AJYN Programming Language:

PLT Final Report
Professor Stephen Edwards

Group JAYN:
Jared Kennedy
Ananya Das
Yaniv Schiller
Neel Goyal

May 13, 2003

Table of Contents:

Introduction:

AJYN Introduction and Overview

For the past few years, certain companies have dominated the marketing in graphic animations. For years, Macromedia has been developing their Flash product, making them the market leader in easily distributed movie animations.
These files are greatly complex and allow for all sorts of unique, easy to use animations. However, due to the complexity of the design, we find that Flash files are hard to develop. The Ajyn programming language will provide an easy to use Object-Oriented interface to allow for quick, comprehensive development of complex graphic designs and animations.

Object-Oriented Design:
It is important for our language to be object-oriented in application for ease of handling various graphics functions. By declaring each graphics primitive as a separate object, be it a square or an ellipse, the user can keep track of a single object throughout the code and transform or manipulate various graphics objects through the course of the program. Furthermore, by using object-oriented design, it is possible to group many lesser objects together to create a parent object composed of many primitives. For instance, if a user wants to create a snowman object and move it around the screen, this can be done easily by creating an object composed of three circle primitives and grouping them together based on either their relative or absolute positions into one object. If the design was not object-oriented, to move the snowman the user would need to keep track of the three independent circles through using nested loops or create a polygon resembling the shape of the snowman. Therefore, the main purpose of using object-oriented programming is to make and manipulate grouped graphics objects with ease.

Positioning
Objects are positioned based on an x-y scale which places the origin at the upper left corner of the window. Moving rightwards from this point increases the values on the x-direction and moving downwards from this point increases the values on the y-direction.

AJYN Objects

- ellipse
- rectangle
- point
- line
- polygon

Animation techniques

- translation
- rotation
- scaling
-

Additional AJYN Features

- Grouping – various objects may be grouped together to form a single unit.

- Hiding Objects – an object can be set as invisible for a specified number of frames.

Animation:
With the continuing growth of the internet and computer graphics, the popularity of animation is mounting. Computer animation can be found all over the world wide web, in computer games, and in many of the everyday programs we use. Companies such as Macromedia have created graphical programs which allow users to create complex animations. AJYN presents another approach to computer animation. The AJYN programming language allows users to program animations.
One of the most valuable properties of AJYN is its easy to use animation techniques. AJYN's object-oriented design can make complex animations relatively straightforward. Animation techniques include translations, rotations, and size adjustments. We may use one of these techniques alone. or we can combine techniques to perform more complex animations.

Procedures
In order to create an animated AJYN program, we must first decide which of the three animation techniques are necessary to employ. For example, suppose we wished to create a Ferris wheel. We would need to utilize the rotation technique. We begin by creating an ellipse to create the actual Ferris wheel. We then use the rotation transformation on our ellipse object. When using the rotation function, we are required to specify three elements of data: 1) the number of degrees of rotation, 2) the start frame at which the rotation should begin, 3) the stop frame at which the rotation should end.

Ouputting to .swf Files
In order to compile the actual animation file, AJYN uses an open source project knows as Ming. Ming, which is located on Sourceforge, replaced Macromedia in creating an open source API for swf files. Ming, written in C, comes with extensions for Java, PHP, and C++. The backed application of AJYN accepts a file containing the intermediate representation and then uses Ming to compile the actual animation file.

Language Tutorial:

Writing an program in AJYN

AJYN is a very straightforward language with few commands to remember to create complete and colorful animations. Before getting into the syntax of the language, lets go over some basic things to remember while programming with AJYN. First, one begins a program by declaring a program name, screen size and number of animation frames. Then one declares the shape objects that are going to be used in the scene. Colors are specified by declaring a new color and all shapes declared afterwards are displayed using that color. Declaring groupings comes after declaring shapes. Once the elements of the scene are setup, then one can transform them over a series of frames to create an animation and hide objects for certain frames. Now that we have quickly covered the flow of a program, lets look at the syntax of the language and see how we actually design an animation in AJYN.

Starting a program:

We start a program using a declaration that gives the program a name, a screen size and a total number of frames. Suppose we want to design a clock. We could name the program Clock, give it a screen size of 500 pixels by 500 pixels and have a total frame count of 2160 frames. The declaration would look like:

```
Program Clock = 500, 500, 2160;
```

Here we can also see an important element of all declarations in AJYN. They all must end in a semicolon (;).

Declaring shape objects with colors:

Now that we have instantiated a program, lets continue by declaring various objects with different colors. There are five primitive shape objects that can be declared. Furthermore, one can declare these to be of any color on the RGB scale. A color declaration looks like:

```
ActiveColor = 0, 0, 255;
```

This has just declared the color to be blue. Any object that is declared below this color declaration will be blue, until an new color is declared. RGB values range from 0 to 255, so one cannot declare a color with a negative value or greater than 255. Lets now make a blue point with the point declaration and name the point Point1. Lets place this point in the center of the scene.

```
point Point1 = 250, 250;
```

Points just take an x coordinate and y coordinate as parameters. The coordinate system's origin is in the scene's upper left corner. At that point, the x, y coordinates are (0, 0).

Now, lets make our clock that we were designing. First we must declare the ellipse that will serve as the clock itself. We declare this first so that it appears behind the hands. Lets make the clock red.

```
ActiveColor = 255, 0, 0;
ellipse ClockBody = 250, 250, 200, 200;
```

The ellipse is centered at (250, 250) and has an radius in the x direction of 200 pixels and a radius in the y direction of 200 pixels. Here is how one would declare a circle in AJYN. Now we make the hands. Before we do so, we must realize that we wish to rotate the hands around the center of the clock. The default method of rotation in AJYN is done by rotation about the center of a shape object. Therefore, we declare two hour hands and two minute hands going in the opposite directions. By coloring one of each white, it is rendered invisible. These will be grouped together later. Here is how we declare the clock's hands:

```
ActiveColor = 0, 0, 0;
line L1 = 250, 250, 250, 150;

ActiveColor = 255, 255, 255;
line L2 = 250, 250, 250, 350;
```

In the declaration of L1, first we declare the first endpoint of the line, which is at (250, 250). The next two parameters are the other endpoint of the line, which in the case of L1 is at (250, 150). L1 is black and we declare L2 to be white, so it is rendered invisible. It is also begun from the same place but its other endpoint is 100 pixels in the opposite direction as L1. Those were the hour hands, now lets do the minute hands:

```
ActiveColor = 0, 0, 0;
line L3 = 250, 250, 250, 50;

ActiveColor = 255, 255, 255;
line L4 = 250, 250, 250, 450;
```

These hands extend to the rim of the outer ellipse. Next we'll group the visible and invisible lines together to get a clock hand object centered at the center of the clock. This clock example does not demonstrate how to declare shapes objects of rectangles and polygons. An example of a rectangle is:

```
rect R1 = 250, 250, 20, 30;
```

This is a rectangle of the last chosen color (so if this code left off from declaring line L4, the color would be white so the rectangle would be invisible). It's upper left corner is at (250, 250) and the 20 corresponds to a width of 20 pixels and the 30 corresponds to a height of 30 pixels. An example polygon is:

```
ActiveColor = 255, 0, 0;
poly Poly1 = 0, 0, 500, 0, 500, 500;
```

Polygons take a set of points, so the six parameters in the declaration above correspond to three points. A line is drawn connecting all the points, including a line that connects the last declared point with the first. This code above draws a red triangle that starts in the upper left corner of the screen, then goes to the upper right corner, then the lower right corner and then connects the first and last points (so, it connects the upper left corner with the lower right corner).

Grouping:

Groups are declared by giving the group a name and listing the shape objects that are being grouped together. The last two parameters are the x and y coordinates where the grouped objects are centered in the scene.

```
group hours = L1, L2, 250, 250;

group minutes = L3, L4, 250, 250;
```

Animating and transforming objects and groups:

There are three ways to transform and animate graphics objects in AJYN, namely rotation, translation and scaling. For the clock example, we animate it by rotating the grouped hands. Remember, the hands were grouped so that they could be rotated about the center of the clock rather than the center of the clock's hands. Transformations follow the same framework. After declaring the type of transformation we wish to use, we tell the program which object or group for which the transformation should be applied. Next we apply the given transformation (which is different for all three) and then we give the start frame and stop frame for which we want the transformation to occur. So lets rotate the hands of the clock in the correct fashion.

```
rotate(minutes, 4320, 0, 2160);

rotate(hours, 360, 30, 2160);
```

Here we take the grouped object minutes and rotate it 4320° over all the frames of the animation. If one recalls from the declaration of the program at the beginning of this tutorial, we set the total number of frames to be 2160. In the rotation of the grouped object hours, we rotate for the same number of frames. For hours, we rotate it 360° for these frames. We got 4320 by taking 360 * 12. One could also program this expression directly into the declaration of the rotation. For example:

```
rotate(minutes, (360 * 12), 0, 2160);
```

would also work. One must be sure to enclose this mathematical expression in parentheses, however, or the compiler will generate an error. Although there is no place in this clock example to translate or scale an object, lets see how those transformations

work, using our polygon and rectangle from the shape declaration section. First lets translate the rectangle to the upper left corner of the screen:

```
translate(R1, 10, 15, 30, 60);
```

The translation gives the rectangle new center x and y coordinates at 10 and 15 respectively. Also, this transformation takes place from frame 30 to frame 60 of the animation. Now lets shrink the triangle:

```
scale(0.05, 60, 2160);
```

From frame 60 to the end of the animation, we shrink the triangle polygon to $1/20^{th}$ its original size. There are two things to note in the scaling transformation. First, scaling to a negative size is impossible and will yield an error in compilation. Also, floating point numbers between -1 and 1 must be declared with the 0 preceding the decimal point. Calling scale with the first parameter of .5 instead of 0.5 will also yield an error in compilation.

Hiding objects:

The last thing we can do is hide objects for a set number of frames. This is declared similar to the declarations for other transformations. If we want to hide the red triangle polygon for the entire animation (so to not ruin our pretty picture of the clock), we can do so by using the following function:

```
setInvisible(Poly1, 0, 2160);
```

Hiding can be done over any set of frames. The 0 corresponds to the frame for which we want to begin to hide the polygon and 2160 corresponds to the frame for which we wish to render the polygon visible once again.

Commenting and code documentation:

It is always good style to add comments to code. This holds true for any graphics animation where several elements may be placed in the scene and require keeping track of. AJYN supports only single line comments that are denoted by '//'. Any text that occurs in the file on a given line after // is discarded by the compiler. Comments can appear anywhere in the source file except for the very last line.

Compiling a source file in AJYN

starting from the .g file with antlr in the ~/antlr/ directory and the java files and source.ajn in ~/files/

```
export CLASSPATH=~/antlr/;
java antlr.Tool AJYN99.g
export CLASSPATH=$CLASSPATH:~/files/;
javac *.java
```

java AJYN < source.ajn

that will produce an x.ir

then run ./a.out and this a flash media (.swf) file

Language Reference Manual:

1. Introduction

For the past few years, certain companies have dominated the marketing in graphic animations. For years, Macromedia has been developing their Flash product, making them the market leader in easily distributed movie animations.
These files are greatly complex and allow for all sorts of unique, easy to use animations. However, due to the complexity of the design, we find that Flash files are hard to develop. The Ajyn programming language will provide an easy to use Object-Oriented interface to allow for quick, comprehensive development of complex graphic designs and animations.

2. Lexical conventions

Token types include identifiers, keywords, constants, function calls, types, and separators.
All statements (with the exception of a group) end in a semi-colon (;).

2.1 Comments

Comments in AJYN are restricted to one line. The characters // introduce a comment.

2.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore counts as alphabetic. All characters are significant. It is good convention to start with a lowercase letter.

2.3 Keywords

The following identifiers are reserved for keywords and cannot be used otherwise:

ellipse	rect
line	point
group	rotate
translate	scale
MAX	Program
ActiveColor	

2.4 Function calls

There are no user defined functions. AJYN allows for the use of the following predefined functions: rotate, translate and scale.

2.5 Constants

There two constant types in AJYN. They are as follows:

2.5.1 Integer Constants

An integer constant is a sequence of digits, always expressed as a decimal. MAX is a reserved integer set at 65,536.

2.5.2 Floating Point Constants

A floating point number consists of an integer part and a decimal part. The integer and decimal parts both consist of a sequence of digits, the decimal part is proceeded by a '.'. For instance, 12.0 is a valid floating point number while 12. is not. One must also be careful to declare a value such as 0.3 with the 0 before the decimal point. Declaring a value as .3 will give an error.

2.5.3 Negative Numbers

There is not direct support of negative numbers in AJYN. This reduces the possibility for errors in programming. However, the user can specify a negative number by using a mathematical expression that evaluates to a negative value. See section 5.1.

3. Objects and Types

There are five fundamental data objects within the AJYN language. These are points, lines, rectangles, ellipses and polygons in addition to floating point and integer numbers.

Integers (*int*) are represented in 16-bit 2's complement notation.

