An Introduction to
Modern Software Development Tools
Apple's XCode Version 3.2.6

Charles J. Ammon / Penn State
August, 2011

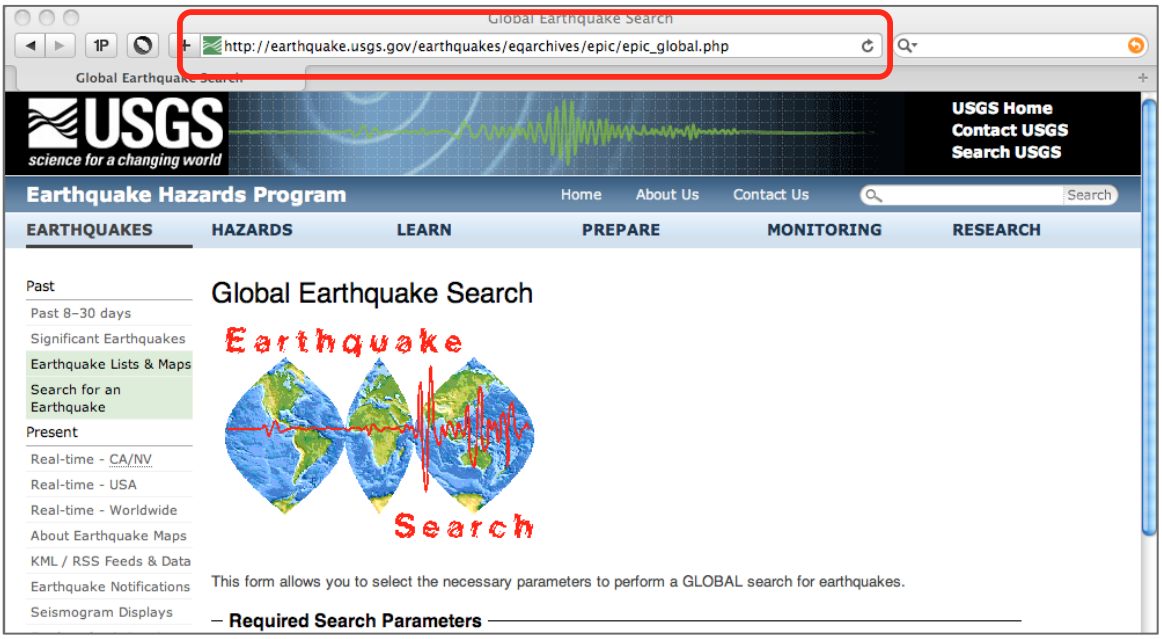
Most seismology that I am familiar with is performed on UNIX compatible computers (with the exception of some high-performance systems used for the solution of differential equations). So in this brief introduction to modern development environments I will focus on Apple's OS X tool, Xcode. I know from talking with students that the modern tools that are provided by companies such as Apple are confusing for beginners. These tools offer a tremendous number of features, the trick is to become comfortable with the simplest features first, then move onto more complex. In one hour we are not going to exercise much of Xcode, but we will construct a simple program that can be run from the command line (like much seismic software).

Getting Some Data To Work With

One of the most common things that I do as an earthquake seismologist is search for hypocenters using the US Geological Survey's Earthquake-Search Web Tool. Sometimes I'm interested in the hypocenters themselves, other times I want to use them as sources for data requests related to earth-structure investigations. The point is I often start an investigation with the USGS hypocenters.

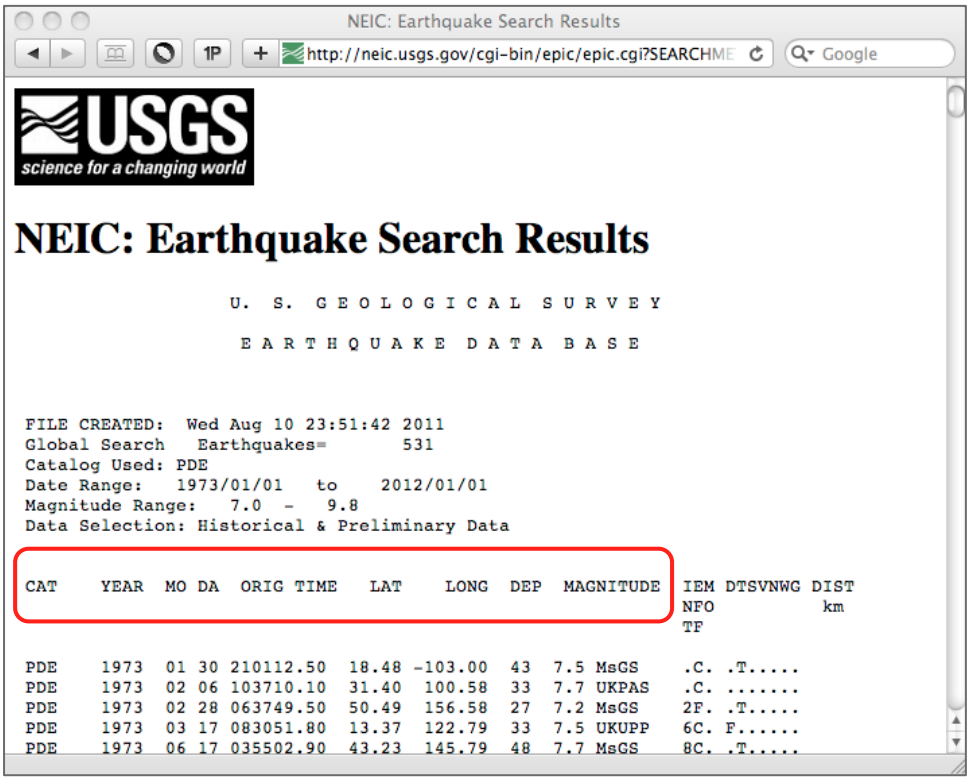
The Figure below shows an image of the site that I mean, and the web address is highlighted with a red box. The address is (perhaps you can copy and paste it from this PDF)

http://earthquake.usgs.gov/earthquakes/eqarchives/epic/



Point a web browser at this site. We are going to perform a global search for all events with a magnitude greater than 7.0 since 1973. Fill in the fields on the web browser to complete the search. You need to specify the start and end dates, and the minimum (7.0) and maximum magnitudes (use 9.8). Then click “Submit Search”.

