Upload Assignment: Group Project 2

Cancel Save Draft Submit

ASSIGNMENT INFORMATION

Points Possible
25

Problem

PostScript is a programming language designed primarily for describing the layout of printed pages. A PostScript program (stored as text files with the extension .ps) is interpreted by a viewer such as GsView or a printer. Interpreting a program results in a set of strokes drawn on the page (either on your screen, or on real paper in the printer). PostScript programs can also be translated into other popular page layout formats such as PDF and Word. For a brief history of PostScript, see this wikipedia article. If you're interested in learning the difference between PostScript and PDF, Adobe published an article on the subject.

In this project, you will implement a C++ library that we will call **CPS,** short for "**C++** to **P**ost**S**cript." CPS will allow its user to specify drawings at a high level of abstraction, and output the drawings as PostScript. CPS consists of:

- **A shape language** that allows basic shapes such as squares, circles, and polygons to be defined, rotated and scaled versions, and aggregate shapes, for example a vertical "stack".
- **A shapes-to-PostScript translator** that takes as input a drawing specified using CPS's shape language and produces a PostScript file from it.

Purpose

The purpose of this project is to help you develop your understanding of viewing a datatype (or collection of datatypes) as a kind of language. In designing and implementing the transformations on the PostScript commands, you'll get practice using the design patterns that we've been learning about, such as *Interpreter*.

Specification

Shape language

The CPS shape language relies on the following fundamental notions:

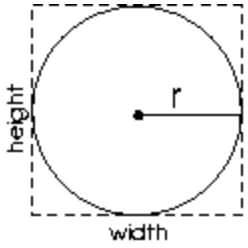
- **Bounding box.** Every shape has a *bounding box*: an imaginary rectangle that encloses the shape. *Bounding boxes are not drawn.* The *height* (*width*) of a shape is the height (width) of its bounding box. Bounding boxes are shown below using dotted lines.
- **Current point.** The *current point* represents the coordinates of the current location of the cursor. In PostScript, these coordinates are given by the command currentpoint. A shape is drawn so that its bounding box is *centered* around the current point. The center of an bounding box is shown below as a small black circle.
- **Units.** Postscript's basic unit of measurement is 1/72 of an inch. When referring to lengths below (e.g. radius, width or height), we are assuming this measurement system. For example, if a shape has a height of 36, that means 36 units, which is 36/72 = 1/2 inch.

CPS is based on a language of immutable shapes, that are either basic or compound.

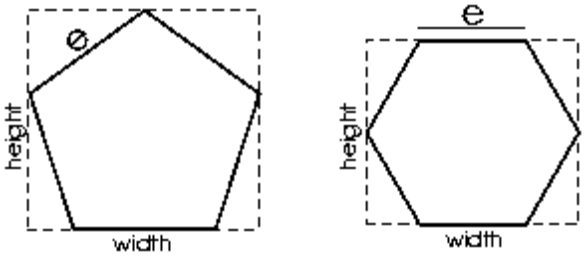
Basic shapes

Basic shapes are primitive shapes that can be created without reference to any other shapes.

1. **Circle(double radius)**. Creates a circle with the given radius. The height and width of a circle are both 2*radius.



2. **Polygon(double numSides, double sideLength)**. Creates a regular polygon with the given number of sides, each of the given length, oriented so that its lowermost side is horizontal.



3. **Rectangle(double width, double height).** Creates a rectangle of the given width and height.

4. **Spacer(double width, double height)**. Like a rectangle, but without

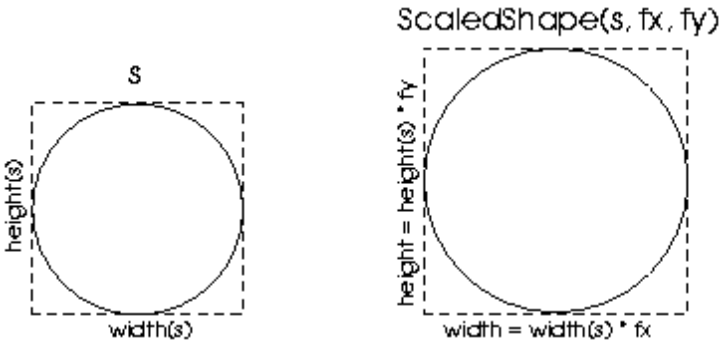
drawn borders. A spacer is not visible on the page.

5. **Square(double sideLength).** Equivalent to Polygon(4, sideLength).
6. **Triangle(double sideLength).** Equivalent to Polygon(3, sideLength).

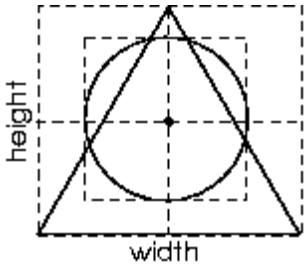
Compound shapes

Compound shapes are shapes that are constructed from one or more other shapes, which may themselves be basic or compound.

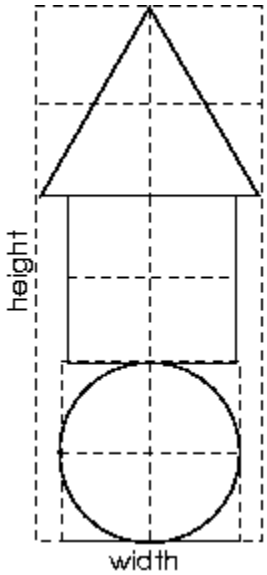
- **Rotated(Shape shape, RotationAngle rotationAngle).** Takes a shape and a rotation angle, which is either 90, 180 or 270 degrees. Creates a version of the shape that is rotated counterclockwise by the specified number of degrees around the origin of its bounding box. If the rotation angle is 90 or 270 degrees, the height (width) of the resulting shape is equal to the width (height) of the original shape.
- **Scaled(Shape shape, double fx, double fy).** Takes a shape, a horizontal scaling factor fx, and a vertical scaling factor fy. Creates a version of the shape that is scaled horizontally and vertically by the given scaling factors.