Floating point numbers (*float*) are all single precision and use the IEEE 754 floating point notation.

3.1 Shape Objects

The remaining primitives in AJYN are data types containing identifier names and a set of property variables.

3.1.1 Points

Points are declared by:

point *identifier-name = param, param*;

Parameters are integers. The first parameter is the value of the x position. The second parameter is the value of the point's y position.

3.1.2 Lines

Lines are declared by:

line *identifier-name = param, param, param, param;*

Parameters are integers or references to integers. The first two parameters are the x and y coordinates of one endpoint of the line. The third and fourth parameters are the x and y coordinates of the line's other endpoint.

3.1.3 Rectangles

Rectangles are declared by:

rect *identifier-name = param, param, param, param;*

Parameters are integers and the first two can also be references to integers. The first two parameters are the x and y coordinates for the upper left corner of the rectangle. The third parameter is the width of the rectangle and the fourth parameter is the height.

3.1.4 Ellipses

Ellipses are declared by:

ellipse *identifier-name = param, param, param, param;*

Parameters are integers and the first two parameters can also be references to integers. The first two parameters are the x and y coordinates for the center of the ellipse. The third parameter is the radius in the x direction and the fourth is the ellipse's radius in the y direction.

3.1.5 Polygons

Polygons are declared by:

poly *identifier-name = param, param, … , param, param;*

Parameters are integers or references to integers. A polygon requires at least three points and therefore, at least six parameters, as every pair of parameters represent a point in the polygon. A polygon declaration must contain an even number of parameters representing point pairs. The last pair and first pair of points will be joined to form the closing line of the polygon.

4. Expressions

An expression can be an instantiation or a function call.

4.1 Instantiation

type identifier '=' *parameter list*

The parameter list is dependent on the type. (See above).

4.2 Function call

function name '(' *identifier* ',' *parameter list* ')'

Possible function names are rotate, translate, and scale. The parameter list is dependent on the function name.

5. Parameters and Passing by Value

All parameter passing in AJYN is pass-by-value. Objects are passed directly rather than a reference to an object being passed, and modification of the object through transformations modifies the properties of the object itself. All parameters can be of either type *int* or *float*.

5.1 Mathematical Expressions

When passing parameters, use of infix mathematical expressions are acceptable. For example, (3 + 2) is an acceptable way to pass 5 as a parameter. Note that using negative values in a mathematical expression is not supported in AJYN, but if the user wishes to declare a negative number, it can be done using a mathematical expression such as (0 - 3) which evaluates to -3.

6. Groups

A group is a collection of objects that are treated as one. The definition spans several lines. Before instantiating a group, it is necessary to declare all of the shapes that will grouped together. These shapes can be any of the graphics primitives defined in section 3 of this document. The identifiers of these objects are used to group them together in the declaration of a group. Knowing this, groups are declared by:

group identifier-name = identifier-of-object1, identifier-of-object2, …, identifier-of-objectn, centerx, centery;

These last two parameters represents the center x and y coordinates of the group and can be used to place the group in a desired location other than the one specified in the declaration of the group's shapes.

7. Color

Colors can be set to RGB values. There is also an *ActiveColor* object that sets the color of every object instantiated afterwards to that color. The default *ActiveColor* is black.

```
//A Blue Diagonal Line
ActiveColor=0,0, 255;
Line L1=0,0,50,50;
//A Red Diagonal Line
ActiveColor=255,0,0;
Line L2=0,50,50,0;
```

7.1 Invisibility with Colors

Any object declared after setting the color to white (RGB value of 255, 255, 255), the object will be rendered invisible in the scene. This function is useful in rotating objects at points other than the center of the object. By making an invisible object, and grouping it with a visible object, depending on the placement of the two objects relative one another, the rotation can occur at some point other than the center of the visible object.

8. Animation

The main idea in AJYN is to allow for objects to be easily animated. This is setup to control the movement of specific graphics objects through frames. There are three function calls with which an object can be animated. It can be translated, rotated or scaled. The is also another function call that renders an object invisible.

8.1 Frames

Animation is specified through a series of frames. Each frame lasts for a specific amount of time before proceeding to the next. The progress of frame to frame gives the appearance of an object moving through a space over a period of time. The start frame is always 0.

8.1.1 MAX frame

The reserved word MAX stores the maximum number of frames for the animation, and for that reason, it contains the highest *stopframe* (see 8.2.2) that can be specified.

8.2 Translation

Translations are specified by:

translate(*identifier, newx, newy, startframe, endframe*);

This takes an object and moves it from its current location to *newx* and *newy* incrementally from the *startframe* to the *endframe*.

8.2.1 *newx* and *newy*

These two values contain the new center x and y coordinates of the object being translated. Instead of specifying distances that the object is to be translated, *newx* and *newy* contain the destination of the object being translated.

8.2.2 *startframe* and *stopframe*

These frames contain the start and stop location in the animation for which the given transformation occurs. The values for *startframe* and *stopframe* can also be specified with mathematical infix expressions. These values also must be specified in all other transformations.

8.3 Rotation

Rotations are specified by:

rotate(*identifier, rotate-degrees, startframe, stopframe*);

This function takes an object and rotates it according to *rotate-degrees* a specified amount incrementally from *startframe* to *stopframe* (see 8.2.2).

8.3.1 *rotate-degrees*

Designating this with a positive value creates a clockwise rotation and a negative value corresponds to a counterclockwise transformation.

8.4 Scale

Scaling an object is specified by:

scale(*identifier, scale-size, startframe, stopframe*);

This function will scale an object to its current size multiplied by the *scale-size* incrementally from *startframe* to *stopframe* (see 8.2.2).

8.4.1 *scale-size*

This value must always be greater than zero. A negative scaling size will yield an error because of its impossibility in 2D geometric transformations. Furthermore, if the scaling size is too large, then it will make an object that is too large to fit within the viewing area. The *scale-size* can also be specified by means of a mathematical infix expression.

8.5 Setting Invisibility

This function is specified by:

setInvisible(*identifier, startframe, stopframe*);

This function renders an object invisible in the scene from *startframe* to *stopframe* (see 8.2.2).

9. Defining a Program

The first line written in AJYN initializes and names the program being defined. Its format is as follows:

Program *program-name = screen-width, screen-height, total-frames*;

The program name can be any identifier (see 2.2). *screen-width* and *screen-height* determine the size of the screen in pixels.

9.1 *total-frames*

The total number of frames for which animations can occur is defined in the instantiation of a program. This value must be less than MAX (see 2.5.1).

10. Sample Program

```
//*****************
//Test Code for a Clock
//*****************

//***
//PROGRAM INSTANTIATION
//***
Program Clock = 500, 500, 2160;

//***
//OBJECT DECLARATIONS
//***
ActiveColor = 0, 0, 255;
line L1 = 250, 250, 250, 150;

ActiveColor = 255, 255, 255;
line L2 = 250, 250, 250 350;

ActiveColor = 0, 255, 0;
line L3 = 250, 250, 250, 50;

ActiveColor =255, 255, 255;
line L4 = 250, 250, 250, 450;

ActiveColor = 255, 0, 0;
ellipse E1 = 250, 250, 200, 200;

//***
//GROUP DECLARATIONS
//***
group hours = L1, L2, 250, 250;
group minutes = L3, L4, 250, 250;

//***
//ANIMATION AND TRANSFOMATION DECLARATIONS
```

```
//***

rotate(minutes, 4230, 0, 2160);
rotate(hours, 360, 30, 2160);

//end of program
```

Project Plan:

Development Process

We split up the work into front end, backend, documentation and testing. Front end and backend were the largest jobs, but it was easiest for one person to tackle each of those jobs. Documentation and testing required going back and forth between the front end and backend people to ensure that both modules went together and to gather an overall group understanding of what was happening. To allow for greater coverage, we shared documentation and testing between two people. We had an intermediate file for the front end to produce and the backend to use for outputting to swf. Therefore, front end and backend could work independently. Furthermore, testing could be done on the front end and backend independently as well.

Who did what?

Ananya Das - Documentation and Testing
Jared Kennedy - Documentation and Testing
Neel Goyal - Front End Development
Yaniv Schiller - Backend Development

Code Style and Convention for Developers

Working as a team requires easily understandable code in order to conserve time and energy that could be spent trying to figure out what a simple function does. Though placement of curly braces and semi-colons are not entirely important, they should be placed in a manner which allow for less confusion.

The Importance of Commenting

Commenting code is the best way to communicate with other programmers. Since the frontend was done in Java and ANTLR, and the backend was done in C++, determining what a function does can be complicated unless the code is well documented. Block commenting or single line commenting seemed fine, as long as they explained what was going on. Javadoc was not used since the team felt little need for it.

Naming Conventions

Naming variables used is often key in processing the code of a programmer. In many instances, variables are already known to the team, such as the parameters passed to objects in their declarations. However, other instances may leave a programmer wondering what the use of it is unless it has an understandable identifier.

Project Timeline

<u>Early February</u>

- Project Topic Brainstorming & Selection

<u>Mid-February</u>

- Basic ideas about possible objects and transformations
- Rough ideas about language formatting and syntax
- Basic ideas for backend procedures (ie. generating flash files)

- Whitepaper composition

Mid-March
- Analysis of syntax rules
    - -lexical conventions
    - -commenting
    - -identifiers
    - -declaring objects
    - -calling functions
    - -parameter passing and arguments
- Analysis of the object-oriented programming aspects
    - -types of objects
    - -types of functions
- Development of grammar

April
- Outline of front end
    - -completion of lexer
    - -completion of parser
    - -ideas about AST
- Outline of back end
    - -decision to use MING
    - -determining basic algorithm of back end

May
- Completion of frontend
    - -completion of AST (writing to an IR file)
- Completion of backend
    - -reading from the IR file
    - -generating the appropriate swf file

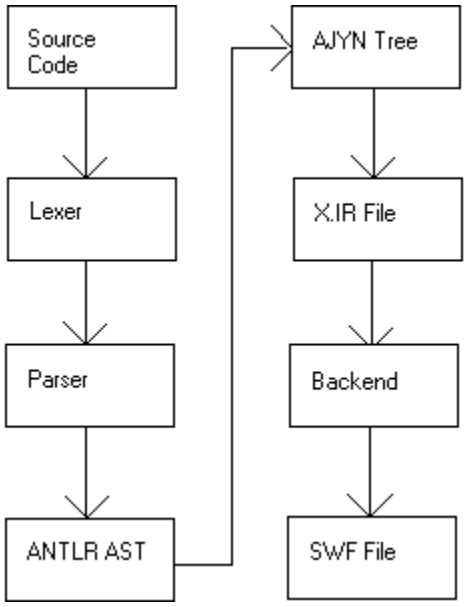
Architecture Design:

AJYN's compiler consists of many parts, several of which are common to all compilers. The source file, which has extension **.**ajn, is read and scanned by a lexer. The lexer generates tokens, and creates an abstract syntax tree (AST) from them, provided no errors in scanning occur. The AST is converted to another type of tree, which is easy to walk in order to generate a suitable intermediate representation. In this instance, after error checking is done, the frontend creates a file called x.ir, which basically consists of a series of numbers.

The Frontend

The frontend was created using ANTLR, which can be found at www.antlr.org. The primary components of any frontend are the lexer and parser, which are then used to generate an AST. The common AST which ANTLR uses becomes translated into the AJYN tree, created using java data structures.

For visual aid, a block diagram has been provided below.



Block Diagram for AJYN Compiler

It is important to note that error checking occurs at the lexer and prior to the generation of the x.ir file when the AJYN Tree is created from the ANTLR AST. Syntactic errors are discovered when the source is being scanned, such as if a user inputs a float or expression when an integer is required or when a misspelling in a function call occurs. If a syntactic error is found, the compiler stops, alerts the user, and no AST is generated.

Semantic checking occurs before writing the x.ir file in order to ensure a flawless intermediate representation will be generated. Such errors include the use of unknown identifiers or parameters that are not allowed in the language, such as a frame number higher than MAX. Furthermore, a lookup table has been implemented in the AJYN Tree

to prevent the use of the same identifier twice for different objects. If a semantic error is discovered, the compiler will not generate an intermediate representation file, and the user will be notified.

Runtime Environment

Since the goal of the AJYN language is to generate a Macromedia Flash Format file, which has the extension .swf. In order to view such files, one must have a Flash file viewer. One may find this viewer at www.macromedia.com. However, many newer web browsers, such as Internet Explorer 6 and Netscape come with the ability to play Flash files, including the ones generated by the AJYN compiler.

The Backend

The Backend Program was written in c++. It is a separate program from the front end and is called only after the front end's completion. Furthermore, because it is only dependent on the IR, it can easily be ported to work with another front end application. This program uses an swf API provided by ming. Ming, an open source project located on Sourceforge, replaced Macromedia in creating an open source API for swf files. Ming, written in C, comes with extensions for Java, PHP and C++. AJYN uses this source to compile the actual animation file.