The search-output file format is relatively simple. The list contains a set of standard fields that we are going to read into a simple C program, and print them to the screen. This is an oversimplified problem, but our focus here is on the Xcode development environment, not the program. The fields that we will parse are those within the red box in the figure below.



Copy the text output into a text editor and save the results in a file called myHypocenters.txt. For now save the file on your computer's Desktop. We'll move it later. Start with the hypocenter lines – ignore the lines that summarize the search and the column headings. The start of your file should look like:

```
PDE    1973  01 30 210112.50  18.48 -103.00  43  7.5 MsGS    .C. .T.....
PDE    1973  02 06 103710.10  31.40  100.58  33  7.7 UKPAS   .C. .......
PDE    1973  02 28 063749.50  50.49  156.58  27  7.2 MsGS    2F. .T.....
PDE    1973  03 17 083051.80  13.37  122.79  33  7.5 UKUPP   6C. F......
```

One of the nice features of C is the fact that it has constructs called structures (so does MATLAB), which are a step towards objects in object-oriented languages such as C++, Objective-C, Python, and Java. Here is the structure that we are going to use:

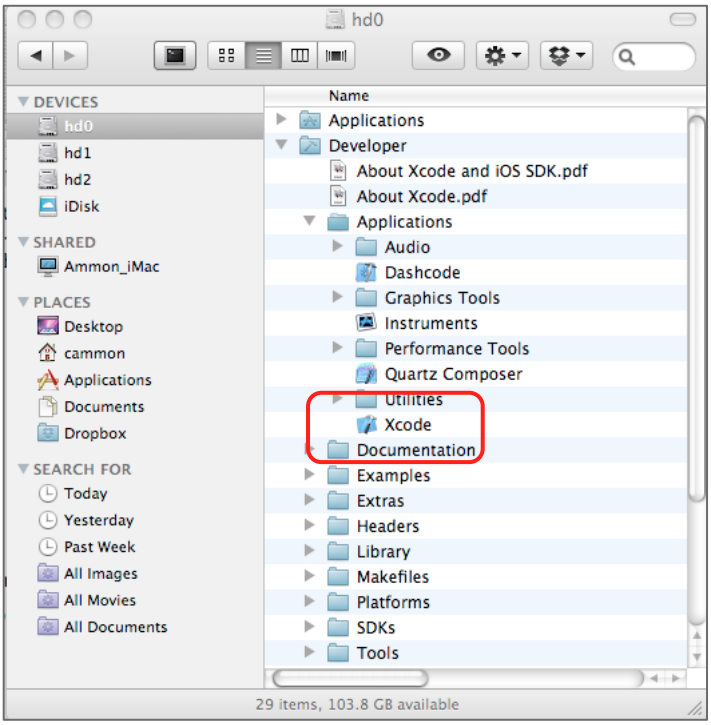
```
struct seismicOrigin
{
    char catalogName[8];
    int  year;
    int  month;
    int day;
    int hour;
    int minute;
    float seconds;
    float latitude;
    float longitude;
    float depth;
    float magnitude;
```

```
        char  magType[8];
    };
```

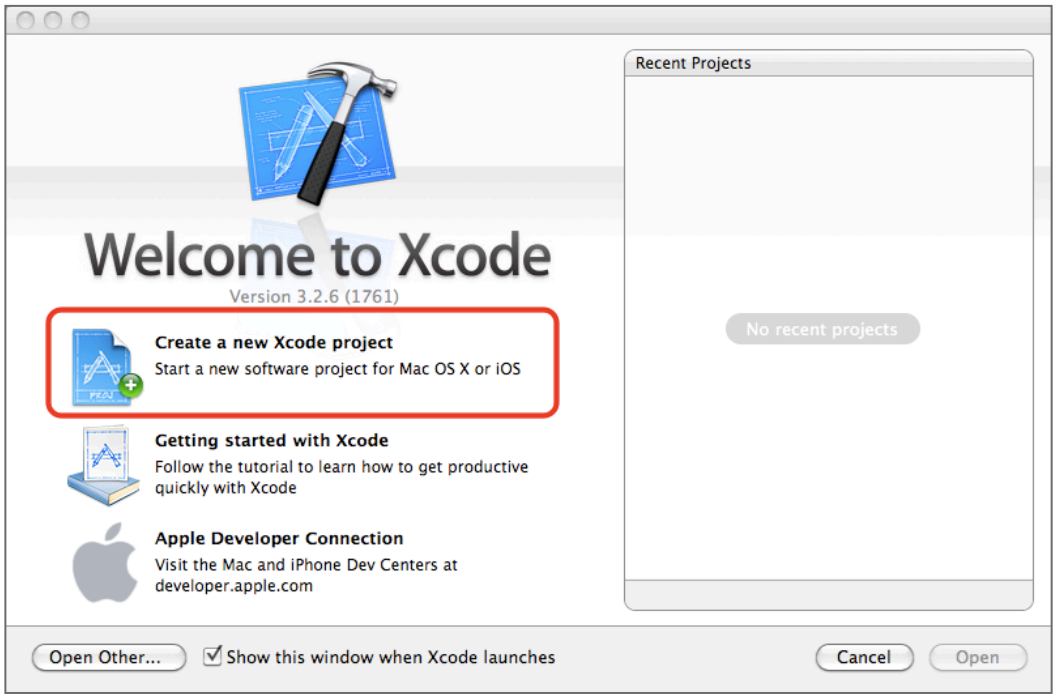
The structure is called a seismicOrigin, and we will read each line of our output file into a new seismicOrigin. We'll store the information for all the earthquakes in an array of seismicOrigins. This is convenient, but again, our focus is not on how to do things in C.

