

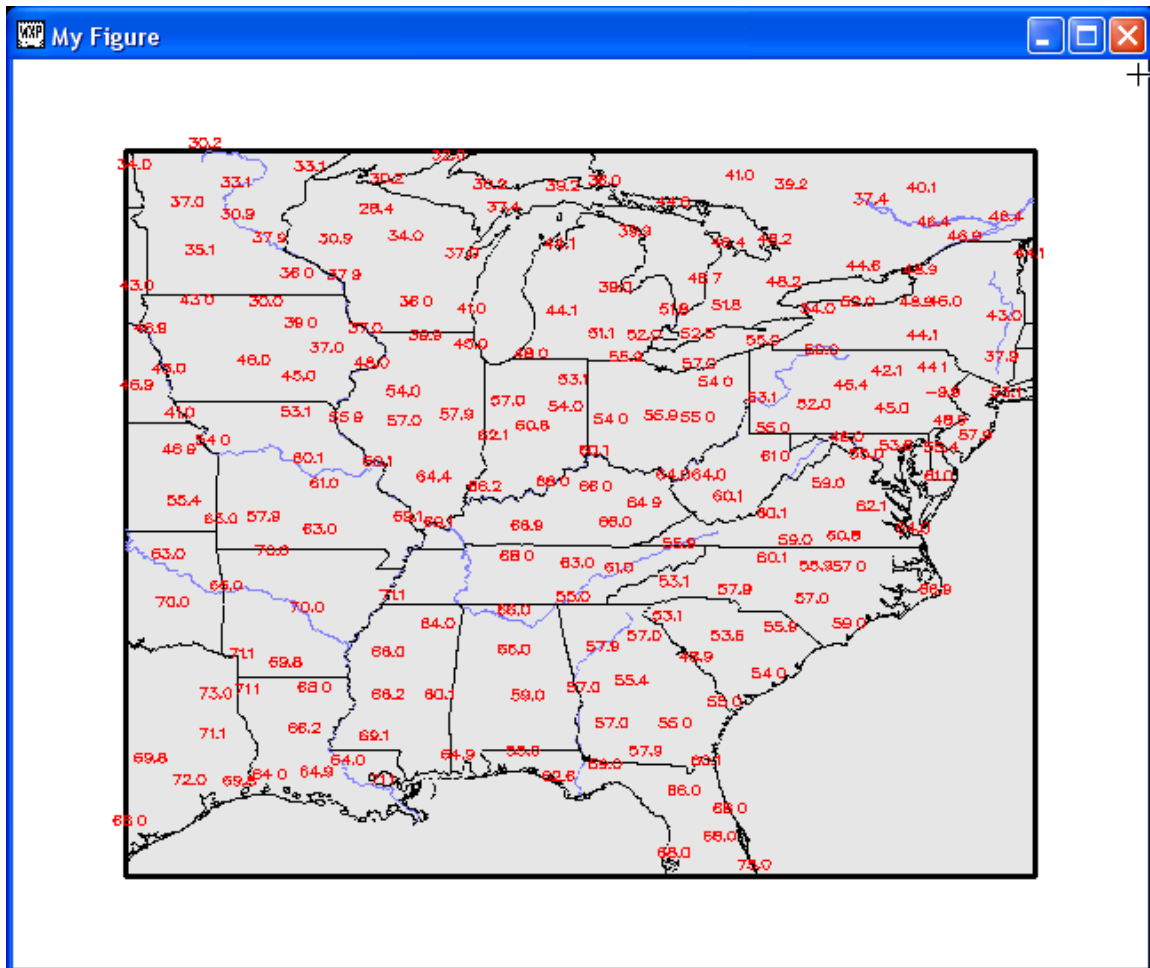
Chapter 19: Plotting Weather Information

We've learned how to draw a map in the graphics window. We've learned how to read and process surface weather information. It's time to put these two concepts together and start plotting weather information onto the maps!

Here is how I want your program to perform:

1. The user starts the program.
2. The program prompts the user to ask which variable to plot on the map (temperature, pressure, dewpoint, wind chill, whatever).
3. A map is drawn.
4. Numbers are plotted on the map, indicating the temperature (or pressure, or wind chill, or whatever) at each station.

Your final plot is going to look something like this:



The sao.easy.cty File

To plot the weather data at the location of your city, you are going to need to know where each city is—the latitude and the longitude of the city. This information (and more!) is available in a file called `sao.easy.cty`, which is located in the `/home/schragej/ats315` directory.

