

Using Xcode For Macintosh

In CS106B, you have the option of writing your programs on the Mac or PC. For the Macintosh environment, you will write your programs using a C++ compiler by Apple called Xcode. Our CS106 libraries were developed and tested on Xcode version 2.5 which is available for free download from Apple's developer site. Xcode 2.5 requires Mac OS X 1.4 (Tiger) or newer¹. The libraries also work on Xcode 3.0 for Max OS X 1.5 (Leopard). You can also use Xcode on the Mac OS computers in the public clusters on campus. Please see the next section for instructions on how to do this.

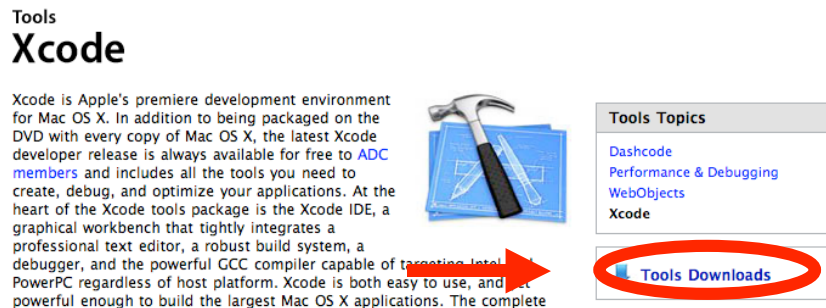
Using Xcode at a cluster

Xcode is available for use in the Lair (the Tresidder Macintosh cluster), on the Macs in Meyer library, and in the residential computer clusters. The 106 libraries will already be loaded on them, so you should be able to just open Xcode and proceed from "Creating the add2 project" section.

Downloading your own copy of Xcode from the Web

First, note that most versions of Mac OS X come with Xcode preinstalled. However, you should check the version of Xcode you have. The easiest way to do this is to first open the application. Then, from the Xcode menu, select "About Xcode". The window that opens should say what version you have. If it is version 2.5 or later, you have the correct version. If not, use the following directions to download the most recent version.

Use your web browser to connect to <http://developer.apple.com/tools/Xcode/>. Click on the Tools Download link.

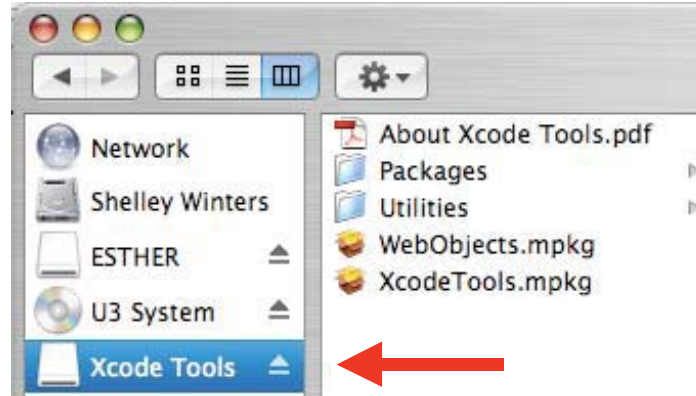


The screenshot shows the Apple Developer Tools page for Xcode. At the top, it says "Tools" and "Xcode". Below this is a paragraph of text describing Xcode as Apple's premiere development environment for Mac OS X. To the right of the text is an image of a hammer and a pencil. Further right is a "Tools Topics" sidebar with links for Dashcode, Performance & Debugging, WebObjects, and Xcode. At the bottom of the page, there is a "Tools Downloads" link, which is circled in red. A red arrow points from the text "targeting Intel" in the paragraph to the "Tools Downloads" link.

Then, if you have Mac OS X 10.4 (Tiger), click on the "Xcode 2.5 (DMG)" link. If you have Max OS X 10.5 (Leopard), click on the "Xcode 3.0 (DMG)" link. At this point, if you do not have one, you will need to create an Apple account. However, creating an account is free, so go ahead and make one. Click on the Xcode 2.5 / Xcode 3.0 (Disk Image). This will begin the download.

¹ If you have an earlier version of Mac OS and would like to use Xcode, contact us and we can give you special instructions

Once the download is finished, there should be a drive called “Xcode Tools” loaded.