Launching Xcode

To get started, launch Xcode, which is located in the Developer Folder on the computer's top-level directory or the System Disk. This is the place where the Applications Folder and the System folder normally reside. The Xcode application is located in /Developer/Applications. On my computer, the top level disk is called hd0 and the location of XCode looks like:



Double-click the Xcode icon to launch the application. You should see a window that looks like the one below. Or you can use Spotlight to launch Xcode, if you are familiar with that approach.

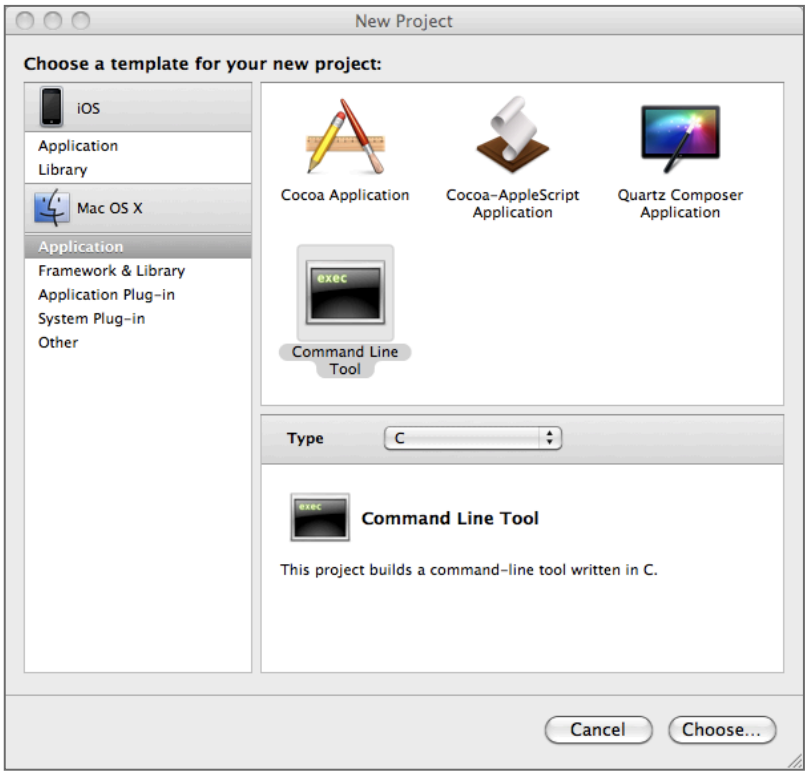


The first thing that we have to do is create an Xcode project.

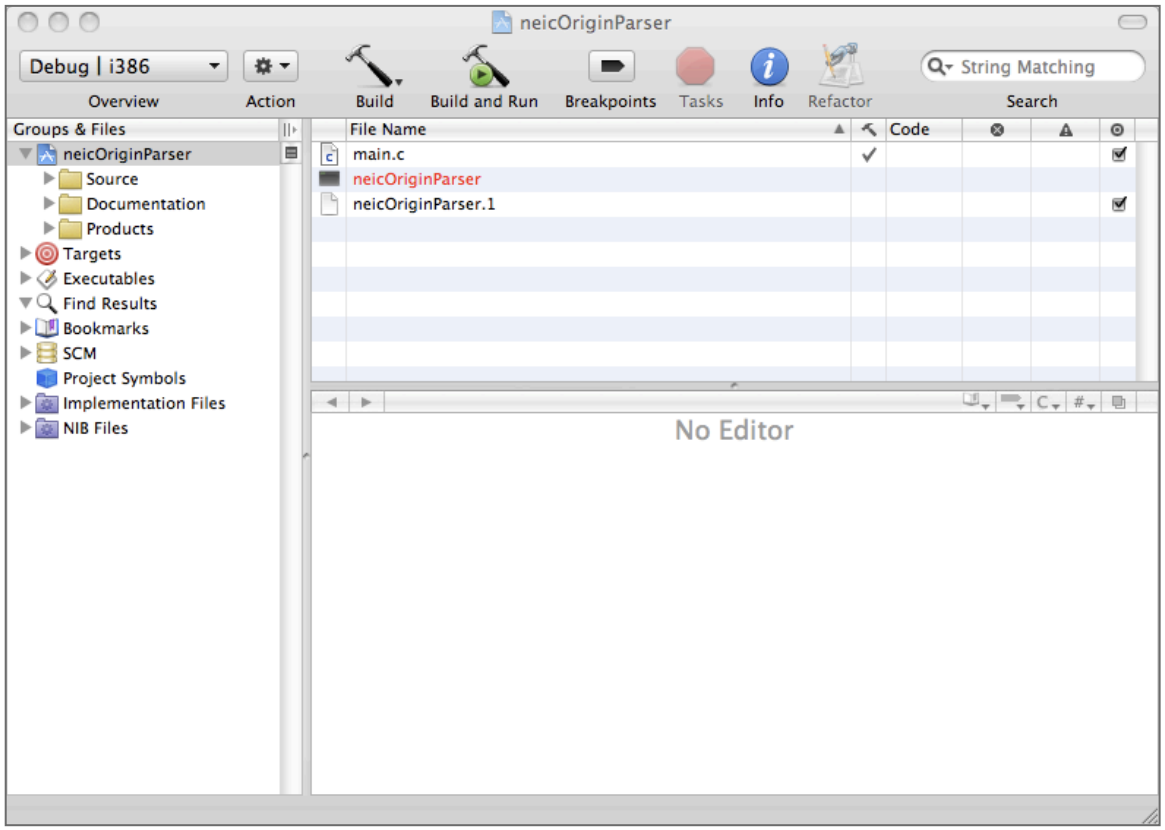
Xcode Projects

Xcode manages codes as parts of "Projects". For now, just consider the case where we have one program in a project (you can package libraries and programs together). The Project is where you store all the files related to the program – the C-language source code, the C-language include files, the compilation parameters, the compiled codes; everything related to this program. We are going to create the simplest of programs, a command-line tool (a program that you execute using the command line.

