MDraw

Graphics Manipulation Language

Final Report

Jingyu Shi (js4151)

Huimin Sun (hs2740)

Dongxiang Yan (dy2224)

COMS 4115

Columbia University

August 16, 2013

1 Introduction

MDraw means “my draw”, which is designed as a 2D graphics drawing and manipulation language in the scope of this project due to time constraints.

Now days, although there are many popular graphics drawing and manipulation software solutions, there is no dedicated text-based language to facilitate and automate this process. Some repeating works are boring and time consuming. The case is even worse for complex graphics. Some graphics are hard to be repeated and scaled manually. By using MDraw, these issues could be resolved with ease. The purpose of this language is to provide a powerful solution with easy management and simple structure. It enables users to automatically draw and manipulate numerous graphics by taking advantage of this programming language.

2 Language Tutorial

MDraw is a simple C-like text-based language with powerful 2D graphics drawing functionality. Before checking the details of this project, the basic types, syntax and examples are shown as below to provide a quick glance.

2.1 Lexical Conventions

2.1.1 Basic Types

There are 4 built-in types in MDraw, they are point, line, curve and ellipse.

- Point p1;

p1 = Mpoint(x, y); /* 2 parameter as the coordinate of the point*/

- Line l1;

l1 = Mline(x1, y1, x2, y2); /* the coordinates of the 2 end point*/

- Curve c1;

c1 = Mcurve(x, y, w, h, angst, angExt); /*parameters as location, size, angular extents*/

- Ellipse e1;

e1 = Mellipse(x, y, w, h);

e1.draw(); p1.draw(); c1.draw();

There are also other basic types like int.

- Int i;

i = 10;

2.1.2 The Basic function

The basic function is the entry point of every mdraw program, and it is also where the basic functionalities are carried out.

Here is a simple example of the basic function:

```
void basic(){
  if(1<2){
      print “This is the start!”;
  }
}
```

All the user defined functions have the same syntax:

```
Built-in type function_name ( type arg1,type arg2 ){

#variable declarations

#function calls

}
```

2.1.3 The paint function

In the paint() function we call the function draw() which will draw the graphics on the pop up window.

2.1.4 Your first program

The following example shows a simple user defined function called pline, which takes in 2 arguments as the start and end point of a line. And in paint() function draw() is called to draw a line.

```
void basic(){
        for(int i; i<5; i++){
                print "Hello world!";
        }
}

Line pline (Point p1, Point p2){
int x1;
int y1;
int x2;
int y2;
x1 = p1.getX();
y1 = p1.getY();
x2 = p2.getX();
y2 = p2.getY();
Line l;
l = MLine(x1,y1,x2,y2);
return l;
}

void paint(){
Point p1;
p1 = MPoint(100,100);
Point p2;

p2  = MPoint(200,200);

Line l;

l = pline(p1, p2);

l.draw();

}
```

2.1.5 Installation and Compilation

To compile mdraw source code, make sure you have a 1.6+ Javac version and download the built-in Java_lib ;

To compile and run a mdraw file , sh ./run.sh name.mdraw

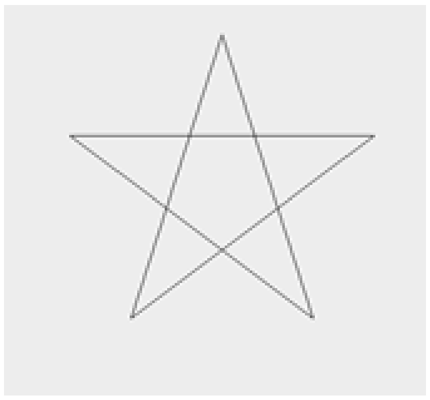
2.2 Examples

2.2.1 Star

This is a simple five-pointed star example. For given five points with coordinates, by connecting each pair points with straight lines, the star is built.

/* The source code starts here:

```
void basic(){}

void paint(){
Point p1;
Point p2;
Point p3;
Point p4;
Point p5;
p1 = MPoint(300,200);
p2 = MPoint(150,300);
p3 = MPoint(450,300);
p4 = MPoint(210,480);
p5 = MPoint(390,480);
Line l;
int x1;
int x2;
int x3;
int x4;
int x5;
int y1;
int y2;
int y3;
int y4;
int y5;
x1 = p1.getX();
y1 = p1.getY();
x2 = p2.getX();
y2 = p2.getY();
x3 = p3.getX();
y3 = p3.getY();
x4 = p4.getX();
y4 = p4.getY();
x5 = p5.getX();
y5 = p5.getY();
l = MLine(x1,y1,x4,y4);
l.draw();
l = MLine(x1,y1,x5,y5);
l.draw();
l = MLine(x2,y2,x3,y3);
l.draw();
l = MLine(x2,y2,x5,y5);
l.draw();
```



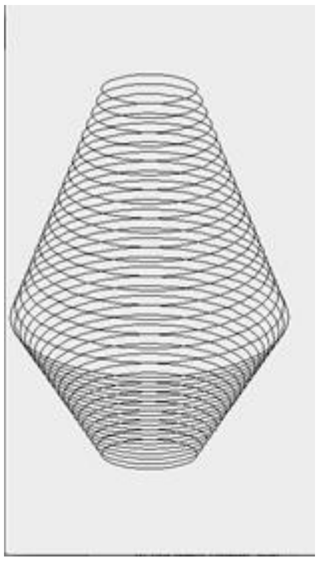
```
l = MLine(x3,y3,x4,y4);
l.draw();
}
```

The source code ends here. */

2.2.2 Pottery

This example takes advantage of defined type Eclipse to draw a pottery-like shape with many eclipses in just several lines of codes. The positions and shapes of eclipses are manipulated to form this pottery shape.

