CS106 - Using Metrowerks CodeWarrior Pro 5

In CS106, you have the option of writing your programs on the Mac or PC. For the Macintosh environment you will write your programs using the C compiler by Metrowerks called CodeWarrior Professional Release 5. CodeWarrior provides a very nice editing environment and debugger, and it is the state-of-the-art C compiler for the Mac.

Getting a copy of Metrowerks CodeWarrior

CodeWarrior is available for use in the Lair (the Tresidder Macintosh Cluster). Often it will already be in the applications folder on the machine's hard disk. If there is a folder there named something like `CodeWarrior Pro 5`, then the application can be found within that folder. Otherwise, you need to go to the "Apple" menu and choose the item called `Get Apps`. This will bring up a submenu, from which you can choose the item `CodeWarrior`. After a moment, the `"CodeWarrior Pro 5"` folder should appear in the `Applications` folder on the hard disk. The CodeWarrior IDE 4.0.1 application, along with all of the library files, can be found in the `"CodeWarrior Pro 5"` folder. The IDE (Integrated Development Environment) is the program you run to start CodeWarrior.

For your own machines, you will also need to obtain a copy of CodeWarrior itself, but you may not copy this program from Tresidder. CodeWarrior is proprietary software, and unauthorized copying of software is a serious violation of Stanford policy. CodeWarrior Professional Release 5 is available at the Bookstore. You may also find earlier versions, such as CodeWarrior 11 (that was the last one before the "Professional" series) or CodeWarrior Pro 4, but you are strongly urged to buy Pro 5. It is the "official" version this year in CS classes.

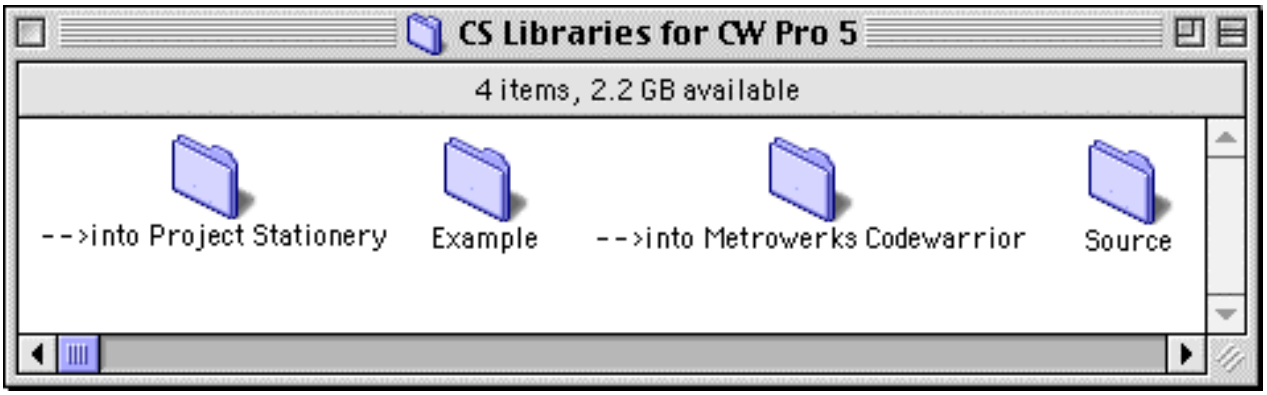
We will also use the special CS libraries developed here at Stanford. These are in the public domain and may be freely copied, as described in the section to follow.

Downloading Software from the Web

Use your web-browser to login to our class website to download the necessary files. Click on "Software", then follow the link to the MetroWerks CodeWarrior libraries. You will download a file called cslibs5.hqx, and under most browsers, this file will then be automatically expanded to give you a folder entitled `"CS Libraries for CW Pro 5"`. If it does not, use Stuffit Expander to expand the file `cslibs5.hqx`. If for some reason you do not have Stuffit Expander you can download it from Aladdin software at http://www.aladdinsys.com .