The basic structure of the backend application is easy to understand. In the beginning the program reads in the data from the IR. The data is read into predefined classes that represent the structure of each object. There are object classes for the shapes and the groups, animation classes for the animations, and invisibility class for invisibility definitions. Certain extra information is added to some classes. For example, the center of all shapes and groups are calculated and stored with the shape.

After the data is read the backend proceeds to process the data. First it sets the width, height and video length of the animation to the numbers specified by the user. Then it adds all visible objects to the location of their respected positions. The objects are added to the animation based on their center. Then, for each frame, each object's size and position and visibility is recalculated and updated in the animation. One of the powers of flash is that if an object doesn't move then it doesn't get re-stored in the animation. We take advantage of this and only sore the repositioned, animated objects. When the getShape() function is called on a group, the program loops through every shape it references and creates a new shape based with a center based in the middle of the group.

More specifics on architecture:
When displaying every shape, the shape class calls its getShape() function where a SWFShape object (passed in by reference) is updated with the shape details. Another version of such a function was created to create the shape at a specified location. The latter function is used to create groups. Without it all the objects in a group would be lumped in the group. The SWFShape object is a class used in Ming to create a shape object.

When displaying a circle, the shape object calls an ellipse() function, where 8 curved

lines are created to for the ellipse. SWF files do not have methods of storing circles or ellipses so it has to be done in curved edge segments.

Implementing Invisibility is not a simple feat either. One cannot just remove the object from display and the reinsert it. Problems arise in terms of animation, the animation would resume its location and size where it left off. This is not desired results. To fix the problem all that is done when an object is invisible is setting its scale to .00001, making the shape too small for flash to display. When the frame comes for the shape to reappear, the current scale is recalculated and the shape is shown, it the position and size where it would have been had it not been set to invisible.

Intermediate Representation Specifications
each object has an identifier_number
each object has a previously know number of paramters
then we follow each with color numbers

Each line can be one of the following:

section_id number_params params

OBJECTS SECTION:
type_id_number identifier_number number_of_paramaters parameters colorR colorG colorB

GROUPS SECTION:
group_id_number identifier_number number_of_objects identifier_numbers x y

FUNCTION SECTION:
func_id_number obj_identifier_number number_of_paramaters parameters

INVISIBILITY SECTION
obj_identifier_number start_frame end_frame

section_id		#parameters
program	01	3
objects	02	0
groups	03	0
funcs	04	0
visibs	05	0
EOF	09	0

types_id_number		#parameters
point	11	2
line	12	4
rect	13	4

ellipse	14	4
poly	15	N

group_id_number:

group	19

func_id_number	id	#parameters
translate	21	4
rotate	22	3
scale	23	3

Test Plan:

With the completion of our front and back ends, we conducted a series of test procedures in order to ensure the robustness of our programming language. We found that possible errors fell into one of 6 categories:

1- Spelling errors – these are errors caused by the user. Spelling errors included the following types:
- spelling errors in keywords
- spelling errors in user-defined words

2- Usage errors – these are errors caused by misuse of our programming language. Usage errors include the following types:
- incorrect number of arguments
- incorrect type of arguments
- using keywords as identifiers
- misusing '=', ‘( )’, ‘;’, and ‘,’
- commenting incorrectly

3- AJYN restriction errors – these are errors due to certain restrictions of our programming language. The user may assume that the following actions are allowed, but since our language does not handle them, they will cause errors:
- attempting to declare a group within another group
- attempting to perform a transformation on a single member of a group

4- Programmer errors – these are errors that would most likely be caused unintentionally by the programmer
- attempting to use the same name for two different objects
- attempting to transform an object without calling it first
- attempting to perform an overlapping animation (ie. attempting to scale the same object by .5 and .75 during overlapping frames)
- attempting to perform a transformation with the *startframe* greater than the *stopframe*
- duplicating a point in the declaration of a line

5- Out of bound errors –these are errors caused by using numerical values that are out of the range of valid values. The following are possible out of bound errors:
- out of range [0, 255] for RGB values
- declaring a *startframe* a value that is less than zero
- declaring a *startframe* a value that is greater than MAX or greater than the total number of frames of the program
- declaring scale-size as a negative value
- declaring radius or width/height as a negative value

6- Numerical errors – these are errors due to certain numerical formatting rules of the AJYN language:
- using floats incorrectly:

- ex 1) 12.0 must be written as 12.0 and not 12.
- ex 2) .3 must be written as 0.3 and not .3

- negative numbers need 0- in front
  - ex 1) -3 must be written as 0-3 and not -3

Our testing procedure followed with these steps:

Step 1)
We first needed to determine how we would handle each type of error. We could 1) abort the compilation and generate a compilation error or 2) allow the program to compile and generate unexpected results. For example, since errors of Type 1 and Type 2 would not allow for parseable code, we handled all of these types of errors by generating compilation errors.
Step 2)
Since most of the errors of Type 3-6 would not be caught by our parser, these errors were handled differently. Decisions had to be made as to how to handle these errors. Would we format our language so that it generates compilation errors if these types of errors were encountered? Or should we allow the program to compile, and generate some unexpected results in the swf file? These decisions were based on factors of simplicity and efficiency. If too many changes were needed to be made to our language in order to detect an error and generate a compilation error, then we decided to allow the error to pass the compilation phase. If however, allowing one of these types of errors to pass the compilation phase could be possibly detrimental to the entire program, we decided to abort compilation and generate a compilation error.
Step 3)
We created several test programs. Each of the six test programs given below handle one of the six errors discussed above. We ran each of these test programs and verified that the correct actions were taken by our compiler.
Step 4)
The final and perhaps most crucial step of our testing procedure was what we call the "dummy test". In this step we asked non-group members to write programs using AJYN. By doing this we were able to see types of problems an average programmer might encounter when using our language.

Additional testing was conducted on the java data structure, AJYNTree, to ensure that proper calls to the methods produced a perfect x.ir file.

Test Programs

Type 1

```
//TYPE 1 ERRORS
Program Stuff = 400, 400, 2160;

//ActiveColor spelled incorrectly
```

```
ActiveColour = 0, 0, 255;
line line1 = 100, 110, 150, 160;

ActiveColor = 0, 255, 0;
ellipse ellipse1= 170, 200, 10, 15;

//ellipse1 spelled wrong
rotate(line1, 360, 30, 2160);
translate(elipse1, 250, 300, 0, 2160);
```

Type 2

```
//TYPE 2 ERRORS
Program Stuff = 400, 400, 2160;

//incorrect commenting
/comment

//Wrong number of arguments;
ActiveColor = 0, 0, 255;
line line1 = 100, 110, 150;

ActiveColor = 0, 255, 0;
ellipse ellipse1= 170, 200, 10, 15;

//arguments mixed up
rotate(360, 30, 2160, line1);

//using keyword 'ellipse' incorrectly
translate(ellipse, 250, 300, 0, 2160);

//misuse of ( )
ActiveColor = 0, 255, 0;
ellipse ellipse2 = ( 80, 80, 10, 10)

//misuse of =
ActiveColor = 0, 255, 0;
translate = (ellipse1, 400, 400, 0, 1500);
```

Type 3

```
//Type 3
Program Stuff =  500, 500, 2160;

ActiveColor = 0,0, 255;
line L1 = 250, 250, 250, 150;
```

```
ActiveColor = 255, 0, 0;
line L2 = 30, 50, 180, 100;

group lines = L1, L2, 250, 250;
ActiveColor = 255, 0, 0;
line L3 = 40, 60, 280, 100;

//group lines within group linesagain
group linesagain = lines, L3, 60, 100;

//transforming single member of group
translate(L1, 300, 250, 0, 2000);
```

Type 4

```
//Type 4
Program Stuff = 500, 500, 0, 2000;

ActiveColor = 0, 0, 255;
line L1 = 25, 25, 100, 70;

//L1 used again
ActiveColor = 255, 0, 0;
line L1 = 30, 30, 120, 80;

//L1 used again again
ellipse L1 = 150, 60, 30, 30;

//ellipse1 never created
scale(ellipse1, .5, 0, 500);

//Duplicating a point in a line declaration
ActiveColor = 0, 255, 0;
line L2 = 150, 60, 150, 60;

//overlap in animation
scale(L1, 0.5, 0, 500);
scale(L1, 0.8, 20, 450);

//startframe greater than stopframe
scale(L1, 0.5, 500, 10 );
```

Type 5

```
//Type 5
Program Stuff = 500, 500, 2000;

//out of bound for RGB range
```

```
ActiveColor = 0, 0, 355;
line L1 = 25, 25, 100, 70;

//out of bound for the RGB range
ActiveColor = 255, 0-10, 0;
line L1 = 30, 30, 120, 80;

ActiveColor = 0, 0, 255;
line L2 = 25, 25, 100, 70;

//negative values for height and width
ActiveColor = 0, 255, 0;
rect R1 = 150, 100, -25, -30;

ellipse ellipse1 = 150, 60, 30, 30 ;

//start frame is negative
scale(ellipse1, 0.5, 0-10, 500);

//start frame is greater than max of program
scale(ellipse1, 0.5, 10, 3000);

//start frame is greater than MAX
scale(ellipse1, 0.5, 10, 40000);

//scale size is negative
scale(ellipse1, 0-2, 0, 500);


//startframe greater than stopframe
scale(L1, 0.5, 500, 10 );
```

Type 6

```
//Type 6
Program Stuff = 500, 500, 2500;

ActiveColor = 0, 0, 255;
rect rect1 = 30, 20, 50, 60;

ActiveColor = 0, 255, 0;
ellipse ellipse1 = 150, 220, 40, 20;

ActiveColor = 255, 0, 0;
line line1 = 200, 300, 260, 300;

//must be 2.0 instead of 12.
```

```
scale(rect1, 2., 0, 200);

//must be 0.3 instead of .3
scale(ellipse1, .3, 0, 200);

//must be 0-3 instead of -3
rotate(line1, -90, 0, 200);
```

Sample Clock Code and Corresponding Intermediate Representation File

```
//*****************
//Test Code for a Clock
//*****************

//***
//PROGRAM INSTANTIATION
//***
Program Clock = 500, 500, 2160;
        //01 3 500 500 2160

//***
//OBJECT DECLARATIONS
//***
        //02 0

ActiveColor = 0, 0, 255;
line L1 = 250, 250, 250, 150;
        //12 1 4 250 250 250 150 0 0 255

ActiveColor = 255, 255, 255;
line L2 = 250, 250, 250, 350;
        //12 2 4 250 250 250 350 255 255 255

ActiveColor = 0, 255, 0;
line L3 = 250, 250, 250, 50;
        //12 3 4 250 250 250 50 0 255 0

ActiveColor = 255, 255, 255;
line L4 = 250, 250, 250, 450;
        //12 4 4 250 250 250 450 255 255 255

ActiveColor = 255, 0, 0;
ellipse E1 = 250, 250, 200, 200;
        //14 0 4 250 250 200 200 255 0 0

//***
//GROUP DECLARATIONS
```

```
//***
        //03 0

group hours = L1, L2, 250, 250;
        //19 10 2 1 2 250 250

group minutes = L3, L4, 250, 250;
        //19 11 2 3 4 250 250

//***
//ANIMATION AND TRANSFOMATION DECLARATIONS
//***
        //04 0

rotate(minutes, 4320, 0, 2160);
        //22 11 3 4320 0 2160

rotate(hours, 360, 30, 2160);
        //22 10 3 360 30 2160
```

Intermediate Representation File:

```
1 3 500 500 2160
2 0
12 0 4 250.0 250.0 250.0 150.0 0 0 255
12 1 4 250.0 250.0 250.0 350.0 255 255 255
12 2 4 250.0 250.0 250.0 50.0 0 255 0
12 3 4 250.0 250.0 250.0 450.0 255 255 255
14 4 4 250.0 250.0 200.0 200.0 255 0 0
3 0
19 5 2 0 1 250.0 250.0
19 6 2 2 3 250.0 250.0
4 0
22 6 3 4320.0 0 2160
22 5 3 360.0 30 2160
5 0
9 0
```

Lessons Learned:

Yaniv Schiller

The project was fun. I learned the importance of group agreement. It was needed to get anything done. Also, working backwards isn't so bad. Having modular code let's you increase productivity.

Jared Kennedy

I learned many new things from the PLT project that are applicable to my computer science education and my life as a whole. This was the first large project in which I was required to work with a group of people for a whole semester. At first we had to feel one another out to see if we were compatible group mates, but once the uncertainty wore off, we were able to get right to work. I found that by being upfront with people, they weren't going to be put off if it meant that everything got done (and got done right). Now that I have covered the life lesson, lets look at some of the things I learned about that apply to computer science. This project allowed for a complete understanding of what actually goes into the design of a computer language. The picture covers not only the aspects of testing a coding that go into the language, but also the nuts and bolts behind the language and compilation process. I worked primarily on documentation and testing this project, but was very attentive and curious to understand what was happening in all parts of the development process. In terms of testing, I learned that it is not easy to come up with all the possible errors that a language can generate. The scope of our language is highly limited, but that did not keep from making error checking a long and difficult (and at times very draining) process. From learning what goes into building the front end, by parsing a grammar and building an AST, to watching how to convert from an intermediate file to a backend finished product, I am much wiser in the ways of language development than I ever could have been had I not taken part of a project on this scale.