/* The source code starts here:



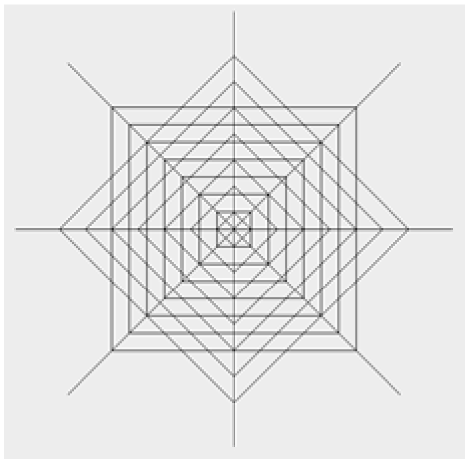
```
void basic(){}

void paint(){
Ellipse e1;
int i;
for(i=0; i<20;i=i+1){
e1 = MEllipse(100-5*i, 100+10*i, 100+10*i, 30+5*i);
e1.draw();
}
Ellipse e2;
for (i=0;i<20;i=i+1){
e2 = MEllipse(5+5*i, 290+i*10, 290-10*i,125-5*i);
e2.draw();
}
}
```

The source code ends here. */

2.2.3 Web

In this example, by manipulating the coordinates of points and drawing lines between pairs of points, a complex web is formed in an easy way.

/* The source code starts here:



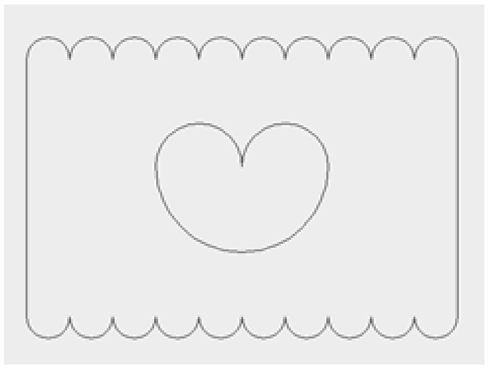
```
void basic(){}

void paint(){
Line l;
int i;
for(i=0;i<7;i=i+1){
int x1 = 160+20*i;
int x2 = 440-20*i;
l = MLine(x1, x1, x2, x1);
l.draw();
l = MLine(x1,x1,x1,x2);
l.draw();
l = MLine(x1,x2,x2,x2);
l.draw();
l = MLine(x2,x1,x2,x2);
l.draw();
l = MLine(300,100+30*i,100+30*i,300);
l.draw();
l = MLine(100+30*i,300,300,500-30*i);
l.draw();
l = MLine(300,500-30*i,500-30*i,300);
l.draw();
l = MLine(500-30*i,300,300,100+30*i);
l.draw();
}

Ellipse e;
e = MEllipse(100,100,400,400);
l.draw();

l=MLine(110,110,490,490);
l.draw();
l=MLine(490,110,110,490);
l.draw();
l=MLine(300,50,300,550);
l.draw();
l=MLine(50,300,550,300);
l.draw();
}
```

The source code ends here. */

2.2.4 Heart

In this example, a heart shape with curve frame is drawn.

/* The source code starts here:

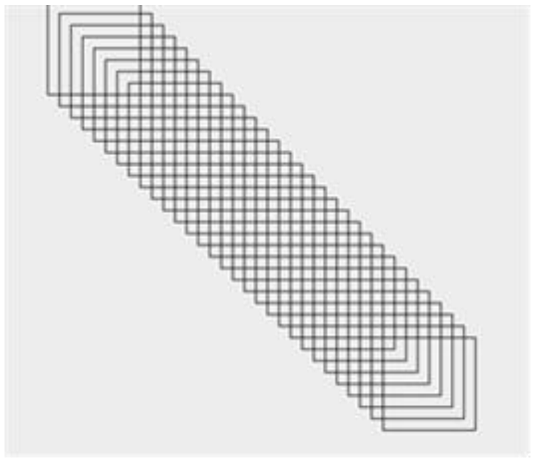


```
void basic(){}

void paint(){
Curve c;
int i;
for(i=0;i<2;i=i+1){
c = MCurve(200+100*i, 200, 100, 100, 0, 180);
c.draw();
}

for(i=0;i<10;i=i+1){
c = MCurve(50+50*i,100,50,50,0,180);
c.draw();
}

for(i=0;i<10;i=i+1){

c = MCurve(50+50*i,400,50,50,180,180);
c.draw();
}

c = MCurve(200,150,200,200,180,180);
c.draw();

Line l;
l = MLine(50,125,50,425);
l.draw();
l = MLine(550,125,550,425);
l.draw();
}
The source code ends here. */
```

2.2.5 Discrete Rectangles

In this example, a box shape is formed by drawing discrete rectangles.

```
/* The source code starts here:

void basic(){}

void paint(){
int i;
for(i=100;i<400;i=i+10){
int sx;
sx = i;
int sy;
sy = i;
int j;
for (j=0;j<2;j=j+1){
Line l1;
l1 = MLine(sx,sy,sx+80,sy);
l1.draw();
sy = sy -80;
}
int px;
int py;
px = i;
py = i;
for(j=0;j<2;j=j+1){
Line l2;
l2 = MLine(px,py,px,py-80);
l2.draw();
px = px+80;
}
}
}
The source code ends here. */
```



3 Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

In MDraw, comments are represented inside of parentheses and asterisks. Comments usually start with /* and end with */.

