Lab 9: Floating point values & structures

Goals

Demonstrate proficiency in creating functions that perform operations on structures containing double precision floating point values. Understand how to use function from the mathematics library in your program.

Background

Floating point

Unlike int, floating point values can represent fractional entities and are thus more useful in most scientfic computations.

There are two types of floating point variables in the C language.

```
float        32 bits
double       64 bits
```

Example declarations

```
float a;
double c;
```

Floating point constants can be expressed in two ways

```
Decimal number          1024.123
Scientific notation     1.024123e+3
```

```
avogadro = 6.02214199e+23;
```

Format codes for floating point input and output:

There are three basic specifiers e, f, and g. Each *must* be prefaced by the letter *l* if a double precision value is being read or printed. The usage of each of the 6 possible codes in output is shown below.

%f	single precision floating point number *(float)* in decimal notation
%lf	double precision floating point number *(double)* in decimal notation
%e	single precision in scientific notation
%le	double precision in scientific notation
%g	single precision with auto notation selection
%lg	double precision with auto notation selection

The size of an output field and the number of digits to the right of the decimal may also be specified in output format strings. Specifying both field width and decimal digits is useful and often necessary when trying to make output values "line up" properly in columns. The *modifiers should not be used for input.*

Floating point input

For input it is not necessary to distinguish, e, f, and g. Any of the codes can be used to read decimal or scientific input values. It is necessary to use *l* if reading double precesion.

```
int howmany;
double fpvalues[2];

howmany = fscanf(stdin, "%lf %lf", &fpvalue[0],
                                   &fpvalue[1]);
```

Floating point output

```
howmany = fprintf(stdout, "%8.3lf %8.3lf \n",
                          fpvalue[0], fpvalue[1]);
```

`%8.3lf` means print with field width of 8 characters with 3 digits to the right of the decimal

Quads

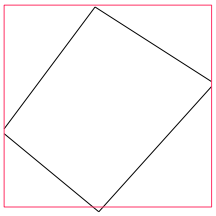
A quadrilateral or "quad" is a four sided closed geometric object. A quad is defined by an ordered set of 4 points where each point consists of an (x,y) coordinate pair.

We will consider quads in the x - y cartesian plane.

We will use the following structure types:

```
typedef struct point_type
{
   double x;
   double y;
} point_t;

typedef struct quad_type
{
  point_t vertex[4];
  double  perimeter;
  point_t bounding_box[4];
} quad_t;
```



The bounding_box of a quad is the smallest axis aligned rectangle that contains the quad. Its vertices are

(minx, miny), (maxx, miny), (maxx, maxy), and (minx, maxy).

where minx is the minimum x coordinate among the 4 vertices of the quad.

The perimeter of a quad is the sum of the lengths of the edges of a quad. The length of any edge of a quad is *sqrt(dx * dx + dy * dy)* where *dx = quad.vertex[(i + 1) % 4].x - quad.vertex[i].x* for i = 0, 1,2,3.

Assignment:

Create a lab9 directory and copy the contents of the labs/lab9 directory to your lab9 directory. The main() function is found in main.c and is already complete. Your mission is to *complete four functions* in the quad.c file.

```
int  quad_read(quad_t *quad);
void quad_perimeter(quad_t *quad);
void quad_box(quad_t *quad);
void quad_print(quad_t *quad);
```

The quad_read() function should read in values comprising the four vertices of the quad. If four vertices (8 values) are successfully read the function should return 4. Otherwise it should return 0. The quad_perimeter() function should compute the perimeter of the quad and store it in the quad structure. The quad_box() function should complete the bounding box. None of these functions should print anything!!

Given the following input:

```
1.0 0.0
2.0 1.0
1.0 2.0
0.0 1.0
```

The quad_print() function should print the input vertices, the perimeter and the vertices of the bounding box in the following format.

```
Input vertices
       1.000    0.000
       2.000    1.000
       1.000    2.000
       0.000    1.000
Perimeter
       5.657
Bounding box
       0.000    0.000
       2.000    0.000
       2.000    2.000
       0.000    2.000
```

Building your program:

To compile your program use the command:

```
gcc -g -Wall main.c quad.c -lm
```

The -lm tells gcc to include the math library in its search for functions that you have called. If you leave it off, you will get an error message of the following form:

```
acad/cs101/labs08/lab9 ==> gcc main.c quad.c

/tmp/ccqoZmZn.o: In function `quad_perimeter':
quad.c:(.text+0x25b): undefined reference to `sqrt'
collect2: ld returned 1 exit status
```

Turn In Work

Show your TA that you completed the assignment. Then turn in your image.c program using the command:

```
sendlab.101.section_number 9 quad.c
```