Neel Goyal

Communication between team members is key - an ideal situation would be one where everyone knows what's happening with the project at all times. I also learned that I should have started this project earlier. Procrastination requires making sacrifices.

Ananya Das

I played a role in the front end and in debugging. From the front end aspect, I learned that the structure behind any program is much more complicated than it may appear. Our language, for example, despite its simplicity, entailed a complex Abstract Synatx Tree consisting of several layers of various nodes. In helping to create this tree, I learned about ways to make the construction of these types of trees more efficient. During the debugging phase, we tried to think of all the possible errors that could occur. The most important thing I learned from this phase was that no matter how sure you are that you've covered every possibility, you more than likely have not. The best way to approach this problem is to pretend that you are an average person using a programming language that is very unfamiliar to you. This way, you encounter many errors that would

initially not occur to you. Another good way to approach this problem is to have non-group members test out the language. By doing this, you can actually see how an average person would use the language.

Appendix:

Front End files - Neel Goyal and Ananya Das

```
//AJYN.java
//Author:  Neel Goyal

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.DumpASTVisitor;
import antlr.RecognitionException;
import antlr.TokenStreamException;

//Invokes the ANTLR generated lexer, parser, and walker on the
//input stream

class AJYN {
	public static void main(String[] args) {
		try {
			AJYNLexer lexer = new AJYNLexer(new DataInputStream(System.in));
			lexer.setFilename("<stdin>");
			AJYNParser parser = new AJYNParser(lexer);
			parser.setFilename("<stdin>");
			// Parse the input file
			parser.file();

			//check if parser errors occured
			if (parser.errors)
			{
				System.out.println("Errors - compilation stopped");
				System.exit(1);
			}
			AJYNWalker w = new AJYNWalker();

			//will stop if errors occur while walking
			w.file((CommonAST) parser.getAST());
		}
		catch(TokenStreamException e) {
			System.err.println("exception: "+e);
		}
		catch(RecognitionException e) {
			System.err.println("exception: "+e);
		}
	}
}
```

```
//Author:  Neel Goyal
//Java files that ANTLR generates are not included
//AJYN99.g

{
        import java.util.*;
        import java.lang.*;
}

//Lexer class
class AJYNLexer extends Lexer;

options { k=2;
                        charVocabulary = '\3'..'\377' | '\u1000'..'\u1fff'; }

//whitespace
WS: (' ' | '\t' | '\n' {newline();} | '\r' {newline();})+
        { $setType(Token.SKIP); } ;

//integer or float
INT
options { paraphrase="a number"; }
 : ('0'..'9')+
   ('.'
    ('0' .. '9')+ { $setType(NUM); }
   )?
   ;

protected LETTER: ('a'..'z') | ('A'..'Z');

ID options { paraphrase="an identifier"; }
: LETTER (LETTER | '0'..'9' | '_')*;

COMMENT: "//" (~('\n'| '\r'))* ('\n' {newline();}| '\r' {newline();})  { $setType(Token.SKIP); };

COMMA options { paraphrase="a comma"; }
: ',';

DOT : '.';

PLUS: '+';

STAR: '*';

DIV: '/';

SUB: '-';

LP: '(';

RP: ')';

EQ: "=";

LB: '{';
```

```
RB: '}';

SEMI: ';';


//parser
class AJYNParser extends Parser;

options {		buildAST = true; }

tokens {
  NUM;
}

{
	boolean errors = false;
	//If an error occurs, the value is set to true, and compilation stops
}

//mathematical expression - sum/subtraction are lowest precedence
expr: prod ((PLUS^ | SUB^) prod)*;
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }

prod: atom ((STAR^ | DIV^) atom)*;

num : INT | "MAX"! { #num = #([INT, "65536"]); };
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }

atom: num | LP! expr RP! | NUM | SUB^ atom;
exception catch[RecognitionException ex] {System.out.println(ex);
System.out.println("Expressions must start with non-negative number");
System.out.println("Floats must be in the form #.#..."); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex);
System.out.println("Expressions must start with non-negative number");
System.out.println("Floats must be in the form #.#..."); errors = true; }
//end mathematical expressions and number definitions


//Some functions used in AJYN.  See syntax / tutorial for further details
funct: "rotate"^ LP! ID COMMA! expr COMMA! num COMMA! num RP!
    | "translate"^ LP! ID COMMA! expr COMMA! expr COMMA! num COMMA! num RP!
    | "scale"^ LP! ID COMMA! expr COMMA! num COMMA! num RP!
    | "ActiveColor"^ EQ! INT COMMA! INT COMMA! INT
    | "setInvisible"^ LP! ID COMMA! num COMMA! num RP!
    ;
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }

//The first line of an AJYN program must be of type init2.
init2: "Program"^ ID! EQ! num COMMA! num COMMA! num
   ;
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }
```

```
//declarations of objects
dec
 : "ellipse"^ ID EQ! expr COMMA! expr COMMA! expr COMMA! expr
 | "rect"^ ID EQ! expr COMMA! expr COMMA! expr COMMA! expr
 | "poly"^ ID EQ! expr COMMA! expr (COMMA! expr COMMA! expr)+
 | "point"^ ID EQ! expr COMMA! expr
 | "line"^ ID EQ! expr COMMA! expr COMMA! expr COMMA! expr
 | "group"^ ID EQ! (ID COMMA!)+ expr COMMA! expr
 ;
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }

//an AJYN file structure - one init2 line followed by a series of
//functs or decs (statement)
file
 : init2 SEMI! (statement SEMI!)+ EOF!;
exception catch[RecognitionException ex] {System.out.println(ex); errors = true; }
catch[TokenStreamException ex] {System.out.println(ex); errors = true; }

//a statement is either a declaration or function
statement
 : dec | funct;

//The tree walker
class AJYNWalker extends TreeParser;

//The java code used in order to create and manipulate the AJYN Tree
//data structure
{
	java.util.Vector words;  //lookup table for identifiers

	//initialization and declarations needed for scoping issues
	//numLits refers to the number of literals defined in the
	//init() function below

	AJYNTree tr;
	boolean prog = false;
	boolean error = false;
	int r = 0;
	int b = 0;
	int g = 0;
	int count = 0;  //number of objects declared
	int numLits = 7;
	int lastFrame = 0;

	//Searches for a string in the words vector

	boolean containsKey(String s)
	{
		boolean result = false;

		for (int i=0; i<words.size(); i++)
		{
			if (s.equals((String) words.elementAt(i)))
			{
				result = true;
```

```
                }
            }

            return result;
        }

        //returns the object identifier number which is the current value
        //of count when the object is added.  numLits offsets it in order
        //to make it 99 and since 7 other objects are added to the words vector.
        //the index of the word is returned

        int get(String s)
        {
            int a = -92;

            for (int i=0; i<words.size(); i++)
            {
                if (s.equals((String) words.elementAt(i)))
                {
                    a = i;
                }
            }

            return a;
        }

        //adds keywords to the words vector and initializes the
        //AJYNTree - called when Program ident = params is read, which
        //should be the first line

        void init()
        {
            String w = "MAX";
            words = new java.util.Vector();
            words.addElement(w);  w = "line";
            words.addElement(w); w = "ellipse";
            words.addElement(w); w = "poly";
            words.addElement(w); w = "point";
            words.addElement(w); w = "rect";
            words.addElement(w); w = "group";
            words.addElement(w);
            tr = new AJYNTree();
        }
}


//the expression walker - all expressions return type float
expr returns [float r2]
{
        float a, b2;
        r2 = 0;
}
        :#(PLUS a=expr b2=expr) {r2=a+b2;}
        |#(STAR a=expr b2=expr) {r2=a*b2;}
        |#(SUB a=expr b2=expr) {r2=a-b2;}
        |#(DIV a=expr b2=expr) {if (b2 != 0) { r2=a/b2; } else { System.out.println("error div by 0"); } }
```

```
        |#(LP a=expr) {r2=a;}
        |i:INT { r2 = (float) Integer.parseInt(i.getText());}
        |f:NUM { r2 = (float) Float.parseFloat(f.getText());}
        ;

//the integer type parameter used for frames and sizes
//MAX is a constant = 65536
num returns [int i]
{
        i = 0;
}
  :j:INT { i = Integer.parseInt(j.getText()); }
  |j2:"MAX" { i = 65536; }
  ;

//the rules for creating instances of objects or using functions

//first, the walker checks to see if the init2 statement has been
//declared.  if it has, the boolean prog == true.  When each object
//is declared, the first thing checked for is a repeat in the identifier
//secondly, the values of parameters that must be above 0 are checked.
//any error found will cause error to be true and no x.ir file will be written.

statement returns [int state]
{

        float p1, p2, p3, p4, p5, p6;
        state = 0;
        int o1, o2, o3, o4;

        if (!prog)
        {
                System.out.println("ERROR! Program was not defined first!");
                System.exit(1);
        }


}
        :#("ellipse" s:ID p1=expr p2=expr p3=expr p4=expr
        {
                if (!containsKey(#s.getText()))
                {
                        if (p3 <= 0 || p4 <=0)
                        {
                                System.out.println("Error - can not have negative or 0 value lengths");
                                System.out.println("for ellipse "+#s.getText());  error = true;
                        }
                        words.addElement(#s.getText());
                        tr.addObject(new Ellipse(count, p1, p2, p3, p4, r, g, b));
                        count++;
                }
                else
                { System.out.println("ERROR - "+#s.getText()+" already an identifier"); error = true;}
        })

        | #("line" s2:ID p1=expr p2=expr p3=expr p4=expr
```

```
{
	//System.out.println("line");
	if (!containsKey(s2.getText()))
	{
		words.addElement(#s2.getText());
		tr.addObject(new Line(count, p1, p2, p3, p4, r, g, b));
		count++;
	}
	else
	{ System.out.println("ERROR - "+#s2.getText()+" already an identifier"); error = true; }
})


| #("rect" s3:ID p1=expr p2=expr p3=expr p4=expr
{
	//System.out.println("rect");
	if (!containsKey(#s3.getText()))
	{

		words.addElement(#s3.getText());
		tr.addObject(new Rect(count, p1, p2, p3, p4, r, g, b));
		count++;
	}
	else
	{ System.out.println("ERROR - "+#s3.getText()+" already an identifier"); error = true; }
})

| #("poly" s4:ID
{
	java.util.Vector v = new java.util.Vector();
	int n=-1;
	if (!containsKey(#s4.getText()))
	{
		words.addElement(#s4.getText());
		n = count;
		count++;
	}
	else
	{System.out.println("ERROR - "+#s4.getText()+" already an identifier"); error = true; }
}//end after s4:ID

p1=expr p2=expr { v.add(new Float(p1));  v.add(new Float(p2)); }
(p3=expr p4=expr {v.add(new Float(p3)); v.add(new Float(p4));})+
{
	if (n != -1)
	{

		tr.addObject(new Poly(n, v, r, g, b));
	}
} )

| #("point" s5:ID p1=expr p2=expr
{
	//System.out.println("point");
	if (!containsKey(#s5.getText()))
	{
```

```
                words.addElement(#s5.getText());
                tr.addObject(new Point(count, p1, p2, r, g, b));
                count++;
            }
            else
            {System.out.println("ERROR - "+#s5.getText()+" already an identifier");  error = true;}
        })

        //Groups can not group other groups properly

        | #("group" s6:ID
        {
            //System.out.println("group");
            Groups gr = new Groups();
            int n1 = -1;
            if (!containsKey(#s6.getText()))
            {
                words.addElement(#s6.getText());
                n1 = count;
                count++;
                //System.out.println(""+n1+"...\n");
            }
            else
            {System.out.println("ERROR - "+#s6.getText()+" already an identifier"); error = true;}
        }

        //Here the identifiers to be added to the group are checked to make sure they exist, are not
        //groups, and have no transforms attached to them already.

        (y:ID
        {
            int ind = -99;
            if (n1 != -1)
            {
                gr.setID(n1);
                ind = get(#y.getText()) - numLits;
                //System.out.println("Made it here");
                if (ind == -99)
                {
                    System.out.println("ERROR - no object with ident "+#y.getText()+"");
error = true;       }
                else
                {
                    //System.out.println("ind = "+ind+" and n1 = "+n1+"");
                    if (ind != n1)
                    {
                        if (tr.hasTransform(ind))
                        {
                            System.out.println("Transforms will be ignored since
"+#y.getText()+" enters a group.");
                        }

                        if (tr.inVis(ind))
                        {
                            System.out.println("Invisibility will be ignored since
"+#y.getText()+" enters a group.");
```

```
                                        }
                                        if (!tr.isG(ind))
                                        {
                                                gr.addElement(ind);
                                        }
                                        else
                                        {
                                                System.out.println("Can not group a group to a
group!");  error = true;
                                        }
                                        gr.addElement(ind);
                                        //System.out.println("added element w/ ind "+ind);
                                }
                                else
                                {
                                        System.out.println("ERROR - can not group same group to
group");  error = true;
                                }
                        }
                }
        })+ p1=expr p2=expr
        {
        if (n1 != -1)
                gr.setPos(p1, p2);
                tr.addGroup(gr);
                //System.out.println("added gr");
        })

        //functions look for valid identifiers, check the frames, and see if the stop frame
        //exceeds MAX or the Programs last frame

        | #("rotate" s7:ID p1=expr o1=num o2=num
        {
                //System.out.println("rotate");
                int ind = get(#s7.getText()) - numLits;
                if (ind == -99)
                {