3.1.2 Identifiers

Identifiers are sequences of letters, digits and underscores (_). Uppercase letters and lowercase letters are considered the same in MDraw because it is not case sensitive. The first character of an identifier must be a letter.

3.1.3 Keywords

The following keywords are reserved for MDraw:

int	Point	Line	Ellipse
Curve	print	string	void
boolean	.draw()	MEllipse	MLine
MPoint	MCurve	.getX()	.getY()
if	else	for	while
return			

3.1.4 Punctuation

The following symbols are used to organize code:

symbol	Meaning
;	Marks the end of a statement
,	Separate argument in functions
.	Dot is the prefix of functions like getX, getY, draw, etc
( )	Used for grouping parameters

3.1.5 Constants

Constants in MDraw include integer, float, string and boolean.

3.1.5.1 Integer constants

An integer constant is a sequence of digits without a decimal point.

```
/* Here is an example of integer constant. */
int a = 8;
```

3.1.5.2 String constants

A string constant is a sequence of characters surrounded by double quotes.

```
/* Here is an example of string constant. */
string a = "This is a string constant."
```

3.1.5.3 Boolean constants

A boolean constant represents true or false.

```
/* Here is an example of Boolean constants. */
 boolean a = True;
 boolean b = False;
```

3.2 Types

3.2.1 General Types

The general types include integers, strings and booleans.

3.2.1.1 Integers

An integer is defined as a sequence of digits without a decimal point.

3.2.1.2 Strings

A string is defined as a sequence of characters surrounded by double quotes.

3.2.1.3 Booleans

A boolean is defined as a "True" or "False" value.

3.2.2 Object Types

In MDraw, object types include Point, Line, Curve and Ellipse.

3.2.2.1 Point

Point is consist of a pair of integer Cartesian coordinates.

3.2.2.2 Line

Line is consist of points with start point and end point.

3.2.2.3 Curve

Curve is consist of four control points: start point, two intermediate points and end points. It is defined as Bernstein-Bezier curve.

3.2.2.4 Ellipse

Ellipse is consist of a center point, height and width.

3.3 Variables

3.3.1 Local Variables

Local variables are declared inside a function or a block. Its scope is within the function or the block. They are not reachable from outside of the function or block.

3.3.2 Global Variables

Global variables are typically declared at the top of the main program. They are not declared in any subset function or block. They are reachable by any functions in the same program. Its lifetime is the same as the lifetime of the program.

3.4 Expressions

3.4.1 Primary expressions

Primary expressions group left to right.

3.4.1.1 Identifier

An identifier is a primary expression. Its type is specified by its declaration. It follows the rules in Section 3.1.2.

3.4.1.2 Constant

A constant is a primary expression. It follows the rules Section 3.1.4.

3.4.1.3 String

A string is a primary expression. It is defined as a sequence of characters surrounded by double quotes.

3.4.1.4 (expression)

A parenthesized expression is a primary expression, which gives the expression high priority in calculation.

3.4.1.5 [expression]

The expression in square brackets is a primary expression. It indicates index into a list.

3.4.1.6 (expression-list)

The expression list consists of 0 or more expressions, which are the arguments, separated by comma.

```
/* Here is an example of expression-list. */
MLine (point_01, point_02);
```

3.4.2 Unary operators

Expressions with unary operators group left to right.

3.4.2.1 – expression

- expression gives the negative of the expression and has the same type. It is only valid for integers.

3.4.2.2 ! expression

! expression gives the logical negation of the expression, which must be boolean type. ! expression returns 1if the value of the expression is 0, 0 if the value is not zero.

3.4.3 Multiplicative operators

*3.4.3.1 expression * expression*
Multiplication is valid between integers and Points, but the two expressions cannot be Points at the same time. If both expressions are integers, the result is an integer. If one of the expression is a Point, like (a, b), each of a and b is multiplied by the other expression and it returns a Point.

3.4.3.2 expression / expression
Division is similar to multiplication. The only difference is that the second expression cannot have a value of zero.

3.4.3.3 expression % expression
Mod gives the remainder of expression / expression. Both expressions must be an integer.

3.4.4 Additive operators
The additive operators + and – group left to right.

3.4.5 Relational operators

3.4.5.1 expression + expression
Addition gives the sum of the two expressions. It is valid between integers. It is also valid between two Points. While adding two Points, like (a, b) and (c, d), the result is (a+c, b+d), which is also a Point.

3.4.5.2 expression – expression
Subtraction is similar to addition.

3.4.6 Equality operators
The equality operators group left to right. They are valid among integers, floats. They are also valid between two Points. It returns a Boolean type.

3.4.6.1 expression == expression
Equal to. Two points (a, b) and (c, d) are equal to each other if and only if a = c and b = d.

3.4.6.2 expression != expression
Not equal to. It is similar to equal.

3.4.7 Relational operators
The relational operators group left to right. They are valid between integers and floats. They are also valid between two Points (a, b) and (c, d). It returns a Boolean type.

3.4.7.1 expression < expression
Less than. While comparing two Points, it returns 1 if a<c and b<d, otherwise returns 0.

