1995 Programming Comprehensive Exam

TANGRAMS

Prof. John Hughes

DUE: Friday January 27 at 4 PM

Introduction

You may have played with Tangrams as a child: a tangram set consisted of some colored plastic pieces and a booklet. The pieces fit together nicely into a square, as shown in Figure 1, but they could also be arranged into a great many other shapes. The booklet contained the outlines of those shapes, and the challenge was to form those shapes with your tangram set.

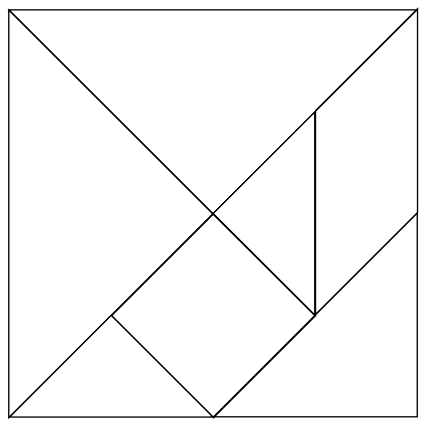

Figure 1.

Just to be clear, here's an exploded view of those pieces:

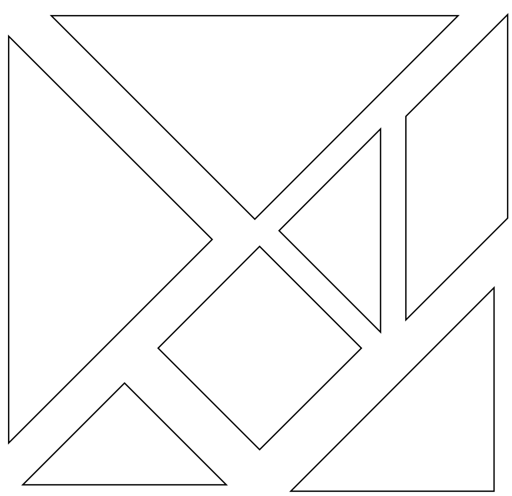

Figure 2.

A typical "problem" in the booklet would be a figure like the one shown in Figure 3:

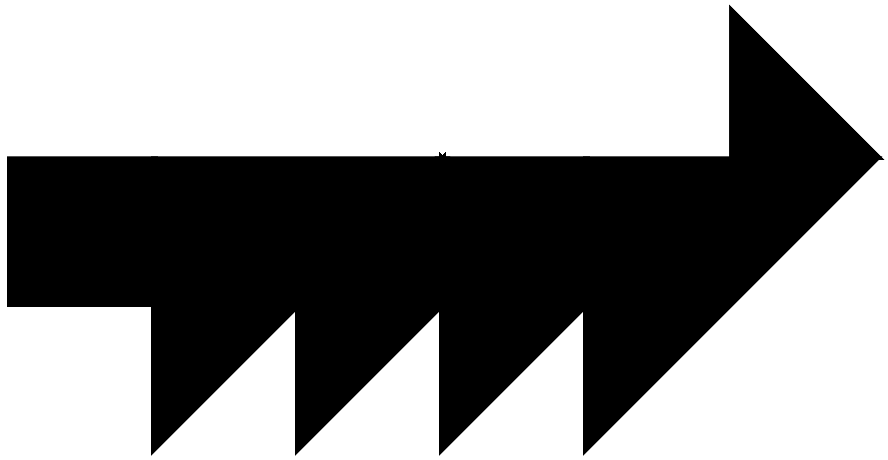

Figure 3.

A "solution" is shown in Figure 4 -- it shows how the tangram pieces can be turned, flipped, and moved around to make the shape shown in Figure 3.