                        System.out.println("ERROR - no object with ident "+#s7.getText()+"");  error =
true;
                }
                else
                {
                        //System.out.println("Ind = "+ind);
                        if (tr.inGroup(ind))
                        {
                                System.out.println("Transforms will be ignored since
"+#s7.getText()+" is in a group.");
                        }

                        if(tr.overlap(22, ind, o1, o2))
                        {
                                System.out.println("ERROR - Rotation overlap on "+#s7.getText());
error = true;
                        }
                        else
                        {
```

```
                                        if (o1 < o2)
                                        {
                                                if (o2 > 65536)
                                                {
                                                        System.out.println("Stop frame for Rotating
"+#s7.getText()+" reset to MAX");
                                                        o2 = 65536;
                                                }
                                                if (o1 > lastFrame || o2 > lastFrame)
                                                {
                                                        System.out.println("Error - frame animation can not
exceed number of frames in Program");
                                                        error = true;
                                                }
                                                tr.addTrans(new Rotate(ind, p1, o1, o2));
                                        }
                                        else
                                        {
                                                System.out.println("Error rotating "+#s7.getText()+" - start
frame must be smaller than stop");
                                                error = true;
                                        }
                                }
                        }
                })


                |#("scale" s8:ID p1=expr o1=num o2=num
                {
                        //System.out.println("Scale");
                        int ind = get(#s8.getText()) - numLits;
                        if (ind == -99)
                        {
                                System.out.println("ERROR - no object with ident "+#s8.getText()+""); error =
true;
                        }
                        else
                        {
                                if (tr.inGroup(ind))
                                {
                                        System.out.println("Transforms will be ignored since
"+#s8.getText()+" is in a group.");
                                }

                                if (tr.overlap(23, ind, o1, o2))
                                {
                                        System.out.println("ERROR - Scale overlap on "+#s8.getText()); error
= true;
                                }
                                else
                                {
                                        if (o1 < o2)
                                        {
                                                if (o2 > 65536)
                                                {
```

```
                                        System.out.println("Stop frame for Scaling
"+#s8.getText()+" reset to MAX");
                                        o2 = 65536;
                                }
                                if (o1 > lastFrame || o2 > lastFrame)
                                {
                                        System.out.println("Error - frame animation can not
exceed number of frames in Program");
                                        error = true;
                                }
                                tr.addTrans(new Scale(ind, p1, o1, o2));
                        }
                        else
                        {
                                System.out.println("Error scaling "+#s8.getText()+" - start
frame must be smaller than stop");
                                error = true;
                        }
                }
        }
    })

    |#("translate" s9:ID p1=expr p2=expr o1=num o2=num
    {
        //System.out.println("Translate");
        int ind = get(#s9.getText()) - numLits;
        if (ind == -99)
        {
                System.out.println("ERROR - no object with ident "+#s9.getText()+""); error =
true;
        }
        else
        {
                if (tr.inGroup(ind))
                {
                        System.out.println("Transforms will be ignored since
"+#s9.getText()+" is in a group.");
                }

                if (tr.overlap(21, ind, o1, o2))
                {
                        System.out.println("ERROR - Translate overlap on "+#s9.getText());
error = true;  }
                else
                {
                        if (o1 < o2)
                        {
                                if (o2 > 65536)
                                {
                                        System.out.println("Stop frame for Translating
"+#s9.getText()+" reset to MAX");
                                        o2 = 65536;
                                }
                                if (o1 > lastFrame || o2 > lastFrame)
                                {
```

```
                                        System.out.println("Error - frame animation can not
exceed number of frames in Program");
                                        error = true;
                                }
                                tr.addTrans(new Translate(ind, p1, p2, o1, o2));
                        }
                        else
                        {
                                System.out.println("Error translating "+#s9.getText()+" - start
frame must be smaller than stop");
                                error = true;
                        }
                }
        }
})

//the only issue with ActiveColor is that values must range from 0 - 255
//this fixes anything out of bounds automatically

|#("ActiveColor" a:INT b2:INT c:INT
{
        //System.out.println("Changing color");
        int d = Integer.parseInt(a.getText());
        int e = Integer.parseInt(b2.getText());
        int f = Integer.parseInt(c.getText());

        if (d > 255)
                d = 255;

        if (e > 255)
                e = 255;

        if (f > 255)
                f = 255;

        if (d < 0)
                d = 0;

        if (e < 0)
                e = 0;

        if (f < 0)
                f = 0;

        r = d; g = e; b = f;
})


|#("setInvisible" s10:ID o1=num o2=num
{
        //System.out.println("Setting invis");
        int ind = get(#s10.getText()) - numLits;
        if (ind == -99)
        {
                System.out.println("ERROR - no object with ident "+#s10.getText()+""); error =
true;
```

```
                }
                else
                {
                        if (tr.inGroup(ind))
                        {
                                System.out.println("Invisibility will be ignored since
"+#s10.getText()+" is in a group.");
                        }

                        if (o1 < o2)
                        {
                                if (o2 > 65536)
                                {
                                        System.out.println("Stop frame for Invisibility on
"+#s10.getText()+" reset to MAX");
                                        o2 = 65536;
                                }
                                if (o1 > lastFrame || o2 > lastFrame)
                                {
                                        System.out.println("Error - frame animation can not exceed
number of frames in Program");
                                        error = true;
                                }
                                tr.addVis(new VisNodes(ind, o1, o2));
                        }
                        else
                        {
                                System.out.println("Error setting " +#s10.getText()+" Invisible - start
frame must be less than stop");
                                error = true;
                        }
                }
        })
        ;

//Here the program is declared
//w3 is the total number of frames
//w1 w2 = x and y size respectively

init2 returns [int state2]
{
                state2 = 0;
                int w1, w2, w3;
}
 : #("Program" w1=num w2=num w3=num
         {
                if (!prog)
                {
                        init();
                        lastFrame = w3;
                        tr.createPNode(w1, w2, w3);
                        prog = true;
                        //System.out.println("init tree");
                }
                else
                {
```

```
                        System.out.println("ERROR! Program already defined!"); error = true;
                }
        })
        ;

        //This creates the x.ir file by calling the writeToIR method
        //defined in the AJYNTree class.  The method is only called if there
        //are no errors in the semantics.
file {int ble; }
 : ble=init2 (ble=statement)+

{
if (!error)
{
 System.out.println("**********");
        //System.out.println(tr.toString());
        tr.writeToIR();
}
else { System.out.println("Errors found - aborting");  System.exit(1); }
}
;
```

```
//Authors: Neel Goyal and Ananya Das
import java.io.*;

//The big tree that will output the x.ir file
//Holds a node for the init statement, a node for all the objects,
//a node for all the groups, a node for all the transforms, and a
//node for the visibility

public class AJYNTree
{
   PNode p;
   ONode o;
   GNode g;
   TNode t;
   VNode v;

   //File to write to
   File IR = new File("x.ir");

   public AJYNTree()
   {
         o = new ONode();
         t = new TNode();
         g = new GNode();
         v = new VNode();

   }

//Creates a PNode which holds the size of the x and y dimensions,
//and total frames.

   void createPNode(int sx, int sy, int f)
   {
         p = new PNode(sx, sy, f);
   }

//Returns the boolean called from the GNode isG - sees if an object is
//actually a group

   boolean isG(int a)
   {
         return g.isG(a);
   }

//Adds a line, ellipse, poly, rect or point
//All classes above extend the Objects class

   void addObject(Objects a)
   {
         o.addElement(a);
   }

//Adds a translate, scale, or rotate Transform
//All transforms extend the Transforms class

   void addTrans(Transforms a)
```

```
    {
        t.addElement(a);
    }

//Adds a Groups node to the tree

    void addGroup(Groups a)
    {
        g.addElement(a);
    }

//Adds a VisNodes to the tree

    void addVis(VisNodes a)
    {
        v.addElement(a);
    }

//Checks to see if a transformation overlaps another of the same type
//on the same object.

    boolean overlap(int a, int b, int c, int d)
    {
        return t.overlap(a, b, c, d);
    }

//Checks to see if an object has a transformation associated with it

    boolean hasTransform(int a)
    {
        return t.hasTrans(a);
         }

//Checks to see if an Object is in a Group

         boolean inGroup(int a)
         {
             return g.inG(a);
         }

//Checks to see if an Object has a visibility node associated with it

         boolean inVis(int a)
         {
             return v.inV(a);
         }

//Prints out the x.ir file string

    public String toString()
    {
        String r = p.toString();
        r += o.toString();
        r+= g.toString();
        r += t.toString();
        r += v.toString();
```

```
            r += "9 0\n";

            return r;
    }

//Calls toString to write the IR

    public void writeToIR()
    {

            String line = toString();

            try
               {
                        FileWriter fw = new FileWriter(IR);
                        BufferedWriter bw =  new BufferedWriter (fw);
                        PrintWriter outFile = new PrintWriter (bw);

                        outFile.print (line); //prints line to file
                        outFile.close(); //closes file
               }

             //catches FileNotFoundException when printing to file
            catch (FileNotFoundException exception)
               {
                  System.out.println("The file " + IR + " was not found.");
               }

            //catches IOException when printing to file
            catch (IOException exception)
               {
                  System.out.println(exception);
               }

    }
}
```

```
//Author: Ananya Das

//Basic Objects definition of an Ellipse.  Each has a x,y center and a
//radius in the x dir and a radius in the y dir.  params1-4 represent those
//respectively.
//For the x.ir, each type Objects needs an ID number and an RGB value
//They also need a type to specify the type of Object

public class Ellipse extends Objects
{
   public double param1, param2, param3, param4;
   public int type_id_num = 14;
   public int id_num;
   public int num_params = 4;
   public int colorR, colorG, colorB;

//Create an Ellipse

   public Ellipse(int iNum, double p1, double p2, double p3, double p4, int cR, int cG, int cB)
   {
          id_num = iNum;
          param1 = p1;
          param2 = p2;
          param3 = p3;
          param4 = p4;
          colorR = cR;
          colorG = cG;
          colorB = cB;

   }

//Print out a string as per the x.ir

   public String toString()
   {
          String result = "" + type_id_num + " " + id_num + " " + num_params + " " + param1 + " " +
param2 + " " + param3 + " " + param4 + " " + colorR + " " +colorG + " " + colorB+"\n";

          return result;

   }
}
```

```
//Author: Neel Goyal

import java.util.*;

//Basic Node that holds all the Groups Nodes

public class GNode extends MNode
{
	public GNode()
	{
		sectionID = 3;
		v = new java.util.Vector();
	}

//Adds a Groups node to the vector
	void addElement(Groups g)
	{
		v.add(g);
	}

//Prints the section 3 0 and then the Groups Node string

	public String toString()
	{
		String r = "3 0\n";
		for (int i = 0; i < v.size(); i++)
		{
			Groups g = (Groups) v.elementAt(i);
			r += g.toString();
		}

		return r;
	}

//Method that checks to see if an object with id number a is actually
//a group - groups can not be grouped

   boolean isG(int a)
   {
	boolean is = false;
	for (int i=0; i < v.size(); i++)
	   {
		Groups g = (Groups) v.elementAt(i);
		if (g.idn == a)
		   is = true;
	   }

	return is;
   }