If not, double click on the downloaded file, “`xcode_2.5.1_8m1910_6936315.dmg`” (it may not be called exactly this; don’t worry if it isn’t). Inside the “Xcode Tools” drive, double click on “XcodeTools.mpkg”. This will open an installer for Xcode. Follow the instructions in the installer, and when you are finished Xcode will be installed.

Downloading and Installing the CS106 Libraries

Here you will download the special libraries, such as genlib, that we use in this course. To start, use a web browser to access the following

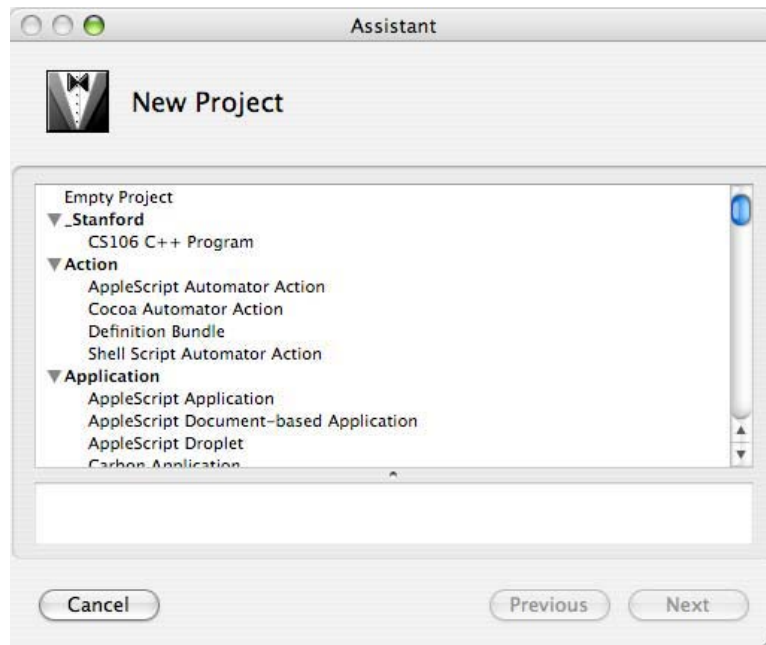
http://see.stanford.edu/materials/icspacs106b/CS106Libs_for_Xcode.zip

Follow the instructions in the installer to install the CS106 Libraries.

Creating the add2 project

Every program written using Xcode has a project that indicates what different program files need to be compiled together in order for the complete program to work. **Note that currently projects with spaces in their names will not create properly for Stanford Projects in Xcode.** Instead of naming a project something like “My Project”, you should use “MyProject” or “My_Project” or something else similar. We are going to create a project that uses a simple pre-written program, named `add2.cpp`. You can download the code for `add2.cpp` by using a web browser to go to the link <http://see.stanford.edu/materials/icspcs106b/add2.cpp> select the "Save As" option from the File menu to saving the file to your own computer. Once you have the `.cpp` file, you will need to create an Xcode project and add the `.cpp` file we provide. You will go through the basic process described below for each programming assignment that you do in this course, using your own code files.

Start the Xcode application. It's a good idea to create an alias for this program and place it on your desktop or in another convenient place. You can also put it in your program dock. Select `New Project...` from the `File` menu. The following dialog will appear:



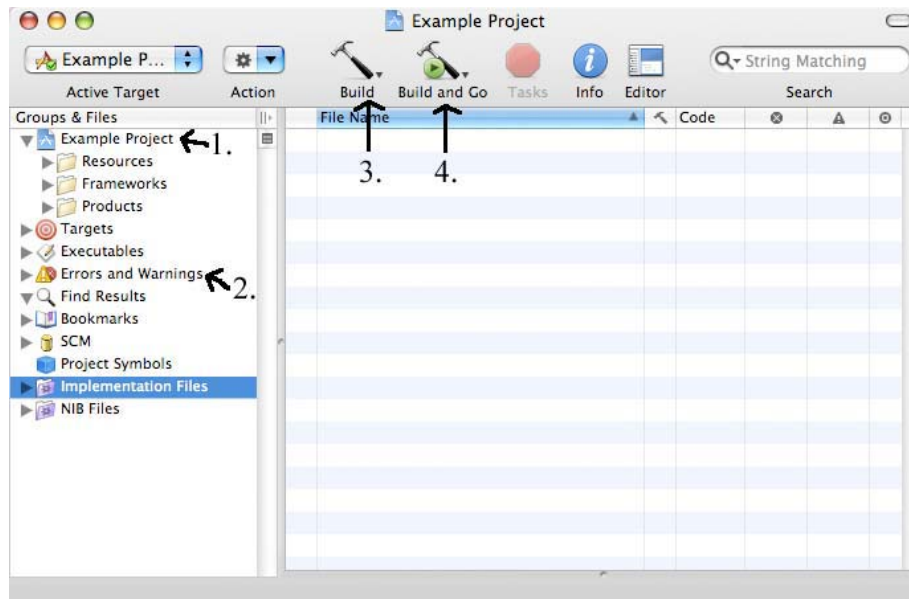
Now you need to tell Xcode what type of project you are creating, what to call it, and where to put it. First, highlight "Stanford CS106 C++" in the list on the left, then click "Next". Type "Example_Project" as the name of the project.

Now you have to show where the project should be located. Xcode has a default location, which you can use if you'd like. Otherwise, click "Choose..." and use the resulting dialog to select where you'd like the project to be. Note that the end of the location will be the same name as your project (e.g., "~/Example_Project/"). This is a folder Xcode will automatically create to store the project files.

Click OK. The project is now created.

How to Use the Xcode Project Window

The Xcode project window has a number of folders and buttons. You will only need to use a few of them; this section will walk you through them.



1. This is your project folder. When you want to create a new file or add an existing file, you right click on this folder and select the appropriate option.
2. Selecting this folder will show you the current errors or warnings in the current project. Note that they will not appear until you build the project (build means to create a machine readable version of your code).
3. This button will build the project.
4. This button will build and, if there are no errors, run the project.

Adding add2.cpp to the Example Project

To start, move `add2.cpp` into the project folder that Xcode created for “Example_Project”. Then, in Xcode, right click on the project folder (1, above). Mouse over “Add” and then select “Existing Files”. The starting folder should be your project folder, so click on “`add2.cpp`”. If not, navigate to the project folder, and then click on `add2.cpp`. Click “Add” and then “Add” again on the resulting dialog window. `add2.cpp` is now loaded into your project.

Building and Running add2.cpp

At this point, you can run the `add2` program. You can do this in one of a couple ways. The easiest way is to click on the “Build and Go” button (4, above). If you opened `add2.cpp`, there is a corresponding button in that window that you can also click on.

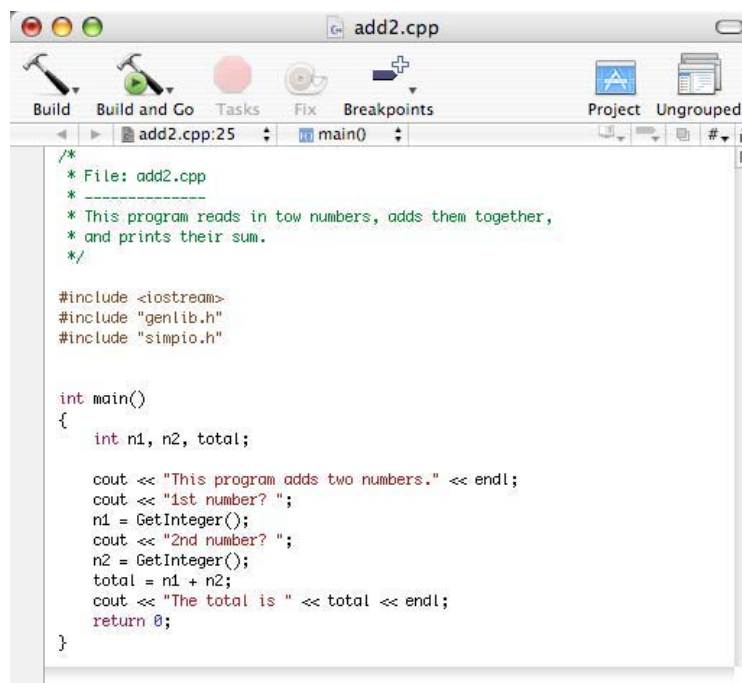
Two new windows will then appear. One is a log window that contains some information about the currently running program. The other is called the *console window*. The console window makes it possible to interact with the running program. Any output generated by the program (usually through the use of the `cout` function) will appear in the console window. When the program needs input values, you will enter these by typing data into

the console window. When your program is finished, you select `Quit` from the menu with the name of the project to return to the editing environment.