3.4.7.2 expression > expression
Great than. It is similar to less than.

3.4.7.3 expression <= expression
Less than or equal. It is similar to less than.

3.4.7.4 expression >= expression
Great than or equal. It is similar to less than.

3.4.8 Logical operators
Logical operators group left to right. They are valid between two Boolean expressions and return a Boolean type.

3.4.8.1 expression & expression
AND. If the first expression is zero, the second expression still needs to be evaluated.

3.4.8.2 expression | expression
OR. If the first expression is not zero, the second expression still needs to be evaluated.

3.4.9 Assignment operators
Assignment operators group right to left. They are valid between two expressions that have the same type.

3.4.9.1 expression = expression
It gives the value of the second expression to the first expression.

3.5 Declarations

3.5.1 Variable declaration
Variable declaration includes the type and the value. The value should be consistent with the defined type. For example,

```
/* declare a point */
Point p1;
p1 = MPoint(300, 200);

/* declare a integer */
Int x1;
```

3.5.2 Function declaration
Function declaration includes function key word and optional expressions as arguments. User-defined functions are supported in MDraw. It consist of type declaration and implementation. For example,

```
/* declare a function drawLine() */
Line drawLine(int x1, int y1, int x2, int y2) {
        return MLine(x1-1, y1-2, x2-3, y2-4);
}
```

3.6 Statements

3.6.1 Expression statement

Most of expression statements are assignments or function calls, which are all ended with semicolon “;”.

3.6.2 Conditional statement

3.6.2.1 if (expression) statement

The expression will be evaluated first. If it is true, then the following statement will be executed. Otherwise, the statement will be ignored.

3.6.2.2 if (expression) statement else statement

The expression will be evaluated first. If it is true, then the first statement will be executed. Otherwise, the second statement will be executed.

3.6.2.3 while (expression) statement

The expression will be evaluated before each execution of statement. This is a loop, which will repeatedly execute statement when expression is evaluated as true.

3.6.2.4 for (expression; expression; expression) statement

The first expression is used to initialize this loop. The second expression is evaluated as true or false. If it is true, the statement will be executed. Otherwise, this loop will be ended. The third expression is used to change the value in second expression after each execution of the statement.

3.6.3 Return Statement

For each function or method without void keyword, return statement should be included.

3.7 Functions

3.7.1 Print Function

The print function is used to display the results of the expression as standard output. The standard format is shown as below:

```
print (expression);
```

3.7.2 draw () Function

The draw() function is used to display the graphics in standard output window. It could be called by the following object types: Point, Line, Curve and Ellipse. For example,

```
Line s_01;
s_01 = MLine(x1, y1, x2, y2);
s_01.draw();
```

3.7.3 MPoint Function

This will build a point with 2 parameters as the coordinate of a point.

```
Point p;
p = MPoint(30, 30);
```

3.7.4 MLine Function

This will build a line with 4 parameters which is the coordinate of the start and end point of a line:

```
Line l;
l = MLine(10,10,30,30);
```

3.7.5 MCurve Function

This will build a curve with 6 parameters which is the location of the upper left corner and the height and width of the ellipse and also the start and end angle of the arc;

```
Curve c;
C = MCurve(200,150,200,200,180,180);
```

3.7.6 MEllipse Function

This will build a ellipse with 4 parameters, like in curve, the first 2 arguments are the coordinates of the upper left corner and the height and the width of the ellipse.

```
Ellipse e;
e = MEllipse(200,150,200,200);
```

3.7.7 Point.getX () Function

Point.getX() gives the X coordinate of the point.

```
Point p;
p = MPoint(10, 10);
int x;
x = p.getX();
```

3.7.8 Point.getY () Function

Point.getY() gives the Y coordinate of the point.

```
Point p;
p = MPoint(10, 10);
int y;
y = p.getY();
```

4 Project Plan

4.1 Project Process

The summer semester is very short. There are only 6 weeks in total, from the very beginning to the final due date. It is never be an easy job to manage and complete the project in such a short period.

After we formed the team in the first class, we began to set up group meeting schedule and discuss our project progress with TA on every week's TA office hour. In this project we fulfilled the Scrum project management strategy.

First, we setup group meeting right after each class. We also meet with TA one time per week at TA office hour. Besides this, we setup online instant messaging discussion group to extend our collaboration. During group meeting, each team member need to prepare to show the answers of the following three questions:

- What have you done since last meeting?
- What do you plan to do before next meeting?
- Any issues or blocks?

This step makes sure that we have good strategy to discuss and solve issues we meet in the development.

Second, we discuss and set up key milestones. This step makes sure that we have clear goals for each sprint. The sprint is the basic unit of development in Scrum. We define the lifetime of sprint as one week according to our milestones. The milestones are listed as below:

- Project Proposal
- LRM
- AST design & implementation
- Scanner design & implementation & test
- Parser design & implementation & test
- Semantic check design & implementation
- Compile design & implementation
- Translate design & implementation & test
- Overall test design & implementation
- Presentation and final project report

Our team did extensive research on choosing project idea. After discussion with TA, we decided to fulfill the idea of MDraw. At the beginning, we plan to transfer our language to bytecode by using OCaml. After discussion, we got two reasons to choose Java as compilation language. First, OCaml has higher learning curve than Java. Our team members are pretty newbies for OCaml. The summer semester is very short. There is no guarantee that we could fulfill our project by using QCaml as compilation language. Second, since this is PLT course, what we need to learn is to focus on language design and implementation. We'd like to save some time on leaning OCaml and spend it on this way.