Choose to create a new project from the splash screen, or create a new project by selecting New Project from Xcode's File menu. Select Mac OS X Application from the column on the left and then select Command Line Tool from the palette of application types that appears. Create a new folder to contain the project – I'll assume that you place is on the Desktop. Although you can name the project whatever you want, for the sake of this exercise, call the project neicOriginParser.



The project template is simple, and includes one source code file, main.c, and a template for a UNIX style "man" page ReFormatNEICOrigins.1, which is located in the Documentaion Folder. We won't worry about the man page, but this is a handy feature. Open the Source folder in the column on the left by clicking the small triangle to the left of the Source folder icon. Examine main.c by selecting it (one click) in the file list on the left. The file will open in the Xcode source-code editor pane.



You can run the template code by clicking the Build and Run button in the Xcode window. When you do so, Xcode will compile the source code, main.c, using gcc, and then run the code. The results are shown in the subwindow of Xcode – the template code simply prints the string "Hello World!" to the terminal. If you don't see the output, choose Console from Xcode's Run Menu. Note that you can resize the various window panes in the Xcode window. At some point you will want to increase the size of the editor pane (where the source code is listed), so that you can work with the code that we plan to create into the editor. We are simply going to replace all the text in main.c with

```
#include <stdlib.h>
#include <stdio.h>

struct seismicOrigin
{
    char catalogName[8];
    int  year;
    int  month;
    int day;
    int hour;
    int minute;
    float seconds;
    float latitude;
    float longitude;
    float depth;
    float magnitude;
    char  magType[8];
```

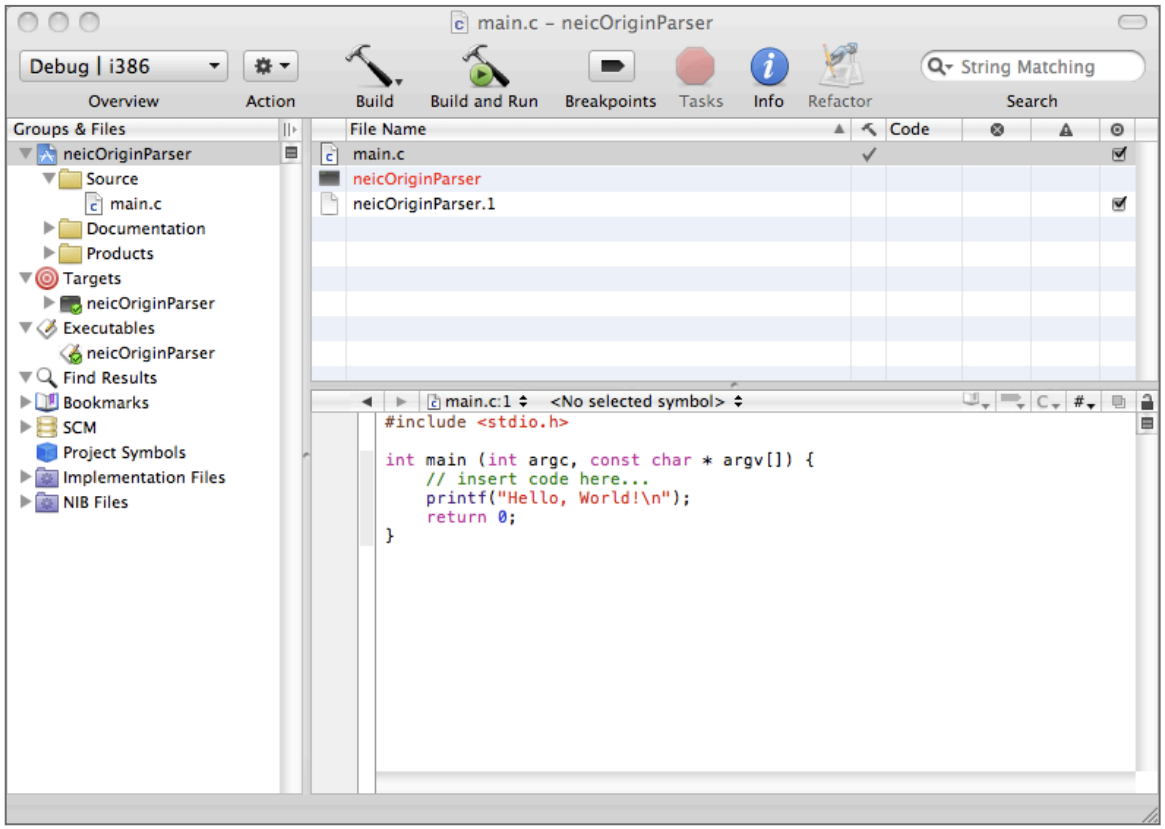
```
};

int main (int argc, const char * argv[])
{
    FILE *fd;
    int nread, ok, count;
       char dummy[64],timeString[32];
    struct seismicOrigin theOrigins[2048], anOrigin;

    fd = fopen("myHypocenters.txt", "r");
       if(fd == NULL)
       {
              fprintf(stderr,"Problem reading the input file!");
              exit(-1);
       }

       ok = 1;
       count = 0;
       while(ok>0)
       {
              nread =
fscanf(fd,"%s %d %d %d %9s %f %f %f %f %s %s %s",
                                          &anOrigin.catalogName[0],
                                          &anOrigin.year,
&anOrigin.month, &anOrigin.day, &timeString[0],
                                          &anOrigin.latitude,
&anOrigin.longitude, &anOrigin.depth, &anOrigin.magnitude,

&dummy[0],&dummy[0],&dummy[0]);

              sscanf(&timeString[0],"%2d",&anOrigin.hour);
              sscanf(&timeString[2],"%2d",&anOrigin.minute);
              sscanf(&timeString[4],"%f",&anOrigin.seconds);

              if(nread < 1) ok = -1;

              if(ok > 0)
              {

       printf("%4d %2.2d %2.2d %2.2d %2.2d %6.3f %9s %7.3f %7.3f %
5.1f %4.1f\n",
                                 anOrigin.year, anOrigin.month,
anOrigin.day, anOrigin.hour, anOrigin.minute,
                                 anOrigin.seconds, timeString,
anOrigin.latitude, anOrigin.longitude, anOrigin.depth,
                                 anOrigin.magnitude);
                           theOrigins[count] = anOrigin;
                           count += 1;
              }

       }

       fclose(fd);


       return 0;
}
```

Hopefully you can copy and paste the source, otherwise we can mail it during this activity. Take a few minutes to read the code after you copy and paste it into the Xcode editor pane. Don't worry about the strange syntax if you are unfamiliar with C. This program is very simple, it opens a file called myHypocenters.txt, reads each origin from each line in the file and stores the results in an array of seismicOrigin structures. It also prints some of the information about each origin to the screen as it progresses through the input file. When it is done, it prints out how many origins it read, and closes the input file.