//Checks to see if an object is actually in a group.  Used when a transform
//is declared on an object that may be in a group

	boolean inG(int a)
	{
		boolean in = false;
```

```
                for (int i=0; i < v.size(); i++)
                {
                        Groups g = (Groups) v.elementAt(i);
                        in = g.inGr(a);
                }

                return in;
        }
}
```

```
//Author:  Neel Goyal

import java.util.*;

//A node that holds information regarding a specific group declared
//Each has an id number, the number of objects in the group, and a
//starting x,y position.  They also have a Vector for holding
//Object ID numbers

public class Groups
{
	int idn, numo;
	float startx, starty;

	Vector v;

//Constructor for initializing the Vector, but not the actual group

	public Groups()
	{
		idn = -95;
		v = new Vector();
	}

//Initializes the group

	public Groups(int id, float sx, float sy)
	{
		idn = id;
		startx = sx;
		starty = sy;
		numo = 0;
		v = new Vector();
	}

//Initializes a group with a specific ID

	public Groups(int id)
	{
		idn = id;
		v = new Vector();
		numo = 0;
	}

//Sets the ID of a group - used when Group() constructor is called

	void setID(int id)
	{
		idn = id;
	}

//Sets the initial position of a group

	void setPos(float sx, float sy)
	{
		startx = sx;
```

```
		starty = sy;
	}

//Checks to see if object a exists in this particular group

	boolean inGr(int a)
	{
		boolean contains = false;
		for (int i = 0; i < v.size(); i++)
		{
			Integer k = (Integer) v.elementAt(i);
			if (k.intValue() == a)
			{
				contains = true;
			}
		}

		return contains;
	}

//Adds an object with id number i to the group

	void addElement(int i)
	{
		numo++;
		Integer j = new Integer(i);
		v.add(j);
	}

//Prints out a string as per the x.ir

	public String toString()
	{
		String r = "19 "+idn+" "+numo+" ";
		for (int i = 0; i < v.size(); i++)
		{
			Integer k = (Integer) v.elementAt(i);
			int g = k.intValue();

			r += ""+g+" ";
		}

		r += ""+startx+" "+starty+"\n";

		return r;
	}
}
```

```
//Author: Ananya Das

//The Line Node which inherits properties of Objects Nodes
//Each Line has 4 parameters, which are x1,y1,x2,y2, respectively -
//the line is drawn from (x1,y1) to (x2, y2)
//For the x.ir, each type Objects needs an ID number and an RGB value
//They also need a type to specify the type of Object

public class Line extends Objects
{
   public double param1, param2, param3, param4;
   public int type_id_num = 12;
   public int id_num;
   public int num_params = 4;
   public int colorR, colorG, colorB;

//Constructor for the line

   public Line(int iNum, double p1, double p2, double p3, double p4, int cR, int cG, int cB)
   {
         id_num = iNum;
         param1 = p1;
         param2 = p2;
         param3 = p3;
         param4 = p4;
         colorR = cR;
         colorG = cG;
         colorB = cB;

   }

//String formatted for the ir file.

   public String toString()
   {
         String result = "" + type_id_num + " " + id_num + " " + num_params + " " + param1 + " " +
param2 + " " + param3 + " " + param4 + " " + colorR + " " +colorG + " " + colorB+"\n";

         return result;

   }
}
```

```
//Author: Neel Goyal

import java.util.*;

//The base class for the Nodes that make up an AJYNTree
//Each will have an ID and a Vector.  The constructor is left
//empty

public class MNode
{
   int sectionID;
   Vector v;

   public MNode()
   {
            ;
   }

}
```

```
//Author: Ananya Das

//Base class for Objects types
//Used so Objects typed can be put in the vector held in the
//ONode - since all Objects will have their own unique toString
//it works well.

public class Objects
{
   public Objects()
   {

   }
}
```

```
//Author: Neel Goyal
import java.util.*;

//The class that holds the vector of type Objects
//It also prints the 2 section defined in the ir spec

public class ONode extends MNode
{

  //constructor

        public ONode()
        {
                sectionID = 2;
                v = new Vector();
  }

        //adds an Objects type to the vetor

  void addElement(Objects o)
  {
                v.add(o);
  }

        //generates a string as per the ir spec

  public String toString()
  {
                String r = ""+sectionID+" 0\n";
                for (int i = 0; i < v.size(); i++)
                {
                        Objects n = (Objects) v.elementAt(i);
                        r += n.toString();
                }

                return r;
  }

}
```

```
//Author: Neel Goyal

//Holds information regarding the program's instantiation
//It holds the number of frames and the dimensions of the
//window

public class PNode extends MNode
{
   int sizex, sizey, frames;

   public PNode(int x, int y, int f)
   {
          sizex = x;
          sizey = y;
          frames = f;
   }

//Prints the section as per the ir spec

   public String toString()
   {
          String r = "1 3 "+sizex+" "+sizey+" "+frames+"\n";
          return r;
   }
}
```

```
//Author: Ananya Das

//Objects definition for a Point.  Params1-2 represent the x,y position
//of the point, respectively.
//For the x.ir, each type Objects needs an ID number and an RGB value
//They also need a type to specify the type of Object

public class Point extends Objects
{
   public double param1, param2;
   public int type_id_num = 11;
   public int id_num;
   public int num_params = 2;
   public int colorR, colorG, colorB;

   public Point(int iNum, double p1, double p2, int cR, int cG, int cB)
   {
          id_num = iNum;
          param1 = p1;
          param2 = p2;
          colorR = cR;
          colorG = cG;
          colorB = cB;

   }

   //Print a string as per the x.ir spec

   public String toString()
   {
          String result = "" + type_id_num + " " + id_num + " " + num_params + " " + param1 + " " +
param2 + " " + colorR + " " +colorG + " " + colorB+"\n";

          return result;

   }
}
```

```
//Author: Ananya Das

import java.util.*;

//Objects defintion for a Polygon with user defined number of points
//Parameters for the points are passed in the form of a vector that is
//checked in the parser (it must have an even number of elements)
//For the x.ir, each type Objects needs an ID number and an RGB value
//They also need a type to specify the type of Object

public class Poly extends Objects
{
   public Vector params = new Vector(); //a polygon must have at least 3 pts
                                        //(x and y coords so 6 params)
   public int type_id_num = 15;
   public int id_num;
   public int num_params;
   public int colorR, colorG, colorB;

   public Poly(int iNum, Vector p, int cR, int cG, int cB)
   {
          id_num = iNum;

          params = p;

          num_params = params.size();

          colorR = cR;
          colorG = cG;
          colorB = cB;

   }

//Prints the string as per the x.ir spec

   public String toString()
   {
          String r = "" + type_id_num + " " + id_num + " " + num_params + " ";
          for (int i = 0; i < params.size(); i++)
          {
             Float k = (Float) params.elementAt(i);
             float g = k.floatValue();
             r += ""+g+" ";
          }

          r += ""+ colorR + " " +colorG + " " + colorB+"\n";

          return r;

   }
}
```

```
//Author:  Ananya Das

//Basic class for holding rectangles.  Params1-4 represent the upper left
//corner x,y position, and the width and height, respectively.
//For the x.ir, each type Objects needs an ID number and an RGB value
//They also need a type to specify the type of Object

public class Rect extends Objects
{
   public double param1, param2, param3, param4;
   public int type_id_num = 13;
   public int id_num;
   public int num_params = 4;
   public int colorR, colorG, colorB;

   //Create a rectangle

   public Rect(int iNum, double p1, double p2, double p3, double p4, int cR, int cG, int cB)
   {
          id_num = iNum;
          param1 = p1;
          param2 = p2;
          param3 = p3;
          param4 = p4;
          colorR = cR;
          colorG = cG;
          colorB = cB;

   }

   //Print out rectangle as per x.ir spec

   public String toString()
   {
          String result = "" + type_id_num + " " + id_num + " " + num_params + " " + param1 + " " +
param2 + " " + param3 + " " + param4 + " " + colorR + " " +colorG + " " + colorB+"\n";

          return result;

   }
}
```

```
//Author: Neel Goyal

//Transforms definition for a Rotation.  Each rotation
//has a number of degrees that it rotates an object by.
//For the x.ir, each type Transforms needs a type to specify the
//type of Transform, and an object or group id number to know what
//it is transforming

public class Rotate extends Transforms
{
	float degs;

	//constructor

	public Rotate(int obj, float d, int startf, int stopf)
	{
		super(obj, startf, stopf);
		degs = d;
		numParams = 3;
		type = 22;
	}

	//Generates string as per ir spec

	public String toString()
	{
		String g = "22 "+object+" 3 "+degs+" "+start+" "+stop+"\n";
		return g;
	}
}
```

```
//Author: Neel Goyal

//Transforms definition of Scale.  Each scale has a factor that
//an object is scaled by.
//For the x.ir, each type Transforms needs a type to specify the
//type of Transform, and an object or group id number to know what
//it is transforming

public class Scale extends Transforms
{
	double factor;

	public Scale(int obj, double d, int startf, int stopf)
	{
		super(obj, startf, stopf);
		factor = d;
		numParams = 3;
		type = 23;
	}

	//Generates a string as per ir spec

	public String toString()
	{
		String g = "23 "+object+" 3 "+factor+" "+start+" "+stop+"\n";
		return g;
	}
}
```

```
//Author:  Neel Goyal

import java.util.*;

//The Node that holds all the Transforms Nodes.

public class TNode extends MNode
{

   public TNode()
   {
         sectionID = 4;
         v = new Vector();
   }

//adds a Transforms Node to the group
   void addElement(Transforms o)
   {
         v.add(o);
   }

//Checks to see if an object with ID Number ask has a Transform associated
//with it - used when an object enters a group.

   boolean hasTrans(int ask)
   {
         boolean has = false;
               for (int i=0; i < v.size(); i++)
               {
                     Transforms n = (Transforms) v.elementAt(i);
                     if (n.object == ask)
                     {
                           has = true;
                     }
               }

               return has;
   }

//Function used to see if a transform of the same type overlaps in frame
//animation

   public boolean overlap(int transtype, int id_num, int start, int stop)
   {
         boolean result = false;
         for(int i=0; i < v.size(); i++)
            {
               Transforms n = (Transforms) v.elementAt(i);
               if (n.type == transtype && n.object == id_num)
                  {
                     if (stop <= n.stop && stop >= n.start)
                        result = true;
                     if (start >= n.start && start <= n.stop)
                        result = true;
                     if (start <= n.start && stop >= n.stop)
                        result = true;
```

```
                                if (start >= n.start && stop <= n.stop)
                                   result = true;
                                if (start == n.stop && n.start < stop)
                                        result = false;
                                if (start < n.stop && stop == n.start)
                                        result = true;
                        }
                   }
                return result;
   }

//Refers to section 4 0 in the .ir - prints 4 0 and then every transform
//Node's toString.

   public String toString()
   {

        String r = ""+sectionID+" 0\n";
        for (int i = 0; i < v.size(); i++)
        {
           Transforms n = (Transforms) v.elementAt(i);
           r += n.toString();
        }
        return r;
   }

}
```

```
//Author: Neel Goyal

//Base class for holding information about a Transforms
//Each Transforms has an object that it's transforming,
//a frame range, and the number of parameters it passes
//to the x.ir

public class Transforms
{
        public int type, object;
        public int numParams;
        public int start, stop;

//constructors

   public Transforms(int obj, int startf, int stopf)
   {
                object = obj;
                start = startf;
                stop = stopf;
   }

    public Transforms()
   {
   }

}
```

```
//Author: Neel Goyal

//Transforms definition for a Translation
//Each translation takes an x,y coordinate (newx, newy) that it must
//reach in the frame range, specified by start and stop
//For the x.ir, each type Transforms needs a type to specify the
//type of Transform, and an object or group id number to know what
//it is transforming

public class Translate extends Transforms
{
	float newx, newy;

	public Translate(int obj, float dx, float dy, int startf, int stopf)
	{
	   //super(obj, startf, stopf);
	   start = startf;
	   stop = stopf;
	   object = obj;

		newx = dx;
		newy = dy;
		numParams = 4;
		type = 21;
	}

	//Prints out the translation as per x.ir spec

	public String toString()
	{
		String g = "21 "+object+" 4 "+newx+" "+newy+" "+start+" "+stop+"\n";
		return g;
	}
}
```

```
//Author:  Neel Goyal

//Node that holds information about a visibility transform
//each holds the objects id number, and a frame range for the
//object to be invisible

public class VisNodes
{
	int obj, start, stop;

	//create a visnode

	public VisNodes(int o, int s, int p)
	{
		obj = o;
		start = s;
		stop = p;
	}

	//print out as per x.ir spec

	public String toString()
	{
		String r = "30 "+obj+" "+start+" "+stop+"\n";
		return r;
	}

}
```

```
//Author Neel Goyal

import java.util.*;

//Class file for the Node that contains Visibility Nodes

public class VNode extends MNode
{
	public VNode()
	{
		v = new java.util.Vector();
		sectionID = 5;
	}

//Add a new VisNodes
	void addElement(VisNodes vis)
	{
		v.add(vis);
	}

//Checks to see if an object with ID number held in the value of a
//has a visibility transform associated with it.  Used when an object
//is added to a group.

	boolean inV(int a)
	{
		boolean in = false;
		for (int i = 0; i<v.size(); i++)
		{
			VisNodes m = (VisNodes) v.elementAt(i);
			if (m.obj == a)
			{
				in = true;
			}
		}

		return in;
	}

//Prints section 5 0 for the ir file, followed by the VisNodes strings

	public String toString()
	{
		String r = ""+sectionID+" 0\n";
		for (int i = 0; i < v.size(); i++)
		{
			VisNodes m = (VisNodes) v.elementAt(i);
			r += m.toString();
		}

		return r;
	}
}
```