The contents of the CS Libraries for CW Pro 5 Folder

Once you have expanded and opened the `"CS Libraries for CW Pro 5"` folder, you should see the following window on your screen:



If you are working from **within Tresidder** the only folder you will need is entitled `"Example"`; everything else should already be properly installed on the machine. You can throw everything else inside `"CS Libraries for CW Pro 5"` into the trash.

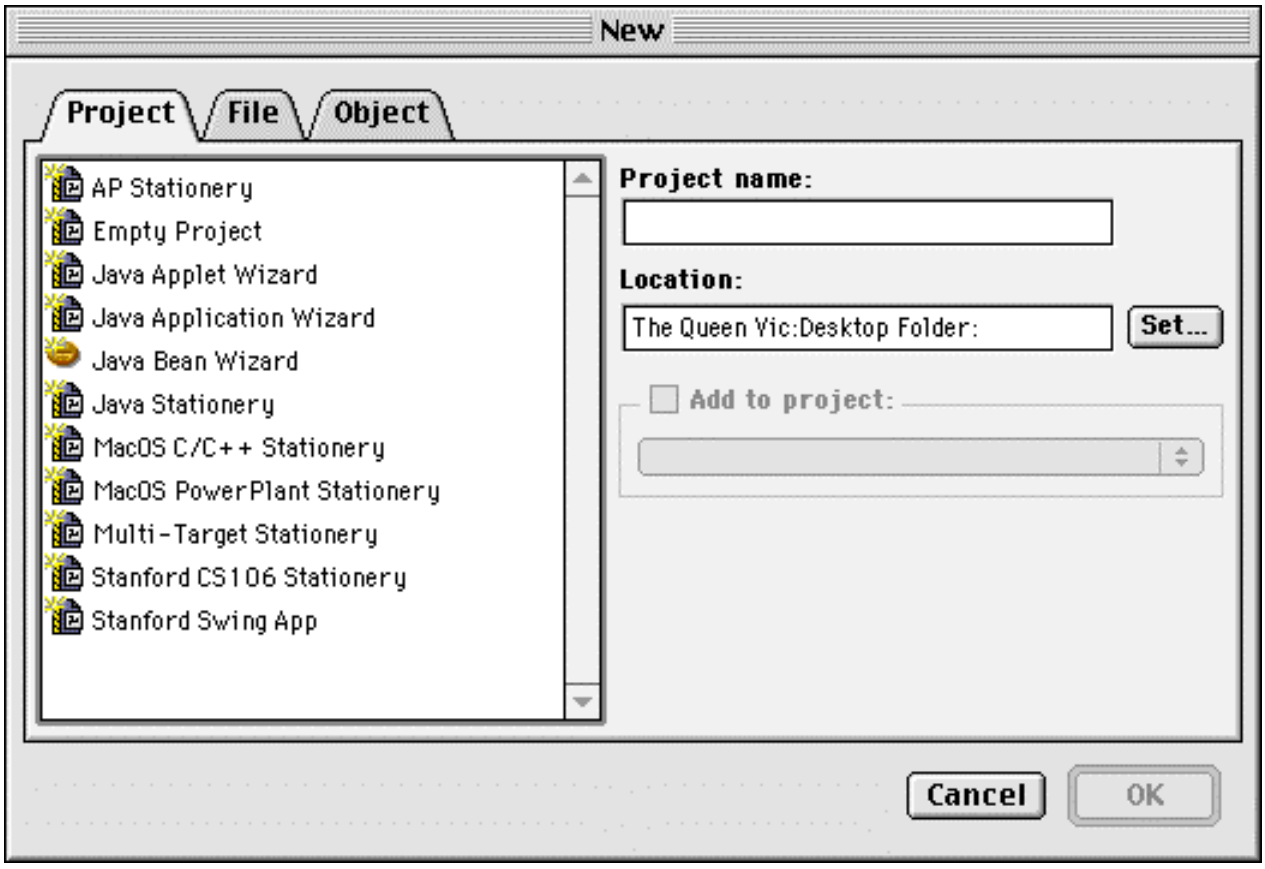
If you are setting up CodeWarrior on **your own machine**, you'll also be using the contents of the `"-->into Metrowerks Codewarrior"` and `"-->into Project Stationary"` folders. Refer to the end of this handout for more details on library installation.