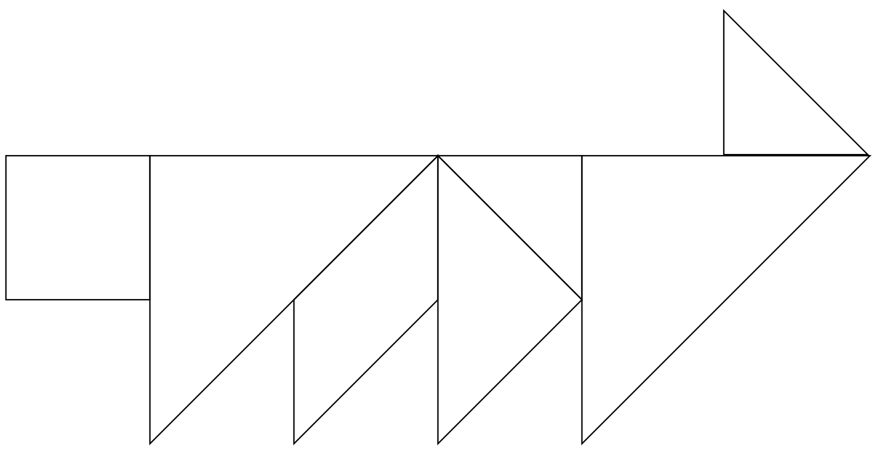

Figure 4.

This year's programming problem is to make a tangram solver. Note that the general problem of fitting together geometric shapes to make more general geometric shapes is probably very difficult, so you'll want to take advantage of the special shapes of the pieces in the tangram set. Nontheless, your algorithm may end up feeling a bit like an exhaustive search, albeit somewhat pruned. Part of doing well on this exam is figuring out how to do the pruning.

Specifications

The tangram pieces are

- Two large triangles, congruent to the triangle with vertices (0,0), (2,0), and (0, 2).
- One medium triangle, congruent to the triangle with vertices (0,0), (2, 0) and (1, 1).
- Two small triangles, congruent to the triangle with vertices (0,0), (1, 0), and (0, 1).

• One square, congruent to the square with vertices (0,0), (1,0), (0, 1), and (1, 1).

• One parallelogram, congruent to the rhombus with vertices (0,0), (1, 0), (2, 1), and (1, 1).

The input to the first version of your program will be a simple closed polygon in the plane. It will be specified by an integer *n* on a line by itself, followed by *n* points, each one on a line by itself. The points will be in the form `[x, y]` where `x` and `y` are *erationals*[1]. An *erational* is an ordered triple `(A, B, C)` of integers, which stands for the real number

$$\frac{A + B\sqrt{2}}{C}$$

There is also a "canonical form" for such numbers:

• C is positive.

• A, B, and C have a greatest common divisor of 1 (with the provision that we say that *any* number divides evenly into zero).

This is useful for checking for equality exactly.

As an example of the input spec, the original square formation in which the tangrams fit, shown in Figure 1, would be described by the following input data.

```
4
[(0,0,1), (0,0,1)]
[(0,2,1), (0,0,1)]
[(0,2,1), (0,2,1)]
[(0,0,1), (0,2,1)]
```

Figure 5.

Your output should consist of the positions of the pieces placed in such a way that they fill the polygon, or the character "`X`" on a line by itself if the polygon cannot be filled with the tangram pieces. (To "fill" the polygon means that (a) all the tangram pieces are used, (b) no two overlap, except along edges, (c) the union of the tangram pieces is the polygon together with its interior.) Assuming that the polygon can be filled, the positions of the pieces should be given in the following order:

first the two large triangles

then the medium triangle

then the rhombus

then the square

then the two small triangles

Each piece is specified by giving the number of vertices of the piece followed by the coordinates of the vertices of the piece in an order such that if one walked around the boundary in this

1. I just made up this word. The right thing to say is that this points lie in the field consisting of the rationals with $\sqrt{2}$ adjoined, but "erational" is a lot shorter.