We have extensive discussion on building LRM. We tried to build our language as simple as possible. At the same time, we hope our language statements are also clear and readable. Of course, we also need to consider the functionality of the language. Our goal is to create a powerful text-based language with simple and easy read syntax. During our group meeting, we tried to seek common points while reserving personal preference. Once LRM is done, we moved to the next development sprint quickly.

AST, scanner and parser design & implementation is the first development sprint. The strategy we used to build program little by little, from simple to complex. For each development sprint, we did test to make sure every additional program block is working. According to Scrum strategy, once we have workable AST, scanner and parser, we continue to do semantic check, build SAST, compile and translate as well. We spend most of the time on development sprint. For each sprint, we tried to build more complex sprint than the previous one.

Overall, with the great passion and team work spirit, we build a nice project

4.2 Project Timeline

Timeline	Progress
Week 1	• Form a team • Set up team management and project strategy • Project proposal
Week 2	• LRM • Sprint 1 (build AST, scanner and parser) • Sprint 2 (test of scanner and parser)
Week 3	• Sprint 3 (build SAST) • Sprint 4 (semantic check design & implementation)
Week 4	• Sprint 5 (compiler, translate design & implementation) • Sprint 6 (test of compiler)
Week 5	• Sprint 7 (test of translate) • Sprint 8 (overall test implementation)
Week 6	• Sprint 9 (examples and test cases implementation) • Project presentation and final report

4.3 Roles and Responsibilities

We are a great team. With the good team management and project plan, we adopt a collaborative approach. Each team member has actively made contributions in all aspects of this project.

We all participated in building proposal, LRM, presentation slides, final report, debugging and fixing issues raised during the development. Besides these, some specific roles are listed below:

Jingyu Shi	• Developed utility functions to generate strings for parser, developed generation of AST, parser, helped with semantic checking, translator, and the execution file MDraw.
Huimin Sun	• SAST, compile, translate file and their test file • Shell file to run the whole program • Some MDraw examples
Dongxiang Yan	• Team management & project plan • Developed generation of AST, parser, scanner and compile • Helped with development of translate, test and examples

4.4 Software Development Environment

OS	Mac OS X
Primary development language	OCaml 4.0
Lexical analysis	OCamllex
Parsing	OCamlyacc
Version control system	Git & GitHub
Scripts	Makefile and sh