Looking at the `add2.cpp` program

If you haven't already, you can also open up a source file by double-clicking on its name in the project window. Doing so opens a new window with the actual text of the program. For example, if you double-click on the `add2.cpp` file, the text of that program will appear in an *editor window*. You can then edit the program in this window to change its behavior. The editor works very much like all of the other editors on the Macintosh.

If you open up the `add2.cpp` file, you will see the following program display:



```
/*
 * File: add2.cpp
 * -----
 * This program reads in two numbers, adds them together,
 * and prints their sum.
 */

#include <iostream>
#include "genlib.h"
#include "simpio.h"

int main()
{
    int n1, n2, total;

    cout << "This program adds two numbers." << endl;
    cout << "1st number? ";
    n1 = GetInteger();
    cout << "2nd number? ";
    n2 = GetInteger();
    total = n1 + n2;
    cout << "The total is " << total << endl;
    return 0;
}
```

Note that we are including `genlib.h` and `simpio.h` from the CS106 libraries. Those files are part of the CS106 libraries that you downloaded from the class web page and can be included in any file in a project created with the CS106 C++ stationery.

Running your program

Writing a program and typing it in is hardly the end of the programming process. You need to test your program and make sure that it works. When you clicked on the “Build and Go” button, the system translated the source program into the internal language of the machine. This process is called *compiling*. If you’ve done everything right, the compilation process will finish in a few seconds*, and you can test your program to see if it works. If it does, awesome! If the program is for an assignment, print out a copy of the program and go on to the next problem.

* Actually, the very first time you compile an Xcode project it may take much longer while it builds a cache to enable all successive compilations to be faster.

Chances are, however, that on your real assignments you won't be quite so lucky. Somewhere in typing in your program, you will have entered something incorrectly or made some other sort of mistake. Perhaps your program does not work as intended, or, worse yet, does not even make it through the compilation process. Making such errors seems to be an unavoidable part of programming. When errors do occur, it is time to sit down, find them, and fix them. This is the process of *debugging*.

Debugging syntax errors

The errors that you are likely to encounter fall into two principal classes. The first class of errors are *syntax errors*, which are those cases in which you violate some linguistic rule of the programming language. As the C++ compiler goes through your program to translate it into its machine-language form, it has encountered some piece of the program that it can't understand. Such errors may be typographical (if, for example, you spell some word incorrectly) or they may reflect a lack of understanding of the language rules.

Xcode is quite good at helping you find your syntax errors. When the compiler encounters one, it adds a red x in the margin of the source file on the line where it encounters the error. It also lists all of the errors it encounters in the "Errors and Warnings" section in the Project Window (2, above). If you select this section you will see all of the errors, and if you double click on any of them it will take you to the line where that error appears. It will also give you a description of the error. When you're new to programming with Xcode, these messages may appear obscure, but you will become more used to them as you go along. In addition, there is a section on the class wiki on error messages. However, since the wiki is new, not all error messages will be there. If you encounter and solve an error message that is not on the wiki, feel free to add it so others can benefit from your knowledge. Once you have corrected all of the errors Xcode found, save the file and try running it again.

Debugging logic errors

Eventually, you will get the last syntax errors out of your program, and you'll be up and running. This is when the fun begins. The most serious errors in a program are the ones the compiler doesn't catch—the errors that occur when your program is doing exactly what you told it to do, only that what you told it to do wasn't at all what you really wanted it to do. These are the second class of errors, known as *logic errors* or, more commonly, *bugs*.

Since this is a larger topic, we will have another handout devoted to debugging logic errors in Xcode that you should read and become familiar with.

Source: <http://see.stanford.edu/materials/icspacs106b/H05M-UsingXCode.pdf>