order, the interior of the piece would be on one's left. Thus the unit square might be described by the points (0,0), (1, 0), (1, 1), (0, 1), or ((1, 0), (1, 1), (0, 1), (0, 0), but *not* by (0, 0), (0, 1), (1, 1), (1, 0) or (0, 0), (1, 1), (0, 1) (1, 0). The vertices should be given as erationals, however, so that the square would actually be listed as

```
4
[(0,0,1),(0,0,1)]
[(1,0,1),(0,0,1)]
[(1,0,1),(1,0,1)]
[(0,0,1),(1,0,1)]
```

Each coordinate-pair should appear on a line by itself, and whitespace anyplace you like is OK. Any with a "#" sign as its first character is allowed in the output, and will be ignored. Thus your output might look like this:

```
#big triangle 1
3
[erational, erational]
[erational, erational]
[erational, erational]

#big triangle 2
3
[ ...
 ... ]

#medium triangle 1
 ...

 ...
#little triangle 2
3
[erational, erational]
[erational, erational]
[erational, erational]
```

The second version of your program should essentially be a wrapper around the first, it should read its input as before, make a note of the current time, and then solve the input problem 100 times, and note the time at the finish. The difference between these two times, in rounded to the nearest second, is the output of the program, on a line by itself. Thus your program might produce, as output, the single line

```
31
```

to indicate that it had taken about 31 seconds to do the algorithm 100 times on the given input.

Simplifications: Since the input consists of a simple closed curve in the plane, the pieces must all be touching each other. Furthermore, you are only to look for solutions to the puzzle in which every piece shares a non-zero-length part of at least one of its edges with some other piece.

(This is actually a consequence of the simple-closed-curve input assumption, but I thought it would be good to make it explicit). Finally, the first edge of the input polygon will be horizontal or vertical or will have slope +1 or -1.

Helpful Stuff: There's a good chance that you'll need to spend some time messing with polygons. The book on computational Geometry, by Preparata and Shamos, is a fine reference. You may also find that the material on polygon clipping in "Computer Graphics: Principles and Practice," by Foley et al., is useful.

You may also want to do some amount of visual debugging. If you're not familiar with any graphics systems, there's a truly simple one available called "xgraph" (it's not the "xgraph" command, which you can use to make plots of various things---it's just a collection of routines to keep the horrors of X out of your way and let you do some graphics). I've left the xgraph files in the comps_handout directory as well, along with a test program and a Makefile. I make no promises about xgraph functioning perfectly--there are almost certainly errors in it. On the other hand, for simple test programs it seems to work very well.

Organizational Stuff: This FrameMaker document is in /u/jfh/comps_handout, as is a PostScript file that will generate a set of tangrams for you (it's called tangram.ps). I'll be around the department most of the week, and will be happy to answer questions if you need something in the handout clarified. If anything important comes up, I'll post news to the comps newsgroup. If something really critical comes up and I'm not around, don't hesitate to try to call me at home -- 245 - 6074 -- at any hour.

Things to Keep in Mind

Your work on this problem should be your own and no one else's. You should hand in a program than can be run on the department sparcstations, and if anything fancy is required to make it run, you should tell me about it. Regardless, your program should have both internal and external documentation, and the necessary Makefiles or whatever else might be needed to convert source files into executables. Documentation should be abundantly clear. You may, however, assume that the reader has read the assignment, so your external doc's "what it does" section can simply say something like "prog1 implements the spec for the first program; prog2 implements the spec for the second." On the other hand, the "algorithms and data structures" section of the external documentation will need to be pretty extensive. Be sure to note the limitation of your algorithms or data structures, if there are any. Don't put off writing the documentation until the last minute!

Your code should be readable, and should adhere to some standard of formatting and variable naming. If your chosen convention isn't self-evident, document it for me.

Handing In

When you are finished with your program, hand it in electronically by creating a subdirectory of /u/jfh/comps_handin whose name is your 4-digit number. Put all your files in there, in world-readable form. That directory will be made readable/writeable/searchable by me alone as of Friday at 4 PM.

Don't even consider waiting until 5 minutes of 4 and copying someone else's work into your directory. I'll be reading all the programs carefully enough that such stunts will be completely obvious.