4.5 Project Log

The Git/GitHub log is shown as below:

```
commit 6865c766a483608300b581f7e76210631c0672e4
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Fri Aug 16 23:35:32 2013 -0400

    release_3.0

commit 4b335da7d7a8110cc36a3bbe3136164bda25359a
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Fri Aug 16 14:26:54 2013 -0400

    updating report

commit 50606867ef7b4a8dec8eedd7faa2350cf2960c32
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 15 19:46:46 2013 -0400

    release_2.0

commit d3c0292fb9ef161de3b86bec6f294587c860b923
```

```
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:26:14 2013 -0400

    Update translate_test.ml

commit ed4225bf142f4b7efd3b22f97a8bfbd18679fbf2
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:25:59 2013 -0400

    Update translate.ml

commit 86b4506217acdd859e7d614c86a4c43601bce8d6
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:25:28 2013 -0400

    Update sast.mli

commit 3a8e8e2f868ba949b8499ecc38d1c2af8f43d20a
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:25:04 2013 -0400

    Update parser_test.ml

commit 89ca288894285d4422d2de16e94a5f43be6e4eef
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:24:34 2013 -0400

    Update parser.mly

commit 06f5dad6f3f8bf64a50c5f0558ee6d0b24ca240e
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:24:13 2013 -0400

    Update compile.ml

commit 5fda425bbcaaa876e00e89d67103626cbc7ca835
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:23:03 2013 -0400

    Update MDraw.ml

commit 2ca59d2926e8ac059b35e0520c4879bfaad82b9f
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 22:22:34 2013 -0400

    Update Makefile

commit dfbc098a8012f827c3dbc9fcf92caa28e0a18431
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 18:43:06 2013 -0400
```

```
    Update parser_test.ml

commit 95f26614fef9837a4d867333b54ca461e3f6f9fd
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 18:39:46 2013 -0400

    Update scanner_test.ml

commit 4cc3678841a4303101e65187eed2169e59047ec8
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 18:36:35 2013 -0400

    Rename ast.ml to ast.mli

commit fa71c19567aceb15bb4e5a8bd7403637c508bb05
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 18:34:58 2013 -0400

    Update ast.ml

commit 831f12beb022efecf02df1cca98bd9ec216d963f
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Aug 12 18:31:34 2013 -0400

    update

commit a0ddf9d2747e5069243a03078cc9afca36b747c3
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Aug 12 11:35:24 2013 -0400

    Final report

commit c07cb2c2d30f975e8e65cc5a20f86bd50363f9bf
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Aug 12 10:56:59 2013 -0400

    updating

commit 2ac384684aa2ccb3ca6b05bb365f7ad135201009
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 10:09:04 2013 -0400

    Update compile.ml

commit d93c3d64fd71ac180e12f93e63b944864e1583a9
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 10:08:57 2013 -0400

    Update ast.ml
```

```
commit 9b580e2d0c656136c906b5ed063a8c54c528f788
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 10:08:46 2013 -0400

    Update MDraw.ml

commit 12b0990bd3e6a4489637c7fde21a381a894232d0
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Mon Aug 12 10:05:16 2013 -0400

    Update parser.mly

commit 61714b7101ba17b63623fc80b0ec20d6d3380481
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 14:08:24 2013 -0400

    Create translate_test.ml

commit 8c607328f995047e7022922742f731000ad0da6d
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 14:02:10 2013 -0400

    Update parser_test.ml

commit 3a13973d441ebbc79c62bf4b64fbd3c2c42ea4e1
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:52:46 2013 -0400

    Create scanner_test.ml

commit 9535bffe49c25780fb709d88596ebf96e4bce4ed
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:47:36 2013 -0400

    Update parser_test.ml

commit b34b6d3d7ed732c62b1adc231094a2da1bd8f2f3
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:38:41 2013 -0400

    Update ast.ml

commit 22f4d73d920accb6cbd15711a0c027619cfe276a
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:38:21 2013 -0400

    Update parser.mly

commit 5184a6d5d81793b2e0ac570a23a0f12072746405
```

```
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:32:56 2013 -0400

    Update parser.mly

commit 3819de6774186bee9c133ac10d3b6a58a16708ca
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 13:00:08 2013 -0400

    Update MDraw.ml

commit 1f5ed5e932c69c62cb0d07b5ea5d0ab1e4852c1a
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:49:09 2013 -0400

    Update parser.mly

commit e0b05e3fe0d207847171c687aa680df679e7744e
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:44:46 2013 -0400

    Update ast.ml

commit 69a7842205f147bb3a90331dc7458d78a91a366b
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:42:47 2013 -0400

    upload sast.mli

commit 2d342b432dd6d0b6a3525126a43b04564feec3b6
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:36:51 2013 -0400

    Update translate.ml

commit 949644542349a7bdfbf77ed38fd977d2a6676dd7
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:19:51 2013 -0400

    Update scanner.mll

commit c0460740bf37fe9c6d614e46c596b2c1f59055cb
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:19:01 2013 -0400

    Update parser.mly

commit 3cf97ba1f677a9af8c9591fae81299143349e6a8
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 12:18:41 2013 -0400
```

```
    Update ast.ml

commit 55b9b7af5216eee3f12d156c251f99d776fe353c
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 11:00:36 2013 -0400

    Update translate.ml

commit 8a47b8cbe55aa14ef2a69c779244235204bc4a43
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 09:34:30 2013 -0400

    Update compile.ml

commit b9074f0566f32743d52565152415ebf967464468
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 09:29:46 2013 -0400

    Update parser.mly

commit a64008e3ff7f3b1601d05559d3edde18a9707fd2
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sun Aug 11 09:26:43 2013 -0400

    Update ast.ml

commit e13d13e1d2561c134cb8eede608ebdce62e882d4
Author: NileKid <jingyu.cindy@gmail.com>
Date:   Sun Aug 11 00:22:00 2013 -0400

    update

commit c7ca5cc411519e1c14c06d862535466884ae32d5
Author: NileKid <jingyu.cindy@gmail.com>
Date:   Sun Aug 11 00:09:04 2013 -0400

    comments

commit 623d61f05887af796357dd3f1e6639c036afa7ae
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sat Aug 10 22:34:30 2013 -0400

    Update translate.ml

commit fbfde89adcba98f695d4ad18f9a479f5f8f8d002
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sat Aug 10 22:28:14 2013 -0400

    Update parser.mly
```

```
commit 73bc8b1f2c24ed546344b20b35013287595704d5
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sat Aug 10 22:19:32 2013 -0400

    Update parser.mly

commit 06d818f83137d5adb3f9db4dd20c0e8c9aa6c580
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sat Aug 10 21:55:34 2013 -0400

    Update ast.ml

commit 9dd0549cf8d4921bf7f5fc8191fcc93b79b6e0ab
Merge: 01203eb d0a5d2a
Author: NileKid <jingyu.cindy@gmail.com>