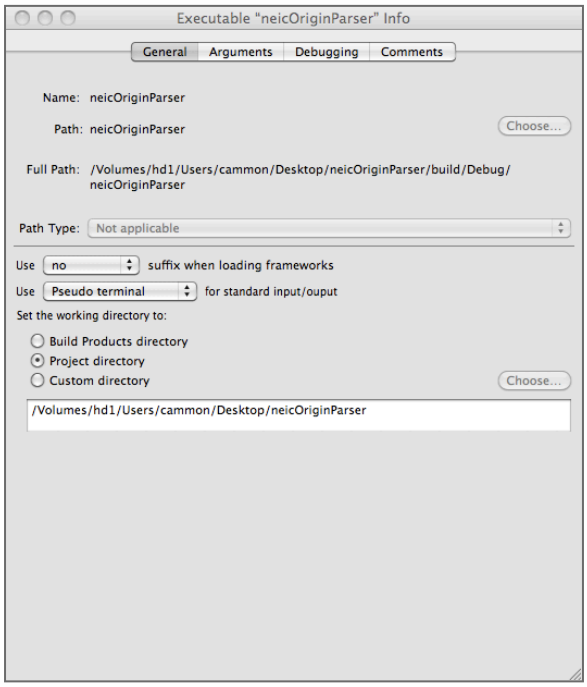
Build and Run the program. You should see a statement that says that the input file cannot be found. If you don't see the output, choose Console from Xcode's Run Menu. That's because we haven't told Xcode the path of folder we would like it to execute the program in when we click Run.

Obviously, our program needs to be able to find our input file. Let's set that up. For convenience, move the USGS hypocenter file that you downloaded earlier into the folder than contains the Xcode sources. The name of the file has to match the name in the C source code, myHypocenters.txt. You don't have to move the data file into the project folder, it's just the way I did it to keep the files together.

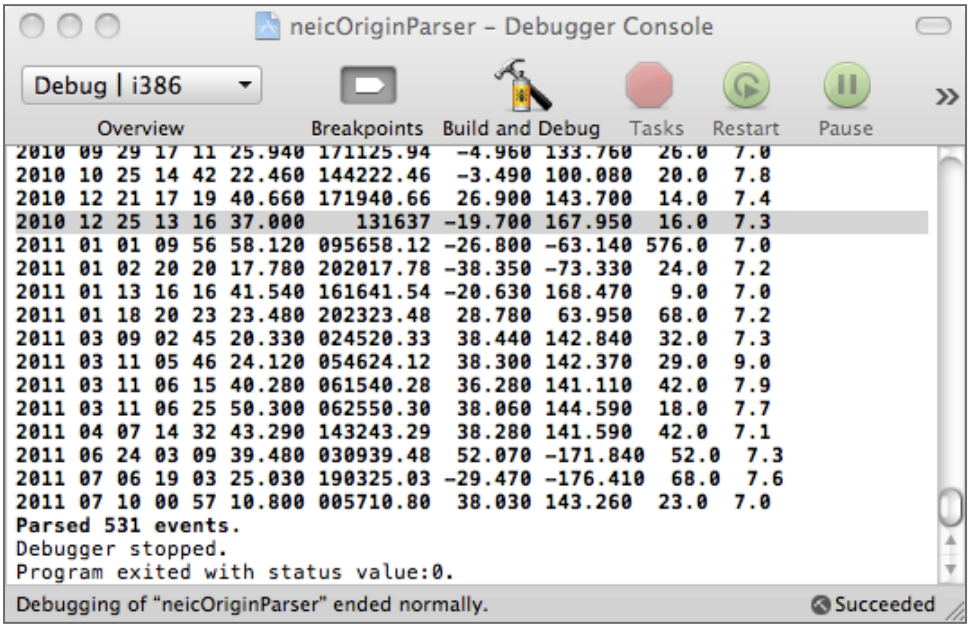
Now we need to set the path for the program's execution in Xcode. Make sure that you can see the Executables list in the left column of the Xcode window (see below). Select the executable (click once on neicOriginParser) and then click the blue Info Button on the Xcode toolbar (the top of the window).