- **Layered(Shape... shapes).** Given a collection of shapes, creates a new shape consisting of all the shapes drawn with their bounding boxes centered around the current point. The height and width of a layered shape is the maximum of the heights and widths of the component shapes.



- **Vertical(Shape... shapes).** Takes an ordered collection of shapes, and creates a shape structured as follows:
  - Shape shapes[i+1]'s bounding box is located directly above the bounding box of shapes[i], and both bounding boxes are vertically aligned around their center.
  - The height of the resulting shape's bounding box is the sum of the heights of the component shapes.
  - The width of the resulting shape's bounding box is the maximum width of the widths of the component shapes.



- **Horizontal(Shape... shapes).** Takes an ordered collection of shapes, and creates a shape structured as follows:
    - Shape shapes[i+1]'s bounding box is located next to (to the right of) the bounding box of shapes[i], and both bounding boxes are horizontally aligned around their center.
    - The width of the resulting shape's bounding box is the sum of the widths of the component shapes.
    - The height of the resulting shape's bounding box is the maximum width of the heights of the component shapes.

CPS to PostScript translator

CPS lets the user translate any shape into a sequence of PostScript commands. The user can specify the name of the resulting file. The resulting file is a legal PostScript file that can be previewed on screen or printed on paper.

Tasks

1. Formalize the shape language as one or more recursive type definitions.
2. Design and implement CPS.
3. Design one or more nice shapes of your own and generate PostScript from them. You'll probably want to write some classes that build on the shape language. For example, you might implement a class Skylines that contains methods returning randomized skylines with the number/height/shape of buildings specified as input arguments to the methods. Or you might implement a class Fractals that creates some fractal figures with a recursion limit specified by the user. These are just examples; it's entirely up to you. A prize will be awarded for the best shape.
4. Design and implement a test strategy for this project. In addition to the test cases themselves, you should give a brief commentary explaining your strategy.
5. Critique the specification of the shape language. Is the language expressive enough to create interesting drawings? Are the shape abstractions well-designed, or not? Do they make certain drawings easy/difficult to create? If you were to design the shape language from scratch, what would you do differently?

Deliverables and Grading

Your deliverables for the project are:

1. The source code for CPS, along with test cases for all your classes.

2. The code and PostScript files for the shapes you invented.
3. A text document named readme.txt that lists the files you have created and says what they are; indicates how your program is used (eg, which file contains the main method and how it is invoked [on the command line? or run the executable?]; gives answers to the questions; and explains any interesting aspects of your design.

All files should be in plain text or PDF with appropriate filename extensions.

Hints

1. The standard way to structure an implementation of a language is to have a collection of classes for representing the syntactic objects of the language (in this case, the shapes), and some functions that perform operations with these objects. (This is basically the "Interpreter" design pattern.) For CPS, these operations might compute the width and height of a shape, and convert a shape into a PostScript program.
2. The "..." syntax in the compound shape definitions above is meant to represent "any number of." You will need to implement this as some sort of C++ container. (I'd recommend just having a constructor that takes an initializer_list.)
3. When implementing a function that generates a PostScript program, it may be useful to (1) assume, on entry to each visiting method, that the point around which the shape is to be centered has already been correctly set; and (2) for basic shapes, to draw the shape by starting a new path, drawing, closing the path, and calling stroke; and (3) for some operation methods, to use gsave on entry and grestore before exit. These are only suggestions, however, and you might find a different (and maybe better) approach.
4. The width and height of the bounding box for Polygon(n, e) is given by the following formulas:

    - Case 1: n is odd.
      $\text{height} = e(1+\cos(\pi/n))/(2\sin(\pi/n))$
      $\text{width} = (e\ \sin(\pi(n-1)/2n))/(\sin(\pi/n))$

    - Case 2: n is divisible by 4.
      $\text{height} = e(\cos(\pi/n))/(\sin(\pi/n))$
      $\text{width} = (e\ \cos(\pi/n))/(\sin(\pi/n))$

    - Case 3: n is divisible by 2, but not by 4.
      $\text{height} = e(\cos(\pi/n))/(\sin(\pi/n))$
      $\text{width} = e/(\sin(\pi/n))$

5. In PostScript, use showpage to finally draw the current page and create a new page.

Why CPS?

PostScript includes commands for drawing lines and arcs, rotation, scaling, and abstraction via function definition. Why not just create a library of PostScript definitions that a user can import and call directly in PostScript? The answer to this question has two parts. First, a library like CPS would come in very handy as a part of a word processing or drawing editor written in C++. The developers of such an editor could use CPS to implement an "export as PostScript" functionality in the editor. Second, CPS design separates the shape language from its translation to PostScript, and this separation would make it easily extendable so that it could produce drawings in other target formats, different from PostScript.

References

- A First Guide to PostScript
- PostScript reference manual
- Adobe's PostScript home page
- Wikipedia entry: PostScript
- GhostView, a PostScript viewer
- PostScript vs. PDF, an article by Adobe

ASSIGNMENT SUBMISSION

Text Submission Write Submission

Attach File Browse My Computer

ADD COMMENTS

Comments

Character count: 0

*When finished, make sure to click **Submit**.*
*Optionally, click **Save as Draft** to save changes and continue working later, or click **Cancel** to quit without saving changes.*

Cancel Save Draft Submit