Date:   Sat Aug 10 21:51:28 2013 -0400

    Merge branch 'master' of https://github.com/JackYan18/MDraw

commit 01203eb00e708f08d16d84e5176a43a11fbad539
Author: NileKid <jingyu.cindy@gmail.com>
Date:   Sat Aug 10 21:44:46 2013 -0400

    main run file

commit d0a5d2ae4c84b09a7f62177d348b5c70b7da3dc1
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Sat Aug 10 21:33:02 2013 -0400

    Update translate.ml

commit 29353492ac9ce7803c448cd8f22ea8f0aafcab8e
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Sat Aug 10 11:47:25 2013 -0400

    Delete bytecode.ml

commit 17d45d9c92777c95e0b0851184d67149da99c3d7
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Sat Aug 10 11:45:41 2013 -0400

    Delete execute.ml

commit 04929ffbad4c6a408bffc27e4d9da7007fb2f812
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 21:08:55 2013 -0400

    Update compile.ml
```

```
commit 15ab15bf02a547dc239dbd7d67f8acfca45cb915
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 20:51:35 2013 -0400

    Update compile.ml

commit bc274ae509f23782f3ec944c25d5ff835c21cd44
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 20:34:14 2013 -0400

    Update parser.mly

commit d0ee97acca7f914bc033fe55ad89345b71775de1
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 20:33:17 2013 -0400

    Update ast.ml

commit eb65eb8b728cb8f01488f58ecf86b69d671b5a9a
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 13:05:08 2013 -0400

    Update parser.mly

commit 0b631865bf253fb55cb34558e1fe985642246f60
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 13:04:55 2013 -0400

    Update ast.ml

commit 59fc16c3508e54dbbe283cd292d66537216c20e3
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 12:54:13 2013 -0400

    Update ast.ml

commit 1b42990da86e7fc036d43f72e1dfe6aa1d37a113
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 11:50:07 2013 -0400

    Update parser.mly

commit 56d4a4af056d6d0a026a2e308b3aed3ff66f0a19
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Fri Aug 9 09:23:14 2013 -0400

    Update compile.ml

commit f8510ffc85f85c0776bd23c38ac8839d5b71cf61
Author: hmsun <sunhuimin1990@gmail.com>
```

```
Date:   Thu Aug 8 20:54:28 2013 -0400

    Update ast.ml

commit 5c0d261b66be732559a0ce76758aea0fe005e3b4
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 20:34:23 2013 -0400

    Update ast.ml

commit 30ad94c32e926b2a5d9118b4f85911f7a9ff4abb
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 19:42:37 2013 -0400

    Update translate.ml

commit bfbfb118457e8b473f27dabe51267c9d2c9ec1a8
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 19:21:15 2013 -0400

    Update translate.ml

commit deb63ef74ea7a6d75f4e5b2c91635642837f8b57
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 18:39:23 2013 -0400

    Update parser.mly

commit ccb5f2744b86875441ec9ee13928b0b1c4f86bda
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 18:36:26 2013 -0400

    Update scanner.mll

commit 4a8f8b9bf0aa21c11e7c61cfadc1ae45288bc94b
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 18:32:03 2013 -0400

    Update ast.ml

commit bec2f4c00e96d8aa61c1ea2ec0a9a3e49ab58322
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 18:31:36 2013 -0400

    Update ast.ml

commit 2f0afe74ab0cd2750082b079ab946560acdc7d45
Merge: b70eca8 fbcf1a0
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 17:59:49 2013 -0400
```

```
    Merge branch 'master' of github.com:JackYan18/MDraw

commit b70eca853f71d0445d942c0266c01d051f719636
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 17:59:40 2013 -0400

    typo

commit fbcf1a01fafea1738b97d2fceed4568e1b65d1be
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 17:53:25 2013 -0400

    Update compile.ml

commit 170d51924c1d494094a4a62d777b714df6fbfc86
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 17:47:01 2013 -0400

    Update compile.ml

commit fe7fe481bfa82e999e9c04b8732095b905288980
Merge: 7314a79 14a8cbe
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 17:44:38 2013 -0400

    Merge branch 'master' of github.com:JackYan18/MDraw

commit 7314a79d389dff116f6275e07d1336d0e74d5dd0
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 17:44:26 2013 -0400

    release_1.2

commit 14a8cbed58c49338f713c7030efd6128eaefd617
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 17:42:34 2013 -0400

    Update ast.ml

commit 0aa9bb22ff57b1191ecc0194146c95eb77d79736
Author: NileKid <Jingyu.cindy@gmail.com>
Date:   Thu Aug 8 17:40:16 2013 -0400

    semantic check

commit 2a286eaf7cb0efb11425b22cdea3246bc009b4a5
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 15:20:07 2013 -0400
```

```
    Update ast.ml

commit ba1a5f3e937a5f993324006913942a1fe37576c4
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 14:21:50 2013 -0400

    Update compile.ml

commit b0c5481015b6c67daa197b92ba8210f3a945b93e
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 14:11:00 2013 -0400

    Update ast.ml

commit af803848c6ac2babef5cff8899608895a71ddc3b
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 13:42:35 2013 -0400

    Update ast.ml

commit 8a9caa4394968848457b70960ef068a104a1cdd6
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 13:40:27 2013 -0400

    Update scanner.mll

commit 5770df9246f0b7acd24be4c183b982dcf4c2f934
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 13:28:00 2013 -0400

    Update parser.mly

commit c9ea43477f8e37f959e3454bf03d324f77a32069
Author: hmsun <sunhuimin1990@gmail.com>
Date:   Thu Aug 8 12:19:41 2013 -0400

    Update parser.mly

commit fc3b19249e92cf536cb8fae47d537731e94a4505
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 11:08:58 2013 -0400

    change files locations

commit 152d353d2bb0598af18d1b410551286015aa7b21
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Thu Aug 8 10:58:45 2013 -0400

    release_1.0 updating project files organization
```

```
commit 36c3a3f1f030100fcd82e9423c6374188e889dac
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Wed Aug 7 17:53:58 2013 -0400

    file relations

commit 517544716af73bc8df8ad115bd7ae6d8956d0297
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 16:23:00 2013 -0400

    compile.ml

commit 384769f9cd498ea39892bfb085edaaf5fd0da93d
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 14:18:47 2013 -0400

    parser.mly

commit d713b46d3e4ccf540a79ae161d973ea65860443f
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 14:17:01 2013 -0400

    interpret.ml

commit 72605161c2c9224b21a4e34a77c0944905dbf799
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 11:32:57 2013 -0400

    updated parser

commit 17e66a9196c5d3d42b5d85ddc15b61554fb9ff6c
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 11:24:08 2013 -0400

    ast line updates

commit e8bd600d0375f6c0a30ee1f8ea630c0996070dca
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Sun Aug 4 10:54:07 2013 -0400

    updated scanner.mll

commit 92e6247eca470e09e77d500585db529c2f9aecb4
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Wed Jul 31 18:33:38 2013 -0400

    add CURVE

commit 4c6c4db0e408b45d9015de6f3987fb8b929c7908
Author: Jack Yan <jack.yan@arvatosystems.com>
```

