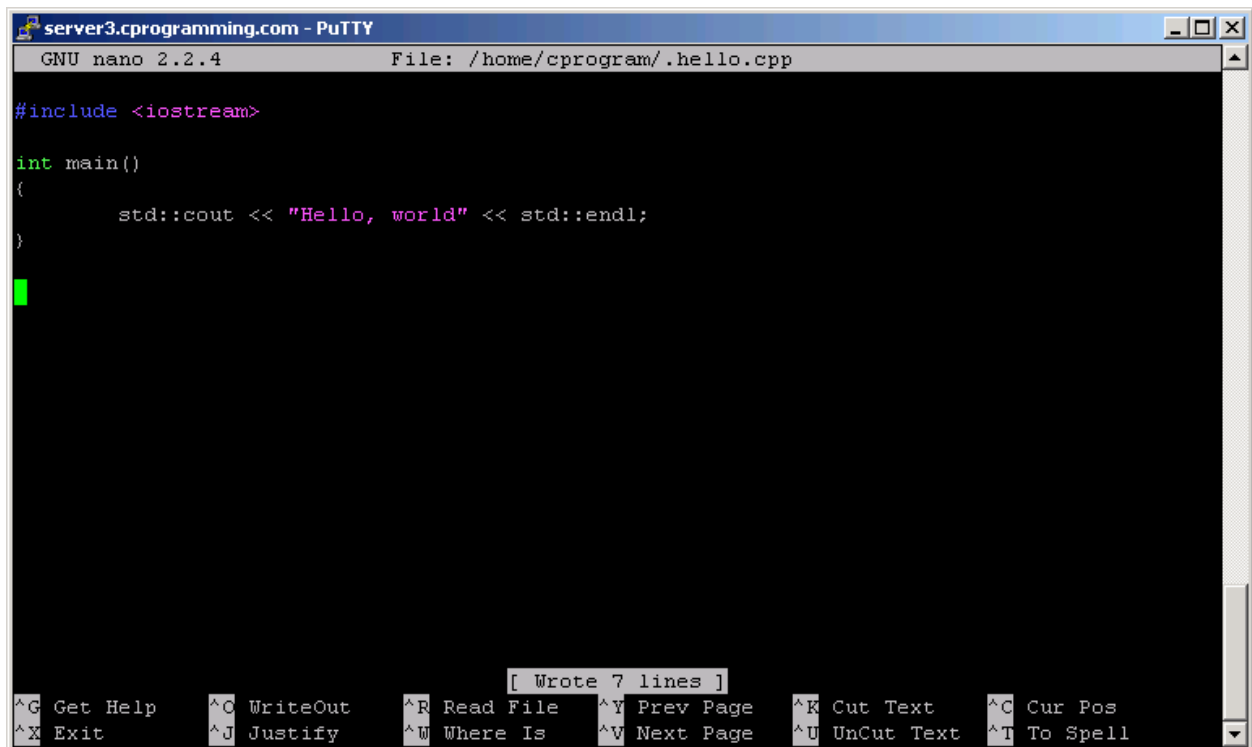
```
File to insert into new buffer [from ./] :           
^G Get Help      ^T To Files      M-F New Buffer
^C Cancel        ^X Execute Command
```

If you want to open the file, rather than insert the text directly into the file you're currently editing, choose New Buffer at this menu, before selecting a file. The shortcut for New Buffer is M-F. The M

stands for meta key—in this case, you'd normally use the Alt key on your keyboard: Alt-F.<sup>4</sup> This tells nano that you are going to open the file. Once you've done that, you can either type in the name to a file, or you can use Ctrl-T to bring up a file list that will let you select the file you want to edit. As usual, you can use Ctrl-C to cancel the operation.

### Looking at a source file

Now that you've learned a bit about editing in nano, you should be able to open a source file and start working on it. If you've got your .nanorc file configured properly, when you open a source file that has some text in it, it should look something like this if you open the file `hello.cpp` we ran earlier. Text is displayed in different colors depending on what its function is; for example, the "Hello, world" text appears in bright pink:



```
server3.cprogramming.com - PuTTY
GNU nano 2.2.4      File: /home/cprogram/.hello.cpp

#include <iostream>

int main()
{
    std::cout << "Hello, world" << std::endl;
}

[ Wrote 7 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Syntax highlighting is file extension based, so until you save the file as a source file (.cpp), it won't have highlighting.

From here on out, whenever you have a sample program you want to run, you can simply create a new text file for that program using nano and compile it using the steps above.

### Learning more

You should now be able to edit basic files in nano, but if you want to learn more, the built-in help is simply a Ctrl-G away. I also found this website to be particularly useful for explaining more advanced nano features:

<http://freethegnu.wordpress.com/2007/06/23/nano-shortcuts-syntax-highlight-and-nanorc-config-file-pt1/>

---

<sup>4</sup>Some folks may have trouble using the Alt key for the meta key; if you find that using the Alt key doesn't work, you can always press and release Esc before pressing the letter—for example Esc F is the same as pressing Alt-F.

### Ready to buy?

If you liked what you've read so far, you can purchase the full title, over 350 pages of C++ knowledge, tips and practice problems: [Buy \*Jumping into C++\*](#)

### By the way...

When you purchase *Jumping into C++*, not only will you be able to download the entire ebook instantly, but if you are not satisfied with *Jumping into C++* after 60 days, just let me know and I'll refund your money – that's how confident I am that you'll learn C++ from my book.

[Buy \*Jumping into C++\* Today!](#)