You shouldn't need anything from the `"Source"` folder. Just ignore it unless you're curious about the code used to create the CS Libraries.

Creating the add2 project

Every program written using CodeWarrior has a project that indicates what different program pieces need to be compiled together in order for the complete program to work. We are going to create a project that uses a pre-written program called `add2.c` which can be found in the `"Example"` folder. You will go through the same basic process for each programming assignment that you do in this course.

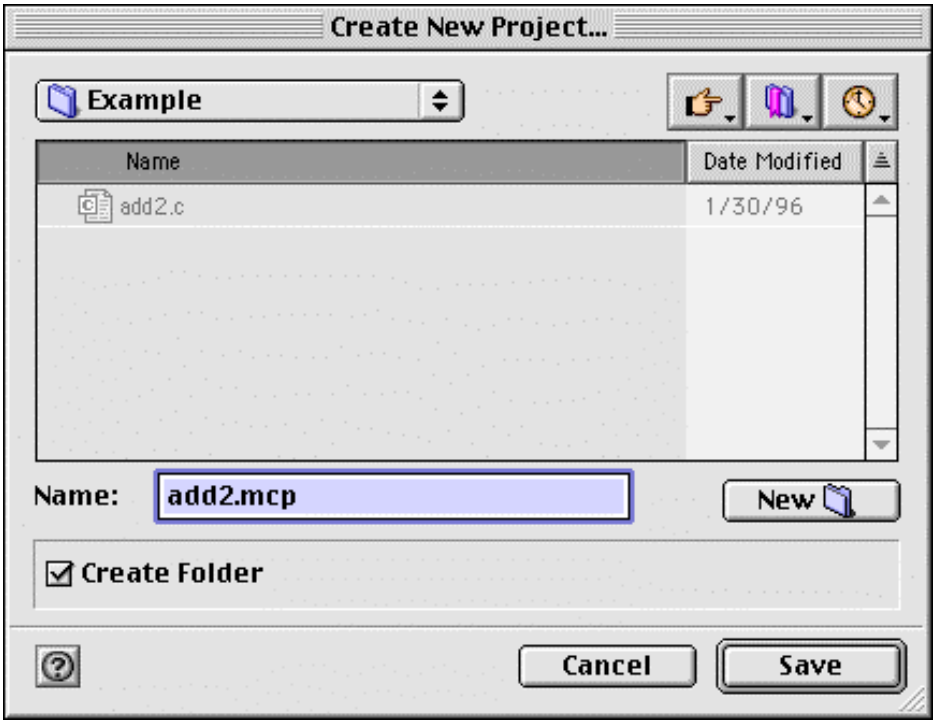
Start the CodeWarrior IDE 4.0.1 application, and select `New...` from the `File` menu. The following dialog will appear:



Now you need to tell CodeWarrior what type of project you are creating, what to call it, and where to put it. The dialog that appeared lets you do just that.

