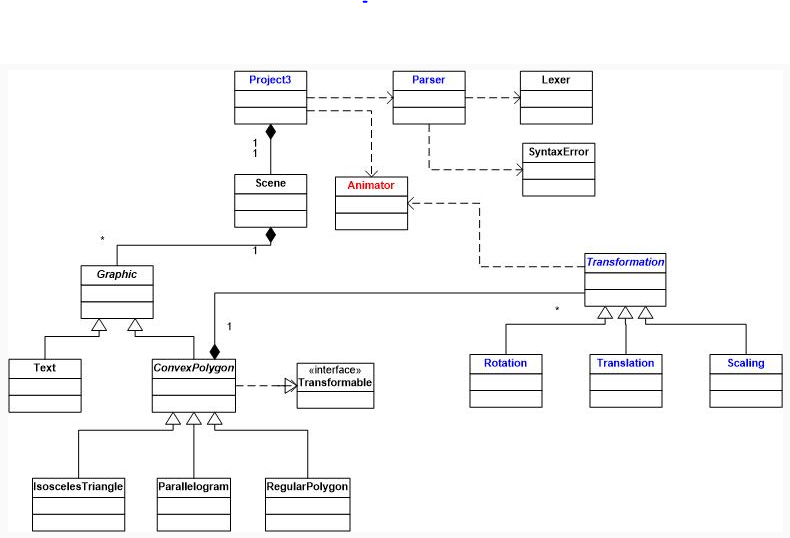
Instructions

Specification

The third project builds on the second project by adding the ability to specify animations of the transformations. The UML diagram for the new version of the project is shown below:



Project 3 UML

All the classes shown in black are unchanged from project 2. There is only one new class shown in red, Animator, which should be a singleton object. It should keep track of whether an animation is currently in process and if so, what step of the animation is currently being drawn. It should have functions to start, pause, and reset the animation. These functions need to be called by the keyboard callback function. It should also have a function to advance to the next step of the animation, which should be called by the timer callback function. Finally it needs functions to return whether an animation is in progress and one to return the current step of the animation. Those functions will need to be called by the Transformation class.

The remaining classes shown in blue will require the following modifications:

Two new callback functions must be added to the main project3.cpp source file. The first function is one that is called when a timer is fired. If the animation is running it advances to the next step by calling the appropriate function in the Animator class described above and then redraws the scene. The second is one that is called when a character is typed on the keyboard. It should start the animation when the letter s is typed, pause it when p is typed and reset the animation back to the initial state when an r is typed. Callbacks for both of these functions must be registered in main.

The Parser class must be modified to add the option of including animation steps on each transformation. The revised grammar is shown below, with the change highlighted in red:

scene -> SCENE IDENTIFIER number\_list graphics END '.'

graphics -> graphic graphics | graphic

graphic -> text | transformable\_graphic transformations END

transformable\_graphic -> isosceles | parallelogram | regular\_polygon

isosceles -> ISOSCELES COLOR number\_list ANGLE NUMBER ';'

parallelogram -> PARALLELOGRAM COLOR number\_list ANGLE NUMBER ';'

regular\_polygon -> REGULAR\_POLYGON COLOR number\_list SIDES NUMBER RADIUS NUMBER ';'

text -> TEXT COLOR number\_list AT number\_list STRING ';'

transformations -> transformation transformations | transformation

transformation -> action steps

action -> rotation | scaling | translation

steps -> STEP NUMBER TO NUMBER ';' | ';'

rotation -> ROTATE ANGLE NUMBER ';'

scaling -> SCALE number\_list ';'

translation -> TRANSLATE number\_list ';'

number\_list -> '(' numbers ')'

numbers -> NUMBER | NUMBER ',' numbers

The modified parser and corresponding token file are provided in the attached .zip file.

The Transformation interface from project 2 must now become an abstract class, which now will contain an overloaded function transform, which is not a virtual function. It should determine whether the animation is currently running and if necessary, invoke the virtual transform function supplying it the current step of the animation.

The Rotation class must be modified so the the transform function computes the total rotation angle by multiplying the angle of rotation by the step of the animation, before performing the rotation.

The Translation class must must be modified so the the transform function computes the total translation distances by multiplying the translation distances by the step of the animation, before performing the translation.

The Scaling class must must be modified so the the transform function computes the overall scale factors by raising the scale factor to the power defined by the current step of the animation.

Sample Input and Output

Below is a sample of a scene definition file that would provide input to the program:

Scene Polygons (500, 500)

Parallelogram Color(0.0, 1.0, 0.0) Angle 80;

Scale (20.0, 20.0);

Translate (-0.2, 0.2) Step 1 to 25;

Scale (1.3, 1.3) Step 25 to 28;

Rotate Angle 30 Step 31 to 42;

End

Isosceles Color (0.0, 0.0, 1.0) Angle 90;

Scale (1.5, 1.5) Step 61 to 71;

Translate (0.1, 0.1) Step 72 to 80;

Rotate Angle 45 Step 81 to 84;

End

RegularPolygon Color(1.0, 0.0, 0.0) Sides 4;

Scale (50.0, 50.0);

Rotate Angle 36.0 Step 91 to 100;

End

Text Color(0.0, 0.0, 0.0) at (5.0, 150.) "Hello World";

End.