There are 6893 stations in the `sao.cty` file. Here is a sample of the data in that file:

```
SBMQ 1    0.03  -51.05
FOOL 1    0.45   9.42
SELT 1   -0.92 -78.62
MROC 1   10.00 -84.22
SVMI 1   10.60 -66.98
VVTS 1   10.82 106.67
SLCO 1  -11.08 -68.87
TGPY 1   12.00 -61.78
SPIM 1  -12.00 -77.12
FTTJ 1   12.13  15.03
MNMG 1   12.15 -86.17
YPCC 1  -12.18  96.82
YPDN 1  -12.40 130.87
```

Each of the 6893 stations is described on exactly one line in the `sao.easy.cty` file. There are several pieces of information about the weather station:

- The four-digit station identifier.
- The “priority” of the station—bigger, more important weather stations have a *lower* priority, while smaller, less reliable weather stations have a *higher* priority number. The priorities range from 1 to 7 and are always integers.
- The latitude and longitude of the station. Remember that negative latitudes are in the Southern Hemisphere and negative longitudes are in the Western Hemisphere.

Therefore, each time you read in a decoded METAR report for a station, you will need to compare the station identifier to the identifiers given in the `sao.cty` file until you find a match, determine whether or not this station is in your clipping region, and then, if appropriate, plot the weather data at the appropriate location. In other words, you are going to need the information that is in the `sao.cty` file many hundreds—maybe thousands!—of times (i.e., once for every observation in the current `*_sao.wxp` file). Needless to say, you are *not* going to want to read this large `sao.cty` file thousands of times each time you make a plot. Rather, we should read the information in `sao.cty` *once* and store the information in memory as a table, or “array”.

Arrays

Computers store tables of information in “arrays”. You can have an array of integers, an array of floats, an array of strings, or whatever. (However, each array can only hold *one* type of information—you can’t have an array that contains some floats, some integers, and some characters, for example.)

Arrays have to be declared, just like any other kind of variable in C. When you declare an array, you need to tell the program how large the array is going to be, so that it sets up a large enough area of memory to store the area. For example, consider this array declaration:

```
int priority[6893];
```

This declaration sets up an array of integers called “priority”. There are 6893 integers in this array. (Of course, there aren’t *any* integers in this area yet—maybe it would more accurate to say that there is “room for” 6893 integers in this “currently empty” array.)

Each “element” of the array is an integer, and each one can be accessed by putting a number (or an integer variable!) inside of the square brackets. For example, to set the first element of the priority array to 3, you could enter:

```
priority[0] = 3;
```

(Notice that the elements of an array with N elements are numbered from 0 to N-1.) Or you could prompt the user to enter the value for the first element of the priority array:

```
printf ("Enter the priority for the station:\n");
scanf ("%d\n",&priority[0]);
```

(Notice this code will enter a value into the first element of priority.)

What’s really useful about arrays, however, is that the “indices” of the array can be integer variables. These integer variables are typically part of a loop, thereby processing all of the elements of an array in just a few easy steps. Consider the following piece of code:

```
1     float tempF[5], tempC[5];
2     int i;
3
4     for(i=0;i<5;i++) {
5         printf ("Enter the temperature at time %d.\n",i);
6         scanf ("%f\n",&tempF[i]);
7         tempC[i] = FtoC( tempF[i] );
8     }
```

Consider what this code is doing:

1. Line 1 declares two array of floating point numbers, each containing 5 numbers.
2. Line 4 sets up a loop from 0 to 4, which are the indices of the elements of tempF and tempC.
3. Lines 5 and 5 get the user to enter a total of 5 Fahrenheit temperatures. Each temperature is scanned in and put into one element of tempF.
4. Line 7 uses our old FtoC function to convert each element of tempF to Celsius. Notice that the argument for FtoC is still just one value: tempF[i]. Each time execution of the loop reaches line 7, the integer i has some value (i.e., either 0, 1, 2, 3, or 4) so there is some floating point number at tempF[i].

This program didn't really accomplish all that much, but suppose, instead of 5 elements, there were 500—or 5,000,000!—temperatures to be processed. All of a sudden, this business of arrays is looking pretty useful!

Therefore, in your program, read in the contents of the `sao.cty` file *once*, and store the information from this file in a couple of arrays—an array of 4-digit strings to hold the identifiers, an array of integers to hold the priorities, two floating point arrays (one for the latitudes and the other for the longitudes), and an integer array to hold the elevations of the station. By the way, an array of 4-digit strings would be declared like this:

```
char id[6893][4];
```

(Perhaps you are thinking that strings are probably just arrays of characters, and you would be right!) Then you can work with the arrays of strings just like you work with any other array. For example, to print the 100th 4-digit station identifier of the `id` array:

```
printf ("%s\n", id[100]);
```

A pseudo-code version of your plotting program will look something like this:

Read in the `sao.cty` file and store it in some arrays.

For each decoded METAR observation in the current `_sao.wxp` file...

 Read in the station identifier and the weather observation.

 Try to match the station identifier with the identifiers in the arrays.

 When you find a match, determine if the station is inside the region you wish to plot.

 If it is, transform the latitude and longitude to (x,y) coordinates and plot the observation.