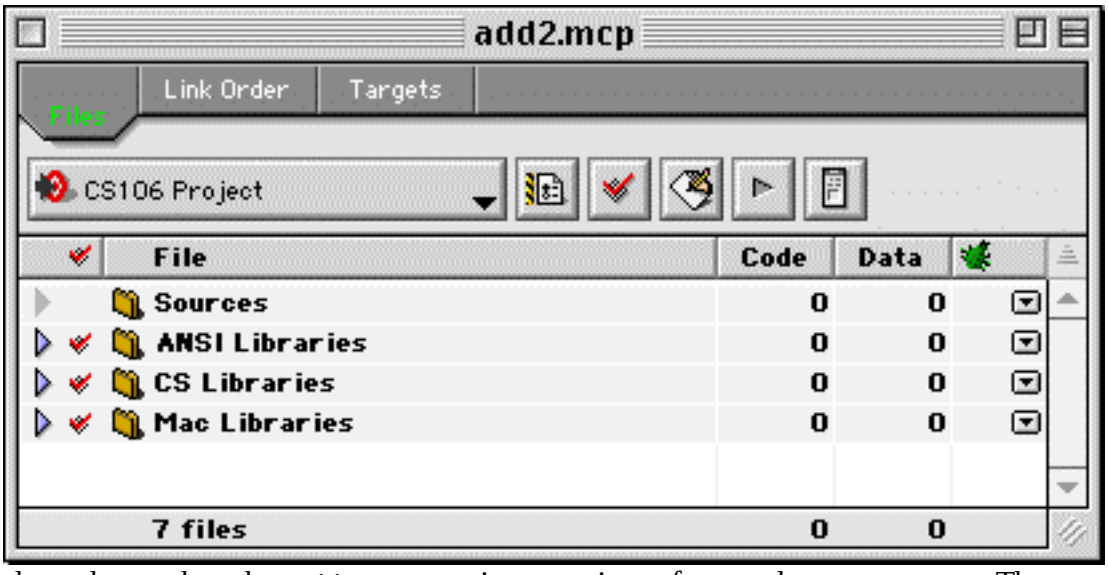
Highlight "Stanford CS 106 Stationery" in the list on the left, then type add2.mcp as the name of the project. Using "`.mcp`" (which stands for (MetroWerks CodeWarrior Project) as the extension on the project name is a useful convention that we suggest you follow--it will help you identify your projects.

Now you have to show where the project should be located, which in this case is in the Example folder. Click the "Set" button, and use the navigation facility at the top of the dialog that appears to find the "`Example`" folder in the folder where you got the CS libraries. The dialog will then look like this:



Since we do not want to create a new folder within the `Example` folder, uncheck the `Create Folder` button, and then press `Save`. This returns you to the New Project dialog.

Click OK. A dialog concerned with project stationery will appear, but since there are no more choices you need to make, just click OK. CodeWarrior now creates your project, and you will see the following project window:



This window shows that the `add2.mcp` project consists of several components. The ANSI, CS, and Mac Libraries are already there (click on the inspection triangles if you want to see what's inside these sections of the project). The CS library contains code

specific to CS106 that facilitates input and output, manipulating strings, and working with graphics.

What we need to do now is add the `add2.c` source file. Drag the `add2.c` file from the `Example` folder onto the project window, positioning the mouse cursor just below the word `Sources`. You will see a small black triangle just below the "S" of `Sources` when you are in the right position, and then you should release the mouse button. The triangle to the left of `Sources` should undim, and if you click it, you will see that `add2.c` is in that section of your project. (If you "miss", just drag `add2.c` around in the project window to the right spot.) Another way to add this file to your project is to use `Add Files` from the `Project` menu--highlight `Sources` first.

At this point, you can run the `add2` program. First go to the `Project` menu, and if `Disable Debugger` appears near the bottom of the list, select it. Then pull down the `Project` menu again and select `Run`. When you do, CodeWarrior will compile your program and run it.