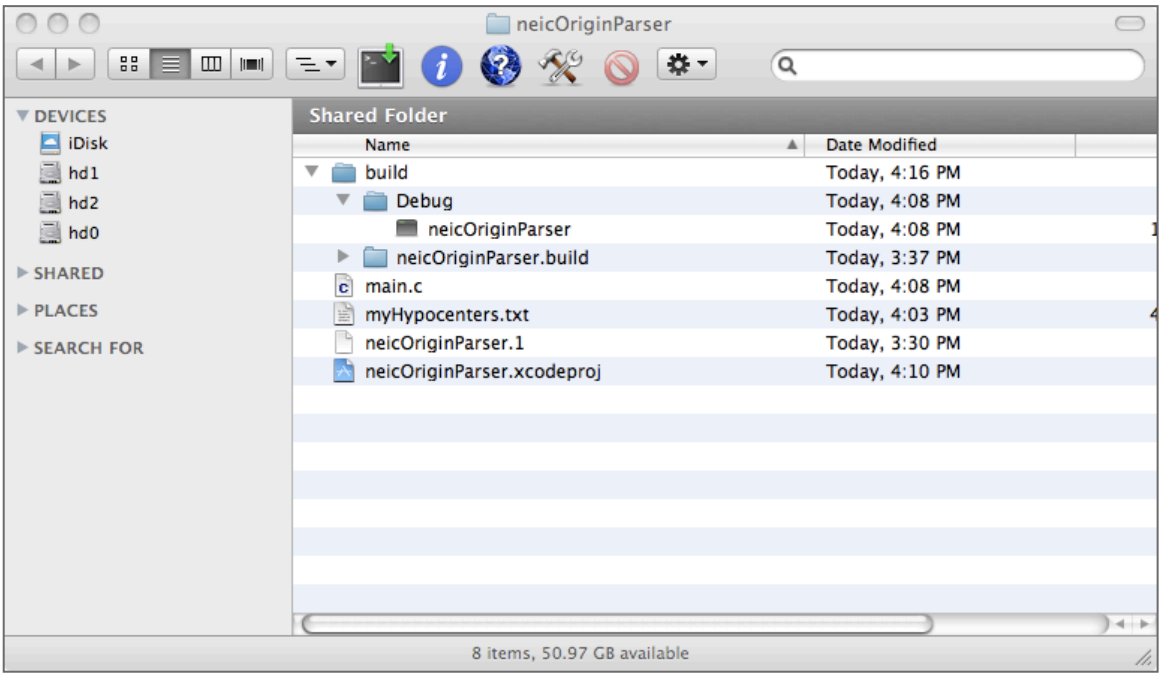
You should see a window that looks like the one below. Just beneath the Pseudo terminal popup menu is where you tell Xcode where you want to run the program when you click the Run or Build and Run button. Choose the Project Directory as the "working directory". That's where we just placed my data. Check that the path XCode lists when you select this option, then close the information window.



Now click Build and Run or Build and Debug in the toolbar along the top of the Xcode window. The program should run. It takes almost no time. If you don't see the output, choose Console from Xcode's Run Menu. Here's the output that I obtained.



Congratulations, you just built a command-line tool with Xcode. You can find the executable in the Project folder (see below). You could move the executable to any folder that you want to use it - /usr/bin, ~/bin, etc.



Command-Line Completion

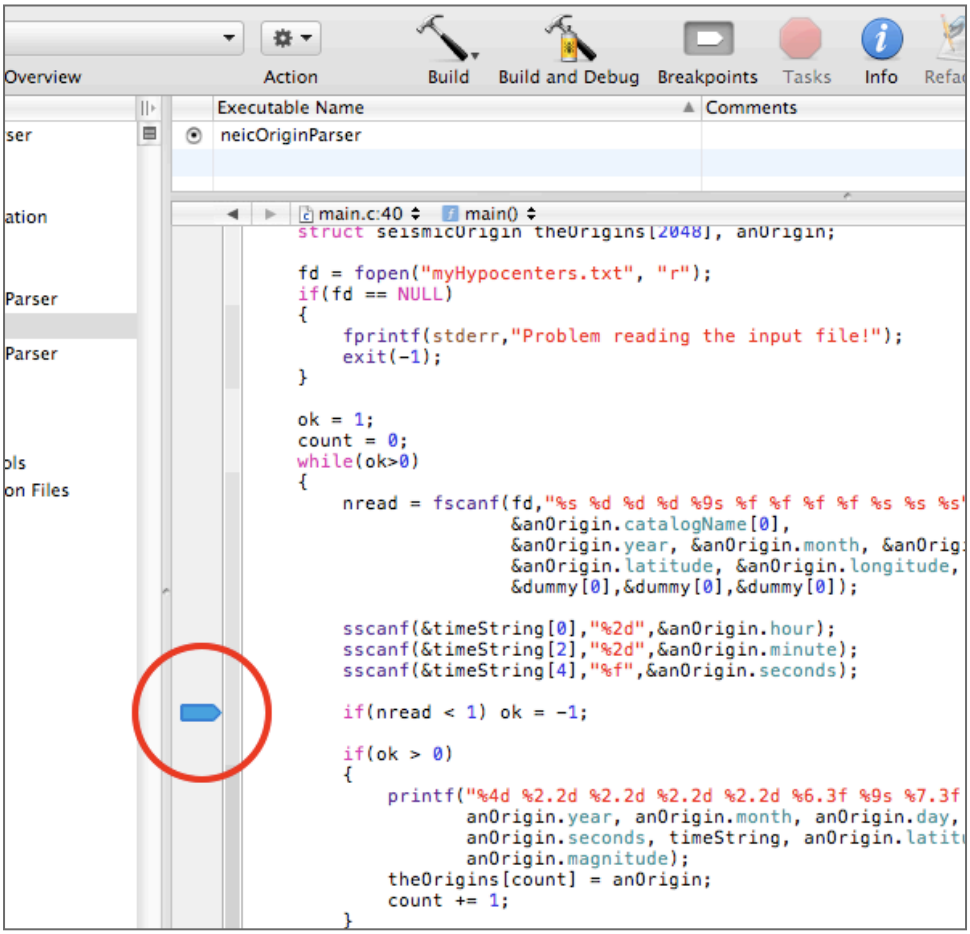
Let's add one line to the program because I want you to see how much Xcode helps you write your code using command-completion. Find the line near the end of main.c that contains fclose(fd);. Click at the end of the line using the mouse and then press return to add another line. Note that XCode automatically indents the new line to make the code more readable. Now slowly enter the line

```
fprintf(stdout,"Parsed %d events.",count);
```