Backend .cpp file - Written by Yaniv Schiller

```
#pragma warning (disable:4786)

/*
AJYN Backend

each object has an identifier_number
each object has a previously know number of paramters
then we follow each with color numbers

Each line can be one of the following:

section_id number_params params

OBJECTS SECTION:
type_id_number identifier_number number_of_paramaters parameters colorR colorG colorB

GROUPS SECTION:
group_id_number identifier_number number_of_objects identifier_numbers x y

FUNCTION SECTION:
func_id_number obj_identifier_number number_of_paramaters parameters

INVISIBILITY SECTION
obj_identifier_number start_frame end_frame


section_id          #parameters
program       01       3
objects       02       0
groups        03       0
funcs         04       0
visibs        05       0
EOF           09       0

types_id_number       #parameters
point         11       2
line          12       4
rect          13       4
ellipse       14       4
poly          15       N

group_id_number:
group         19

func_id_number       id       #parameters
translate            21       4
rotate               22       3
scale                23       3
*/

#include<iostream>
#include<fstream>
#include<vector>
```

```
#include<map>
#include<cmath>
#include<string>
#include <stdio.h>
#include "mingpp.h"

using namespace std;

const unsigned short LINEW = 10;
class object{ // groups too
public:
	int type;//point 11,line 12,rect 13,ellipse 14,poly 15
	int toDisplay, isVisible;
	long idenNumber;
	float centerx;
	float centery;
	float initx;
	float inity;
	float initscale;
	float initrot;
	vector<float> params; //first two usually x and y position

	union {
		struct {//length (thanks shira) for rect
			float tlx,tly;
			float width,height;
		}rect;
		struct {//elipse
			float x;
			float y;
		}radius;
		struct {//line
			float x1,y1;
			float x2,y2;
			float length;
		}line;
	} extras;
	struct{
		int R,G,B;
	} color;
	void getShape(SWFShape *);
	void getShape(SWFShape * s,float offsetx, float offsety);
};
class funcs{
public:
		int type; //translate 21,rotate 22,scale 23
		long idenNumber;
		int startframe,endframe;
		union{
			struct {
				float x,y;
			}newpos;
			float rotateDegrees;
			float scaleSize;
		}details;
```

```
};
class visib{
public:
	long idenNumber;
	int startframe,endframe;
};

class Program{
public:
	float width, height;
	long frames;

};


int nActiveObjects;
vector<object> vObj;
map<int,int> mObj;//which position in the vecotr is this ID is
vector<funcs> vFn;
vector<visib> vVis;

ifstream fin("x.ir");

void ellipse(SWFShape * shape, float cx, float cy, float rx, float ry);

int main(){
	nActiveObjects=0;
	long rowType;
	int state=0, numParams;
	int error=0;
	Program theP;
	while(state!=9)//  for(int i=0;i<5;i++)
	{
		fin>>rowType;
//cout<<"READ this number: "<<rowType<<endl ;
		switch(rowType)
		{
		case 1://program	01	3
			if(state==0){
				state=1;
				fin>>numParams>>theP.width>>theP.height>>theP.frames;
			}
			else
				error=1;
			break;
		case 2://objects	02	0
			if(state>=1){
				state=2;
//				int tmp=fin.peek();
//				if(tmp!='\n')
					fin>>numParams;
			}
			else
				error=1;
			break;
		case 3://groups	03	0
```

```
                        if(state>=1){
                                state=3;
//                              int tmp=fin.peek();
//                              if(tmp!='\n')
                                        fin>>numParams;
                        }
                        else
                                error=1;
                        break;
                case 4://funcs          04      0
                        if(state>=1){
                                state=4;
//                              int tmp=fin.peek();
//                              if(tmp!='\n')
                                        fin>>numParams;
                        }
                        else
                                error=1;
                        break;
                case 5://visibs         05      0
                        if(state>=1){
                                state=5;
//                              int tmp=fin.peek();
//                              if(tmp!='\n')
                                        fin>>numParams;
                        }
                        else
                                error=1;
                        break;
                case 9:
                        state=9;
                        break;
                                //type_id_number identifier_number number_of_paramaters parameters
        colorR colorG colorB
                case 11://point         11      2
                        if(state==2)
                        {
                                //add the obect...
                                object x; int nump;
                                x.type=11;
                                x.toDisplay=1;
                                fin>>x.idenNumber>>nump;
                                x.params.resize(nump);
                                for(int i=0;i<nump;i++)
                                        fin>>x.params[i];
                                fin>>x.color.R>>x.color.G>>x.color.B;
                                x.initx=x.centerx=x.params[0];//first paramter is x
                                x.inity=x.centery=x.params[1];//second paramter is y
                                x.initscale=1;
                                x.initrot=0;
                                vObj.push_back(x);
                                nActiveObjects++;
                                mObj[x.idenNumber]=vObj.size()-1;
                        }
                        else
                                error=1;
```

```
                        break;
                case 12://line          12      4
                        if(state==2)
                        {
                                //add the obect...
                                object x; int nump;
                                x.type=12;
                                x.toDisplay=1;
                                fin>>x.idenNumber>>nump;
                                x.params.resize(nump);
                                for(int i=0;i<nump;i++)
                                        fin>>x.params[i];
//                              if(fin.peek()!='\n')
                                fin>>x.color.R>>x.color.G>>x.color.B;
//                              else
//                                      x.color.R=x.color.G=x.color.B=0;
//
        cout<<"COLOR:"<<x.color.R<<'\t'<<x.color.G<<'\t'<<x.color.B<<endl;;
                                x.extras.line.x1=x.params[0];//first paramter is x1
                                x.extras.line.y1=x.params[1];//second paramter is y1
                                x.extras.line.x2=x.params[2]; //third paramter is x2
                                x.extras.line.y2=x.params[3];//fourthparamter is y2

                                float dx=(x.extras.line.x2-x.extras.line.x1) ,dy=(x.extras.line.y2-
x.extras.line.y1);
                                x.initx=x.centerx=x.extras.line.x1+(dx)/2;
                                x.inity=x.centery=x.extras.line.y1+(dy)/2;
                                x.extras.line.length=sqrt(dx*dx+dy*dy);
                                x.initscale=1;
                                x.initrot=0;


                                vObj.push_back(x);
                                nActiveObjects++;
                                mObj[x.idenNumber]=vObj.size()-1;
                        }
                        else
                                error=1;
                        break;
                case 13://rect          13      4
                        if(state==2)
                        {
                                //add the obect...
                                object x; int nump;
                                x.type=13;
                                x.toDisplay=1;
                                fin>>x.idenNumber>>nump;
                                x.params.resize(nump);
                                for(int i=0;i<nump;i++)
                                        fin>>x.params[i];
//                              if(fin.peek()!='\n')
                                fin>>x.color.R>>x.color.G>>x.color.B;
//                              else
//                                      x.color.R=x.color.G=x.color.B=0;
//
        cout<<"COLOR:"<<x.color.R<<'\t'<<x.color.G<<'\t'<<x.color.B<<endl;;
```

```
                                        x.extras.rect.tlx=x.params[0];//first paramter is top left x
                                        x.extras.rect.tly=x.params[1];//second paramter is top left y
                                        x.extras.rect.width =x.params[2]; //third paramter is width
                                        x.extras.rect.height=x.params[3];//fourthparamter is height
                                        //calculate center
                                        x.initx=x.centerx=x.extras.rect.tlx+(x.extras.rect.width)/2;
                                        x.inity=x.centery=x.extras.rect.tly+(x.extras.rect.height)/2;
                                        x.initscale=1;
                                        x.initrot=0;
                                        vObj.push_back(x);
                                        nActiveObjects++;
                                        mObj[x.idenNumber]=vObj.size()-1;
                                }
                                else
                                        error=1;
                                break;
                        case 14://ellipse       14        4
                                if(state==2)
                                {
                                        //add the obect...
                                        object x; int nump;
                                        x.type=14;
                                        x.toDisplay=1;
                                        fin>>x.idenNumber>>nump;
//cout<<"nump: "<<nump<<endl;
                                        x.params.resize(nump);
                                        for(int i=0;i<nump;i++)
                                                fin>>x.params[i];
//                                      if(fin.peek()!='\n')
//                                      {
//                                              cout<<"here: "<<fin.peek()<<endl;
                                                fin>>x.color.R>>x.color.G>>x.color.B;
//
        cout<<"COLOR:"<<x.color.R<<'\t'<<x.color.G<<'\t'<<x.color.B<<endl;;
//return 0;
//                                      }
//                                      else
//                                              x.color.R=x.color.G=x.color.B=0;

                                        x.initx=x.centerx =x.params[0];//first paramter is center x
                                        x.inity=x.centery=x.params[1];//second paramter is center y
                                        x.extras.radius.x =x.params[2]; //third paramter is radisousx
                                        x.extras.radius.y =x.params[3];//fourthparamter is radiusy
                                        x.initscale=1;
                                        x.initrot=0;

                                        vObj.push_back(x);
                                        nActiveObjects++;
                                        mObj[x.idenNumber]=vObj.size()-1;

                                }
                                else
                                        error=1;
                                break;
                        case 15://poly          15        N
                                if(state==2)
```

```
                {
                        //add the obect...
                        object x; int nump;
                        x.type=15;
                        x.toDisplay=1;
                        fin>>x.idenNumber>>nump;
                        x.params.resize(nump);
                        int i;
                        for(i=0;i<nump;i++)
                                fin>>x.params[i];
//                      if(fin.peek()!='\n')
                                fin>>x.color.R>>x.color.G>>x.color.B;
//                      else
//                              x.color.R=x.color.G=x.color.B=0;

                        //calculate center: Average
                        x.centerx=x.centery=0;
                        for(i=0;i<nump/2;i++){
                                x.centerx+=x.params[2*i];//even param locations are x
positions
                                x.centery+=x.params[2*i+1];//even param locations are x
positions
                        }
                        x.initx=x.centerx/=(nump/2);
                        x.inity=x.centery/=(nump/2);
                        x.initscale=1;
                        x.initrot=0;

                        vObj.push_back(x);
                        nActiveObjects++;
                        mObj[x.idenNumber]=vObj.size()-1;
                }
                else
                        error=1;
                break;
        case 19://group       19
                if(state==3)
                {
                        //add the obect...
                        object x; int nump;
                        x.type=19;
                        x.toDisplay=1;
                        fin>>x.idenNumber>>nump;
                        x.params.resize(nump);
                        int i;
                        for(i=0;i<nump;i++)
                                fin>>x.params[i];
                        fin>>x.initx>>x.inity;
//                      cout<<x.idenNumber<<' '<<x.initx<<' '<<x.inity<<' '<<nump<<endl;
                        //calculate center: Average
                        x.centerx=x.centery=0;
                        for(i=0;i<nump;i++){//average all the centerxs...
                                vObj[mObj[(int)x.params[i]]].toDisplay=0;  //don't display
items used in a group
                                nActiveObjects--;
```

```
                                        x.centerx+=vObj[mObj[(int)x.params[i]]].centerx ;//get each
objects centerx
                                        x.centery+=vObj[mObj[(int)x.params[i]]].centery; //get each
objects centery
                                }

                                x.centerx/=(nump);
                                x.centery/=(nump);
                                x.initscale=1;
                                x.initrot=0;

                                vObj.push_back(x);
                                nActiveObjects++;
                                mObj[x.idenNumber]=vObj.size()-1;
//                              cout<<"Me: "<<mObj[x.idenNumber]<<endl;
                        }
                        else
                                error=1;
                        break;
                case 21://translate           21       4
                        if(state==4)
                        {
                                //add the obect...
                                funcs x; int nump;
                                x.type =21;
                                fin>>x.idenNumber>>nump;

        fin>>x.details.newpos.x>>x.details.newpos.y>>x.startframe>>x.endframe;
/*                              if( vObj[mObj[x.idenNumber]].type==13)
                                {
                                        cout<<"corecting rectangle shift";

        x.details.newpos.x+=vObj[mObj[x.idenNumber]].extras.rect.width/2;

        x.details.newpos.y+=vObj[mObj[x.idenNumber]].extras.rect.height/2;
                                }
*/
                                vFn.push_back(x);
                        }
                        else
                                error=1;
                        break;
                case 22://rotate              22       3
                        if(state==4)
                        {
                                //add the obect...
                                funcs x; int nump;
                                x.type =22;
                                fin>>x.idenNumber>>nump;