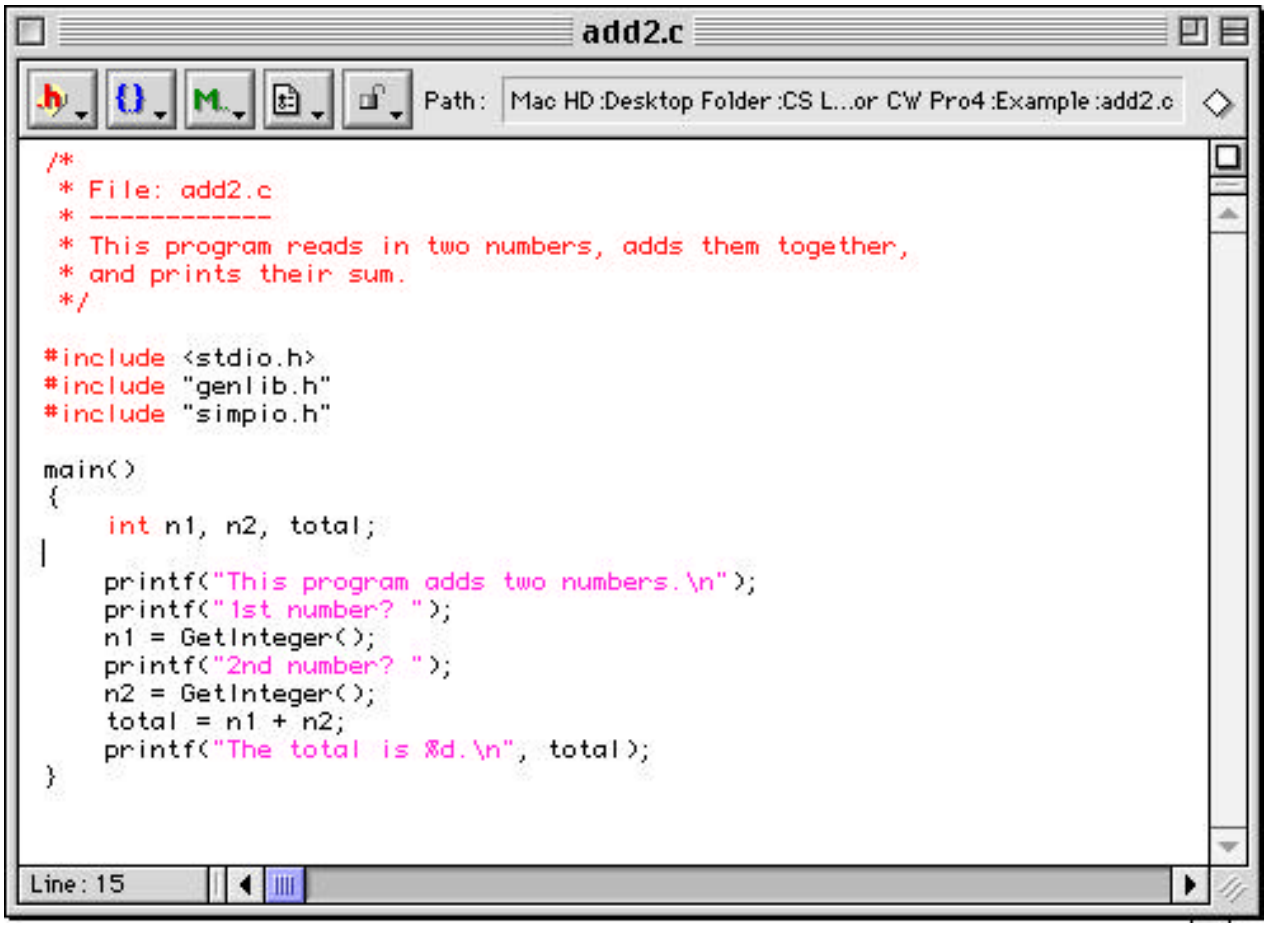
A new window will appear called the *console window*. The console window makes it possible to interact with the running program. Any output generated by the program (usually through the use of the `printf` function) will appear in the console window. When the program needs input values, you will enter these by typing data into the console window. When your program is finished, you select `Quit` from the `File` menu to return to the editing environment. When you quit, the console gives you a chance to save its contents into a text file:

In general, you don't want to save the output unless a particular assignment requests that this output be handed in to your section leader, (in such a case you will be specifically instructed to do so.) Select `Don't Save`.

Looking at the `add2.c` program

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.c` 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.c` file, you will see the following program display:



Note the addition of the library file called `genlib.h`, which is made available to our program via the line

```
#include "genlib.h"
```

Every program that we write during the quarter will include this library, and it seems appropriate to get into the habit early. This program also uses `simpio.h`, which defines the `GetInteger` function (and others that simplify input/output). Note that for programs that include `genlib.h`, the proper heading for the function main is

```
main()
```

The examples in the text use this convention.