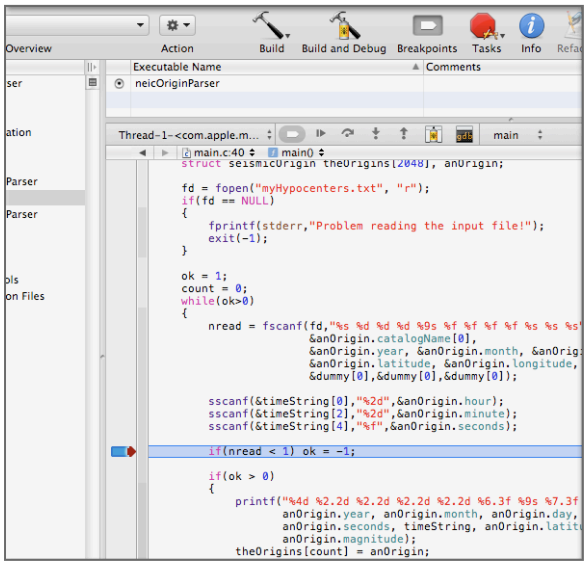
As you begin to type, XCode will guess what you are about to type and present you with completion-options. To accept the completion, just enter a "tab"; to dismiss the suggestions, enter "esc". If you accept the suggestion you can tab to the arguments in the completed text. Command-line completion is most useful when you know the start of a function call but may not remember all the details of the call. When you start using large libraries, this feature is indispensable. XCode will also complete variable names from within your code, saving you time and preventing many typographical errors.

Elementary Xcode Debugging

If that was all that Xcode did well, it would be a terrible waste of time to learn it. Of course it does much more. I'll finish by introducing the ever-important debugging features. Some of the most important items in debugging are breakpoints, which allow you to pause the program execution and examine the values of the variables in the program. To set debugger break points in Xcode, you just click to the left of a line of code. To clear breakpoints, you just drag them out of the region to the left of the code. Look below and then try to set a break point at the same location by clicking to the left of the source code.

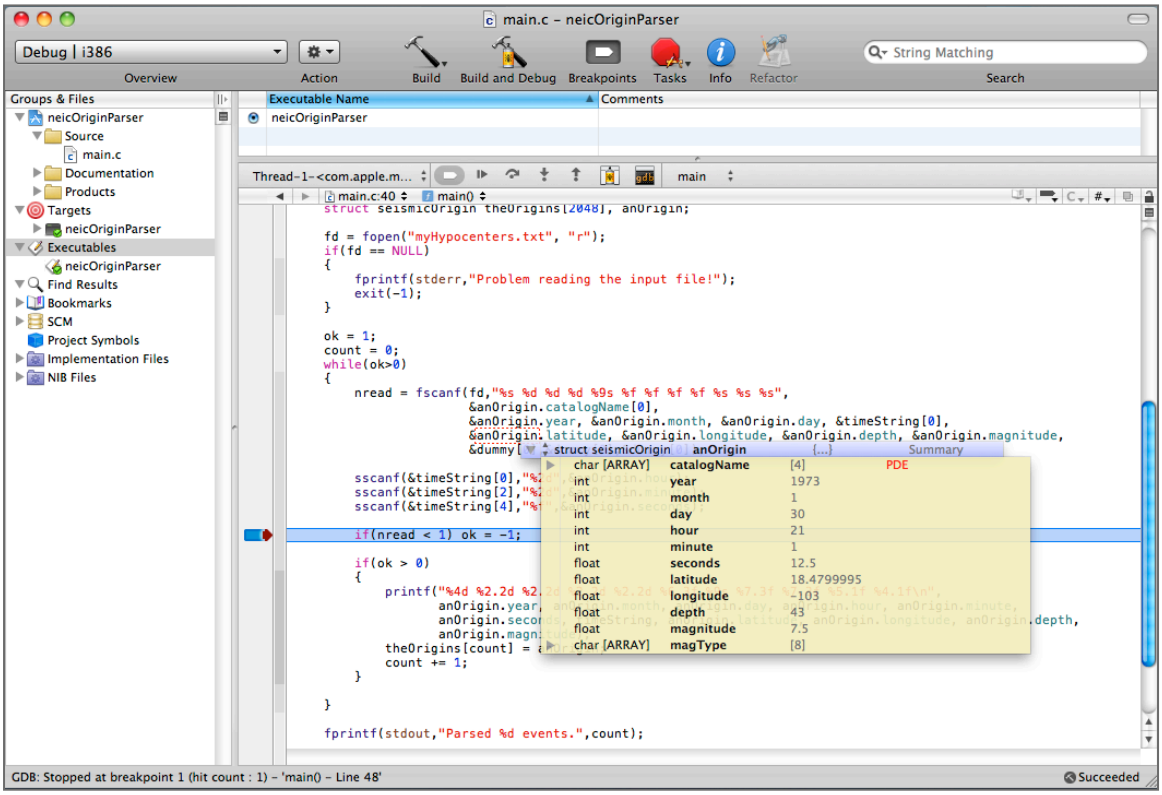


The blue arrow indicates that we have a breakpoint at the line that checks whether the read statement worked. If you Build and Run the program it will pause at that line and wait until you tell it to continue.



