CSC 180 Assignment 3, Fall 2001

Due at the end of Wednesday, December 5, 2001; no late assignments without written explanation.
Note: It is due on Wednesday because that is the last day of classes.

This assignment involves writing programs to make simple graphs out of characters. For example, a parabola can look something like this:

```
*     |     *
*     |     *
 *    |    *
 *    |    *
  *   |   *
---*******---
      |
      |
      |
      |
      |
```

We will restrict the graphs to containing (unlike the above) a maximum of one point for each y value by representing the functions as formulas of x in terms of y. For example, $x = 3y^2 + 2$ is the kind of function your program can plot.

Your program will plot a graph containing ten rows of characters, with values rounded to the nearest integer. To round to the nearest integer, add 0.5 and call the math library *floor* function.

Your ‘‘plot’’ function will be passed a pointer to a function *f* to be plotted. *f* takes one parameter of type double and returns a value of type double.

The second argument to ‘‘plot’’ indicates the range of the graph by specifying the magnitude of the extreme left and right values; 10 will be the default value for this *size* parameter. Your plotting program will call *f* with parameter values ranging from −*size* to *size*. These represent values of y. Each y corresponds to one horizontal line in a graph such as the figure above.

The returned value is scaled by *size* (so that ±*size* is converted to ±10), then rounded to the nearest integer. If it is less than −10 or greater than 10, no asterisk appears in that plotted line (as in the bottom half of the above figure). Alternatively, *f* may return the special return value −999 to indicate that the argument is out of the domain and no point should be plotted there.

A character cell on a typical text computer monitor is about twice as tall as it is wide. So we will use 21 rows, representing y values from −10 to 10 inclusive (after scaling), but 41 columns (representing x values from −10 to 10 after scaling).

Numbering the columns as follows (read the numbers vertically):

```
00000000011111111112222222222333333333344
12345678901234567890123456789012345678901
                    |         *
```

, we draw a centre-line at column 21, and we see in this example a point plotted at $x = 5$. That is, column i represents an x value of $(i - 21)/2$, for $-10 \le x \le 10$ or $1 \le i \le 41$.

We use ‘−’ characters to draw the x axis and ‘|’ characters to draw the y axis, with a ‘+’ at the origin. However, if an asterisk is to be plotted on one of the axis lines, we output the asterisk *instead* of the ‘−’, ‘|’, or ‘+’ character.

Using loops somewhat like those in the ‘‘triangle’’ example (http://www.dgp.toronto.edu/~ajr/180/example/-3/triangle.c), you can output the appropriate numbers of spaces.

Your program

Program code in a file called ‘‘plot.c’’ will perform the plotting, repeatedly calling *f*, a pointer to which is the first parameter. To declare a first parameter which is a pointer to a function accepting one double parameter and returning double, and a second parameter which is a double which is the ‘‘size’’ parameter, the *plot* definition header will look like this:

```
void plot(double (*f)(double x), double size)
```

To call the function *f* within *plot*, the call will look like this:

```
(*f)(x)
```

where x is the value you wish to pass as parameter.

This is to be written in a file called plot.c. plot.c and all files which call plot() will #include "plot.h", which is supplied in ~ajr/a3. Also in ~ajr/a3 is a test main function which asks *plot* to plot the identity function.

(over)

You should divide the contents of plot.c into multiple functions as a way to organize your program. For example, you might have a function which plots a single line, given parameters such as the horizontal location of the asterisk on that line.

The directory ~ajr/a3 also contains a ‘‘functions.h’’ which declares five functions suitable for passing to *plot*:

- *identity*: the identity function
- *odd*: a function which is the identity function for parameters whose integer part (floor) is odd, and undefined for parameters whose integer part is even (this is boolean-like)
- *square*: a function whose output is the square of its input
- *sine*: a function which produces a sine wave. To make the height of the wave more similar to the period of the sine, the return value is $3 * sin(y)$.
- *cubic*: the cubic polynomial $y^3 - 2y + 1$.

You will write these functions in a file ‘‘functions.c’’, which will #include "functions.h" for type checking. Any file using these functions will also #include "functions.h".

Now, you can write the main program, main.c. It repeatedly reads a line of input with fgets() and acts upon it as follows. (Prompting for this line of input is not necessary, and would mess up the display when it is run by the testing software.)

1) There is a ‘‘quit’’ command which exits the program. The program must also exit upon EOF.

2) Otherwise, the input is any one of the above five names, optionally with a size argument. If the user types ‘‘square 5’’, this is a request to plot the square function with a size argument of 5. If the user types simply ‘‘square’’ (and presses return), this is the same as ‘‘square 10’’.

An appropriate error message should be output for invalid input (but it should be terse). You can optionally implement a ‘‘help’’ command and advise ‘‘Type help for help’’ when giving an error message.

To represent the functions in a flexible way, you will declare an array like the following in your main.c:

```
struct {
    char *name;
    double (*function)(double);
} functions[] = {
    { "identity", identity },
    { "odd", odd },
    { "square", square },
    { "cubic", cubic },
    { "sine", sine },
};
```

Then, when the user types a function name, you will perform a linear search of this list, finding the appropriate function pointer to pass to plot(). (This is part of the assignment and is not optional.)

Note that each of your files plot.c, functions.c, and main.c must work with all correct other implementations of these files; for example, in grading we might compile *your* plot.c with *our* functions.c and main.c. Also, your files must all work with the supplied functions.h and plot.h files, #including them as appropriate.

As the due date approaches, I may post some answers to frequently-asked questions on the course web page.

File submission

When you are done, submit your files for grading. You submit the source code files, not the compiled files. Your files ***must*** have the names plot.c, functions.c, and main.c; note that ‘‘functions’’ is plural. You don’t submit plot.h and functions.h because we have them.

Submit your files with the command

```
submitcsc180f 3 plot.c functions.c main.c
```

You may still change your files and resubmit them with the same command any time up to the due time. You can check that your assignment has been submitted with the command

```
submitcsc180f -l 3
```

This is the only submission method; you do not turn in any paper.