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 pulled down the `Project` menu and selected `Run`, 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 ten seconds* or so, and you can

* Actually, the very first time you compile a CodeWarrior project it will take much longer, potentially

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 a copy of the output, and go on to the next problem.

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. Don't despair. Making such errors seems to be an unavoidable part of programming. When those errors occur, it is time to sit down, find those errors, and fix them up. This is the process of *debugging*.

Debugging syntax errors

The errors that you are likely to encounter fall into two principal classes. *Syntax errors* are those in which you violate some linguistic rule. As the C compiler goes through your program to translate it into its machine-language form, it has encountered some piece of the program which 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.

CodeWarrior is quite good at helping you find your syntax errors. When the compiler encounters one, it opens an editor window on the offending program and highlights the line where the error was detected. It also displays an error dialog with a message intended to guide you to the source of the error. When you're new to programming, this message may appear obscure, but you will become more used to them as you go along. When you have read the error message, click anywhere in the error dialog to make the box disappear. You can then go back to your program, find the error, and make the necessary correction. 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 *logic errors* or, more commonly, *bugs*.

Debugging logic errors is a skill that comes only with practice. We will talk extensively about strategies for debugging throughout this course, but it is never too early to learn the most important rule about debugging:

> *In trying to find a program bug, it is far more important to understand what your program <u>is</u> doing than to understand what it <u>isn't</u> doing.*

---

several minutes, while it builds a cache to enable all successive compilations to be faster.

Most people who come upon a problem in their code go back to the original problem and try to figure out why their program isn't doing what they wanted. Such an approach can, in some cases, be helpful, but it is more often the case that this kind of thinking can make you blind to the real problem. If you make an unwarranted assumption the first time around, you may make it again, and be left in the position that you just can't see why your program isn't doing the right thing.

When you reach this point, it often helps to try a different approach. Your program is doing something. Forget entirely for the moment what it was supposed to be doing, and figure out exactly what is happening. Figuring out what a wayward program is doing tends to be a relatively easy task, mostly because you have that computer right there in front of you. If you can't understand what is happening in your program, go back into the editor and add `printf` calls in those areas that seem to be implicated in the problems. These statements may be as simple as

```
printf("I got to here\n");
```

or they may be used to display the values of variables, as in

```
printf("The value of total is %d\n", total);
```

Run your program again and watch what happens. Often, the text generated by the `printf` calls gives you a significant insight into what is going on. And, once you understand what is going on, it is usually not too hard to figure out how this deviates from the intended purpose of the program.