                                fin>>x.details.rotateDegrees>>x.startframe>>x.endframe;//read in the 3
params
                                x.details.rotateDegrees*=-1;
                                vFn.push_back(x);
                        }
                        else
                                error=1;
```

```
                        break;
                case 23://scale             23      3
                        if(state==4)
                        {
                                //add the obect...
                                funcs x; int nump;
                                x.type =23;
                                fin>>x.idenNumber>>nump;
                                fin>>x.details.scaleSize>>x.startframe>>x.endframe;//read in the 3
params
                                vFn.push_back(x);
                        }
                        else
                                error=1;
                        break;
                case 30://Invisibility
                        if(state==5)
                        {
                                //add the obect...
                                visib x;
                                fin>>x.idenNumber;
                                fin>>x.startframe>>x.endframe;//read in the 2 other params
                                vVis.push_back(x);
                        }
                        else
                                error=1;
                        break;

                default:
                        cerr<<"ERROR parsing this number"<<rowType<<endl ;
                }


//                      cout<<"parsed this number: "<<rowType<<endl ;
                        if(error>0)
                                cerr<<"error number: "<<error<<"; state: "<<state<<endl;


        }
        //start playing!
        /*
        2)

in the mean time... do the ming code
*/

  vector<SWFShape *> vShapes;
        vector<SWFDisplayItem *> vDisp;
        map<int,int> mObj2Disp;//which position in the Object vecotr corresponds to which position in
the Display Vector
        Ming_init();

  SWFMovie *m = new SWFMovie();
//      cout<<"Ming_init();\nSWFMovie *m = new SWFMovie();\n";
        m->setFrames(theP.frames);
//      cout<<"m->setFrames("<<theP.frames<<");\n";
```

```
  m->setDimension(theP.width , theP.height);
//          cout<<"m->setDimension("<<theP.width<<" , "<<theP.height<<");\n";
            //put in the shapes
            //cout<<"numObjects: "<<nActiveObjects<<endl;
            int i;
            for(i=0;i<vObj.size();i++){
                        if(vObj[i].toDisplay==1){//if not part of group
                                    //cout<<"showing object "<<i<<endl;
                                    SWFShape *s = new SWFShape();
//                                  cout<<"SWFShape *s = new SWFShape();\n";
                                    vObj[i].getShape(s);
                                    vShapes.push_back(s);
                                    vObj[i].isVisible=1;
                                    SWFDisplayItem * tDI;
                                    tDI=m->add(vShapes[vShapes.size()-1]);
                                    tDI->moveTo(vObj[i].initx,vObj[i].inity);
                                    tDI->rotateTo(vObj[i].initrot);
                                    tDI->scaleTo(vObj[i].initscale);
                                    vDisp.push_back(tDI);
                                    mObj2Disp[i]=vDisp.size()-1;
                        }
                        //else
//                                  cout<<"skipping object "<<i<<endl;
            }
/*

--for each frame
---if !invisibile
----tempMatrix=0
----for each animation
-----update tempMatrix (based on animation, current frame #, and initial pos)
----add to frames vector using tempMatrix
*/

            //all teh objects are already in the array
            int j,k;
            for(i=0;i<theP.frames+1 ;i++){
                        //take care of invisibilites...
                        for(j=0;j<vVis.size();j++){
                                    if(vVis[j].startframe==i){
//                                              cout<<"removing "<<vVis[j].idenNumber<<'
'<<mObj[vVis[j].idenNumber]<<mObj2Disp[mObj[vVis[j].idenNumber]]<<" at frame "<<i<<endl;

                                              k=mObj[vVis[j].idenNumber];
                                              vObj[k].isVisible=0;
                                              vDisp[mObj2Disp[k]]->scaleTo((float).000001);
                                    }
                                    if(vVis[j].endframe==i){
                                              k=mObj[vVis[j].idenNumber];
//                                              cout<<"adding "<<vVis[j].idenNumber<<" at frame "<<i<<endl;
                                              vObj[k].isVisible=1;
                                              vDisp[mObj2Disp[k]]->scaleTo(1);
                                              //look for last scale, otherwise use 1
                                              int maxEf=0;//maximum endfram before hitting me
                                              for(int o=0;o<vFn.size();o++){
```

```
		if(vFn[o].type==23&&vFn[o].idenNumber==vVis[j].idenNumber&&vFn[o].endframe<=i&&vFn
[o].endframe>maxEf){
                                                maxEf=vFn[o].endframe;
                                                vDisp[mObj2Disp[k]]-
>scaleTo(vFn[o].details.scaleSize);
//                                              cout<<"Setting scale to:
"<<vFn[o].details.scaleSize<<endl;
                                        }
                                }
                        }
                }
                //take care of animatrions
                for(j=0;j<vFn.size();j++){
                        if(i==vFn[j].endframe){//set the new initial scale, pos, rot...
                                k=mObj[vFn[j].idenNumber];
                                switch(vFn[j].type){
                                case 21://translate
                                        vObj[k].initx=vFn[j].details.newpos.x;
                                        vObj[k].inity=vFn[j].details.newpos.y;
//                                      cout<<"setting pos: "<<j<<" "<<vFn[j].details.newpos.x<<"
"<<vFn[j].details.newpos.y<<endl;
                                        break;
                                case 22://Rptate
                                        vObj[k].initrot=vFn[j].details.rotateDegrees;
//                                      cout<<"setting ROT: "<<j<<"
"<<vFn[j].details.rotateDegrees<<endl;
                                        break;
                                case 23://Scale
                                        vObj[k].initscale=vFn[j].details.scaleSize;
//                                      cout<<"setting SCALE: "<<j<<" "<<vFn[j].details.scaleSize
<<endl;
                                        break;
                                }
                        }
                        if(i<vFn[j].endframe&&i>=vFn[j].startframe){
                                float newx, newy,newrot,newscale;
                                k=mObj[vFn[j].idenNumber];
                                if(i==vFn[j].startframe){
                                        switch(vFn[j].type){
                                        case 21://translate
//                                              cout<<"reaing init pos: "<<j<<" "<<vObj[k].initx<<"
"<<vObj[k].inity<<endl;
                                                break;
                                        case 22:
//                                              cout<<"reaing ROT: "<<j<<" "<<vObj[k].initrot
<<endl;
                                                break;
                                        case 23:
//                                              cout<<"reaing SCALE: "<<j<<" "<<vObj[k].initscale
<<endl;
                                                break;
                                        }
                                }
//                              cout<<"k: "<<k;
                                int deltaFrames=(vFn[j].endframe-vFn[j].startframe);
```

```
                                        float perc=(float)(i-vFn[j].startframe+1)/(float)deltaFrames;
                                        switch(vFn[j].type){
                                        case 21://translate
                                                newx=vObj[k].initx+perc*(vFn[j].details.newpos.x-
vObj[k].initx);
                                                newy=vObj[k].inity+perc*(vFn[j].details.newpos.y-
vObj[k].inity);
                                                vDisp[mObj2Disp[k]]->moveTo( newx,newy);
                                                vObj[k].centerx=newx;
                                                vObj[k].centery=newy;
                                                break;
                                        case 22://Rptate
                                                newrot=vObj[k].initrot+perc*(vFn[j].details.rotateDegrees-
vObj[k].initrot);
//                                              cout<<"to rot:"<<vFn[j].details.rotateDegrees<<"*"<<perc<<"=new
rot: "<<newrot<<endl;
                                                vDisp[mObj2Disp[k]]->rotateTo(newrot);
                                                break;
                                        case 23://Scale
                                                if(vObj[k].isVisible){
                                                        newscale=vObj[k].initscale
+perc*(vFn[j].details.scaleSize -vObj[k].initscale);
                //                              cout<<"to rot:"<<vFn[j].details.scaleSize
<<"*"<<perc<<"=new scale: "<<newscale<<endl;
                                                        vDisp[mObj2Disp[k]]->scaleTo(newscale);
                                                }
                                                break;
                                        }
                                }
                        }
                        m->nextFrame();
                }

                m->save("x.swf");

                return 0;
}

void object::getShape(SWFShape * s){
        int i;
        s->setLine(LINEW,this->color.R,this->color.G,this->color.B);
//      cout<<"s->setLine(LINEW,"<<this->color.R<<","<<this->color.G<<","<<this->color.B<<");\n";
        switch(this->type){         //point 11,line 12,rect 13,ellipse 14,poly 15
        case 11://point
                s->movePenTo(0 ,0);
                s->drawLine(1,0);
                break;
        case 12://line
                s->movePenTo((this->extras.line.x1-this->initx  ),(this->extras.line.y1-this->inity ));
                s->drawLineTo((this->extras.line.x2-this->initx ),(this->extras.line.y2-this->inity));
//              cout<<"Line Details "<<this->extras.line.x1<<' '<<this->extras.line.y1<<' '<<this-
>extras.line.x2<<' '<<this->extras.line.y2
//                              <<' '<<this->initx <<' '<<this->inity<<endl;

                break;
        case 13://rect
```

```
//                      cout<<"RECT Details "<<this->extras.rect.tlx<<' '<<this->extras.rect.tly<<' '<<this-
>extras.rect.width<<' '<<this->extras.rect.height
//                              <<' '<<this->initx <<' '<<this->inity<<endl;
                        //s->movePenTo(this->extras.rect.tlx,this->extras.rect.tly);
                        s->movePenTo(0,0);
                        s->movePen(-1*this->extras.rect.width/2,-1*this->extras.rect.height/2);
                        s->drawLine(this->extras.rect.width,0);
                        s->drawLine(0,this->extras.rect.height);
                        s->drawLine((this->extras.rect.width)*-1,0);
                        s->drawLine(0, (this->extras.rect.height)*-1);
                        break;
                case 14://ellipse
                        ellipse(s,0,0,this->extras.radius.x,this->extras.radius.y);
                        break;
                case 15://polygon
                        s->movePenTo(this->params[0]-this->initx,this->params[1]-this->inity);
                        for(i=1;i<this->params.size()/2;i++){
                                s->drawLineTo(this->params[2*i]-this->initx,this->params[2*i+1]-this->inity);
                        }
                        s->drawLineTo(this->params[0]-this->initx,this->params[1]-this->inity);
                        break;
                case 19:
                        //how to draw a group object???
                        for(i=0;i<this->params.size();i++){//average all the centerxs...
                                if(!(vObj[mObj[(int)this->params[i]]].color.R==255&&vObj[mObj[(int)this-
>params[i]]].color.G==255&&vObj[mObj[(int)this->params[i]]].color.B==255))
                                        vObj[mObj[(int)this->params[i]]].getShape(s,this->centerx,this-
>centery);
                        }
                        break;
                }
}
void object::getShape(SWFShape * s,float offsetx, float offsety){
        int i;
        s->setLine(LINEW,this->color.R,this->color.G,this->color.B);
        switch(this->type){             //point 11,line 12,rect 13,ellipse 14,poly 15
        case 11://point
                s->movePenTo(this->initx -offsetx ,this->inity -offsety);
                s->drawLine(1,0);
                break;
        case 12://line
                s->movePenTo((this->extras.line.x1-offsetx ),(this->extras.line.y1-offsetx ));
                s->drawLineTo((this->extras.line.x2-offsetx),(this->extras.line.y2-offsety));
                break;
        case 13://rect
                s->movePenTo(this->extras.rect.tlx-offsetx+this->extras.rect.width/2 ,this-
>extras.rect.tly-offsety+this->extras.rect.height/2);
                s->movePen(-1*this->extras.rect.width/2,-1*this->extras.rect.height/2);
                s->drawLine(this->extras.rect.width,0);
                s->drawLine(0,this->extras.rect.height);
                s->drawLine((this->extras.rect.width)*-1,0);
                s->drawLine(0, (this->extras.rect.height)*-1);
                break;
        case 14://ellipse
                ellipse(s,this->initx-offsetx ,this->inity-offsety,this->extras.radius.x,this-
>extras.radius.y);
```

```
                break;
        case 15://polygon
                s->movePenTo(this->params[0]-offsetx,this->params[1]-offsety);
                for(i=1;i<this->params.size()/2;i++){
                        s->drawLineTo(this->params[2*i]-offsetx,this->params[2*i+1]-offsety);
                }
                s->drawLineTo(this->params[0]-offsetx,this->params[1]-offsety);
                break;
        case 19:
                //how to draw a group object???

                for(i=0;i<this->params.size();i++){//average all the centerxs...
                        vObj[mObj[(int)this->params[i]]].getShape(s,this->centerx,this->centery);
                }
                break;
        }
}

void ellipse(SWFShape * shape, float cx, float cy, float rx, float ry){
//          cout<<"CX:"<<cx<<"CY:"<<cy<<"RX:"<<rx<<"RY:"<<ry<<endl;
  const float PI=(float)3.14159265358979323846264338327 95;
  float sa=0,ea=2*PI;
//  float cx=100,cy=100;
//          float rx=20,ry=50;

  float sweep = ea-sa;
  // determine number of segments, 8 at most
  int nSegs = (int)(1+floor(7*(sweep/(2*PI))));
// find angle of each segment
  float subangle = (sweep)/nSegs;
  float angle = sa;
    //begin arc:
    shape->movePenTo(cx + rx*cos(angle), cy+ ry*sin(angle));
//                      cout<<"shape->movePenTo("<<cx + rx*cos(angle)<<" , "<<cy+ ry*sin(angle)<<");\n";
  for(int i =0;i<nSegs;i++){
    angle += subangle/2;
    float controlx = cx + rx * cos(angle) / cos(subangle/2);
    float controly = cy + ry * sin(angle) / cos(subangle/2);
    angle += subangle/2;
    float anchorx = cx + rx * cos(angle);
    float anchory = cy + ry * sin(angle);
    shape->drawCurveTo(controlx, controly, anchorx, anchory);
//                      cout<<"shape->drawCurveTo("<<controlx<<","<<controly<<", "<<anchorx<<",
"<<anchory<<");\n";
  }

}
```