Common Trick: The Look-Up Table

In this assignment, you are reading through a data file that contains the station identifiers and the weather observations for this station. However, in order to plot this

information onto a weather map, you are going to need to know the latitude and longitude (and priority!) of the station—information that is *not* given in the observations. However, all of this information *is* given in the `sao.cty` file. In other words, you need to look-up the latitude and longitude for each station as you find it in the observations. This is a common problem in meteorological computing, and is solved by creating a “look-up table” of the information in the `sao.cty` file.

There are 6893 stations in the `sao.cty` file. Read in the 6893 station identifiers, latitudes, longitudes, and priorities. These arrays should be declared something like this:

```
char id[6893][10];
float latitude[6893], longitude[6893];
int priority[6893];
```

(Note: Learn from my mistakes. When I tried to make this program, I discovered that some of the station identifiers are more than 4 characters long—some are 5 characters long, and some buoys actually have 6 digits! It seems like the `id` array could be declared `id[6893][4]`, but that caused errors when there were 5 and 6 digit long identifiers. This way the program never crashes.)

Now these arrays form a look-up table. When you are reading the `*_sao.wxp` file, you will read in a station identifier and a weather observation. The trick is that you are going to need to find out where that station is. Here, declare the variables that you read in from the `*_sao.wxp` file something like this:

```
char currentid[10];
int currenttemperatureint, currentdewpointint;
```

Notice that these variables are NOT arrays—at each station, there is just one identifier, just one temperature integer (that needs to be decoded), and just one dewpoint integer (that also needs to be decoded).

To find the latitude and longitude of the station, scan through the entire `id` array, looking for a match:

```
while(!feof(datafile)) {

    /* Here you will read in one observation from the
    *_sao.wxp file */

    for(i=0;i<6893;i++) {

        if(!strcmp(currentid,id[i])) {

            /* Entry number i in the look-up table
            matches the current station. Therefore,
            the coordinates of the station are
            (latitude[i],longitude[i]) and the
            priority of the station is priority[i].
            */
```

```

        /* Here is where you will decide what
        to do with this information. */

    }

} /* next i */

} /* end of while */

```

Common Trick: Reusing Code in Ways Not Intended

One of the trickier things about this assignment is you need to determine whether or not to plot any given station, depending on whether or not the station is inside of your clipping region. This way, you'll be able to (largely) keep the observations from being written into the parts of the graphics window that you need to keep clear for labels and so on.

Once you have the latitude and the longitude, you already have a function that transforms coordinates in the form (latitude,longitude) to the format (x,y). You are going to need this information so that you know where to gprint the observation. However, some observations will lie *inside* the graphics window but *outside* the clipping region. If you recall, that problem was solved for line segments using the clip function that I gave you. Similarly, we can use (some would say “abuse”) this same function to decide whether or not a latitude at (x,y) should be plotted. If you do something like this:

```
clip(&x, &y, &x, &y, a1, b1, a2, b2);
```

(where the coordinates of the corners of the clipping region are specified by (a1,b1) and (a2,b2)), you are “clipping” a line segment from (x,y) to (x,y)—in other words, you are “clipping” a point that just so happens to be exactly at the site of your station. Clipping points is, in and of itself, a pretty dull thing to do, but if the point lies *outside* of the clipping region, the clip function returns the coordinates (-1.0,-1.0) for the endpoints. Therefore, if the results of a “clip” function for the coordinates of your station are (-1.0,-1.0), your station is outside of the clipping region and should *not* be plotted under any circumstance!

This is a good example of “reusing” code in sneaky ways. All computer programmers (but especially meteorologists who are forced to write computer programs) are lazy. Don't work harder than you have to! When I problem seems complicated, try to remember if there is already code that does something similar to what you are trying to do now. Often, with a little imagination and a little modification, you can save yourself a lot of work!

Assignment 12: Plotting weather information

Write a program that performs as follows:

- The user starts the program.
- The program prompts the user to ask which variable to plot on the map (temperature, pressure, dewpoint, wind chill, whatever).
- A map is drawn.
- Numbers are plotted on the map, indicating the temperature (or pressure, or wind chill, or whatever) at each station.

Your program should be interactive—prompt the user to select variables to plot.

To get full credit, your program should not plot stations that lie completely outside of the clipping region. However, it is okay to plot stations where parts of the numbers will lie outside of the clipping region, as shown in the example on the first page of this chapter. (Solving *that* problem is quite complex, but doable—you might even say that I would be *impressed* if you solved this.)

Completing the assignment as shown is worth 85%. Higher grades will be earned by students who impress me.

This is one of the more “creative” projects in this course. There are many ways that you could impress me with your programming skills and creativity:

- Make four-panel charts.
- Give the user choices of colors.
- Give the user choices of “priorities” of stations to plot.
- Give the user choices of domains.

This assignment is worth 10 points.