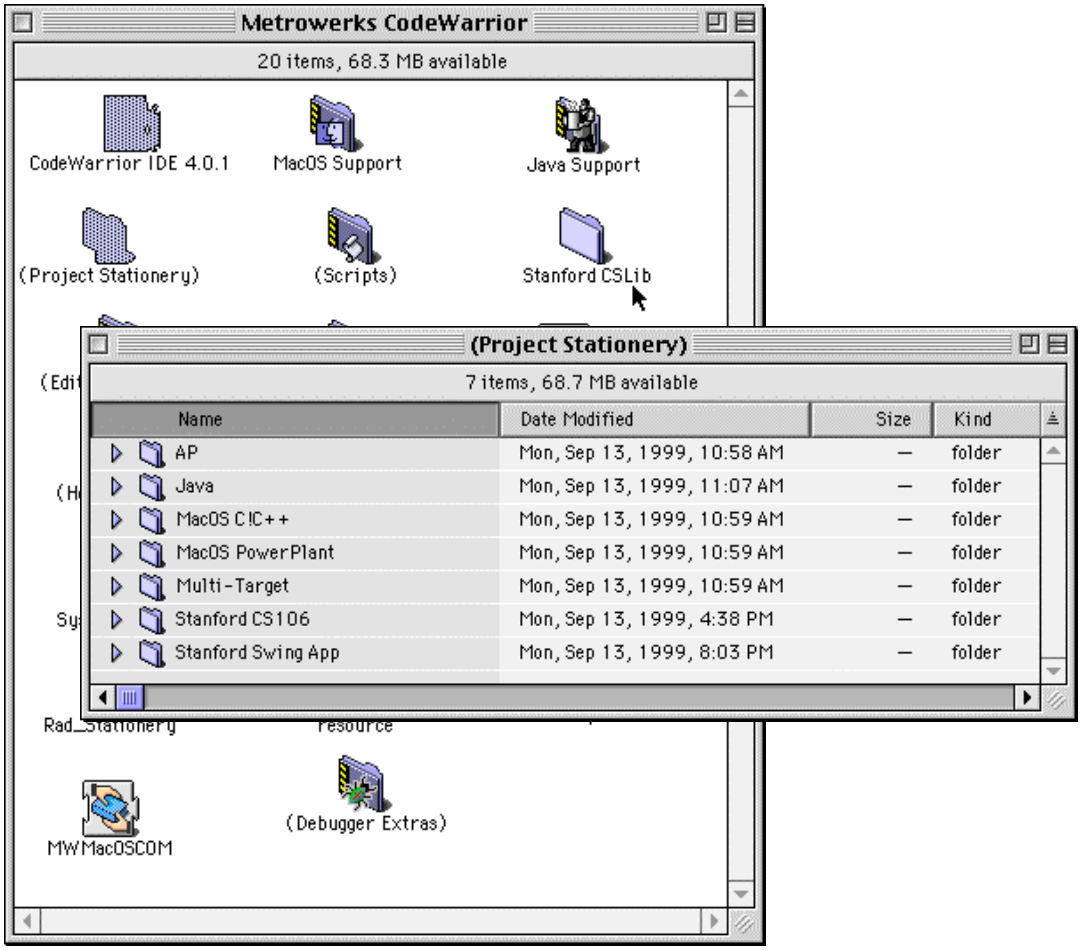
We will talk more about debugging as the quarter progresses.

Special reminder for people using Macintoshes outside of Tresidder

If you have not done so already, you will probably need to install special libraries for using CodeWarrior with CS106. Looking back at the `"CS Libraries for CW Pro 5"` folder you originally downloaded, perform the following steps:

- Move the contents of the `"-->into Metrowerks Codewarrior"` folder, not the folder itself, into the "`Metrowerks CodeWarrior`" folder on your hard drive. `"Metrowerks CodeWarrior"` is located inside the `"CodeWarrior Pro 5"` folder.
- Move the contents of the `"-->into Project Stationary"` folder, not the folder itself, into the `"(Project Stationery)"` folder on your hard drive. `"(Project Stationery)"` is located in the `Metrowerks CodeWarrior` folder inside the `"CodeWarrior Pro 5"` folder.
- If contents found in either of these folders already exists in the appropriate places on the machine you're using, it means that someone else has probably already done these steps for you and you can go on with your life ☺

When you are finished, the affected folders should look something like the ones shown on the next page:


Metrowerks CodeWarrior
20 items, 68.3 MB available
CodeWarrior IDE 4.0.1
MacOS Support
Java Support
(Project Stationery)
(Scripts)
Stanford CSLib
(Project Stationery)
7 items, 68.7 MB available
Name
Date Modified
Size
Kind
AP
Mon, Sep 13, 1999, 10:58 AM
folder
Java
Mon, Sep 13, 1999, 11:07 AM
folder
MacOS C!C++
Mon, Sep 13, 1999, 10:59 AM
folder
MacOS PowerPlant
Mon, Sep 13, 1999, 10:59 AM
folder
Multi-Target
Mon, Sep 13, 1999, 10:59 AM
folder
Stanford CS106
Mon, Sep 13, 1999, 4:38 PM
folder
Stanford Swing App
Mon, Sep 13, 1999, 8:03 PM
folder
(Debugger Extras)
MWMacOSCOM