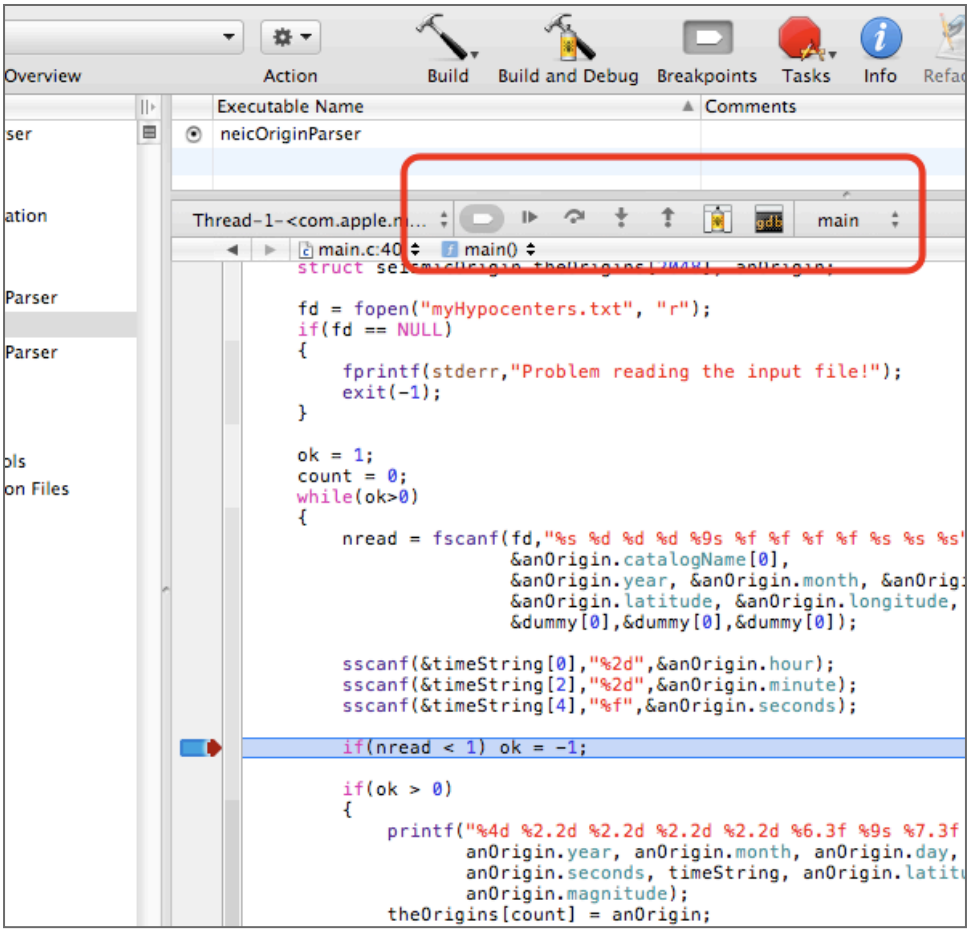
255

256 While it's waiting for you to tell it what to do, you can examine the values of the
257 variables in the code by hovering the mouse over the items of interest. Hover
258 over the variable anOrigin and then move to the discloser triangle in the yellow
259 box that opens. You should see something like:



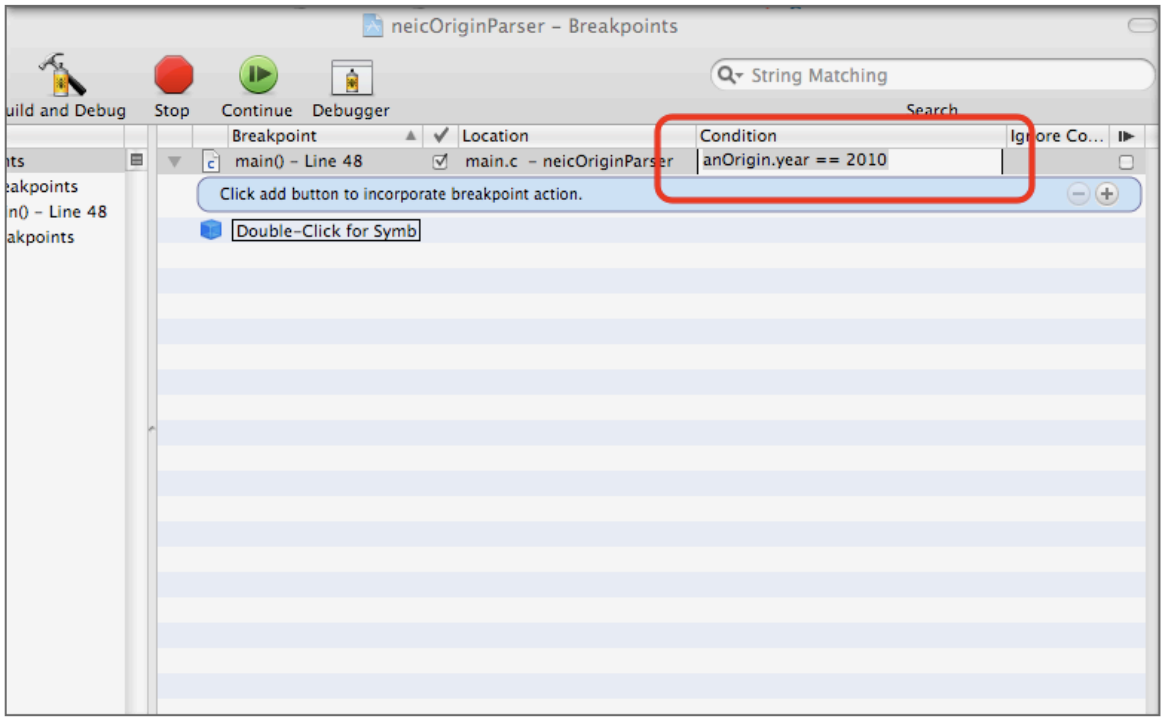
260

The debugger controls are along the top edge of the editor pane. The first icon lets you toggle all breakpoints off and on; the second one continues on from the stopped line; the third executes the next line and pauses, etc. To continue executing the program, click on the second icon, which looks like a “play” button in the debugger controls.



A breakpoint can be more sophisticated. Suppose that we wanted to pause only when the year was 2010? Then we can change our breakpoint to be conditional. If you click once on a breakpoint, it will turn light blue, indicating that it is suspended. Double-click on the blue breakpoint indicator to open the breakpoint editor.

In the illustration below, I’ve set a condition to pause at this breakpoint only if anOrigin.year equals 2010. You edit the condition field by double-clicking in it (like any other text field). That’s a conditional breakpoint. Try it out.



The default settings for a new project create a debugger-compatible executable that includes information for tracking the execution and is a little larger and a little slower than a "release" version of a program. For the utility that we just developed this is not a big issue, it takes little time. For larger, more intensive calculations, optimizing the executable to run fast is a wise decision. You do that by choosing the "Release" version options (a popup menu in the XCode toolbar).

Learning More

We'll use Xcode again in the next section to create a simple calculator-like application with almost not code by relying on the vast frameworks (libraries) that Apple provides. Most of the documentation for Xcode and for these frameworks is electronic and supplemented by tutorials, videos, and screencasts from all over the web. It takes time to learn the details; but if you think about it, it can't be too hard since there are so many people writing iOS applications.