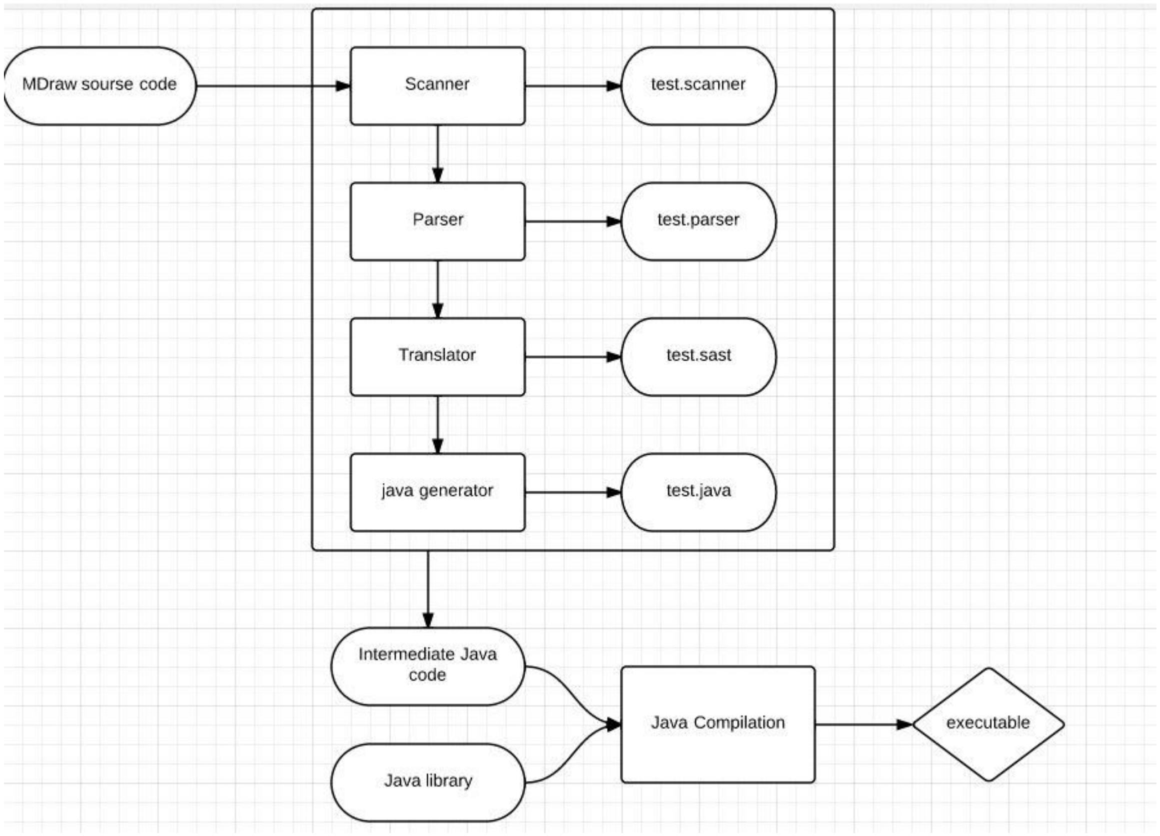
```
Date:   Wed Jul 31 15:24:35 2013 -0400

    Makefile

commit 056dafbaf8aa3c205f2032e16bd6c1910da2a7b4
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:25:20 2013 -0400

    MDraw.ml

commit b2da3a9b651f60be77952dbbded4607901016dcb
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:23:06 2013 -0400

    execute.ml

commit 96ef4a86898cfa80a3cceeb2ee46e9c2b64ebfd6
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:15:28 2013 -0400

    compile.ml

commit 5fc4973ed370462d8ad6821717b3168149bb3ddd
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:14:10 2013 -0400

    interpret.ml

commit df492a64e3509e986f4674020b08d5ff4975ba12
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:12:00 2013 -0400

    ast.ml

commit f274a62cdfeefa0e068464b907fab3b0dd932487
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:08:26 2013 -0400

    parser.mly

commit e967eae6cac38ed1250a62720d14d96a21213d5f
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 18:04:35 2013 -0400

    bytecode.ml

commit 26f019dd5cee6f2b5a4a123095b1093c8eb5d644
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Mon Jul 29 12:26:33 2013 -0400
```

```
    scanner.mll

commit b33f86df010f06d3d05632c443216f875935d385
Merge: 1216723 449d652
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Wed Jul 17 20:14:20 2013 -0400

    Merge branch 'master' of github.com:JackYan18/MDraw

commit 12167236465efb0ae1595cfb072b41a4a2f51c37
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Wed Jul 17 20:13:53 2013 -0400

    LRM draft

commit 449d65220893adf945500b352f95d4de54f4deda
Author: jack yan <dongxiang.yan@arvatosystems.com>
Date:   Wed Jul 10 19:52:14 2013 -0400

    Update README.md

commit 22c5d03daf91f03b045bfc1a186e2ebb6bb3daf6
Author: Jack Yan <jack.yan@arvatosystems.com>
Date:   Wed Jul 10 19:48:44 2013 -0400

    Original project proposal

commit a3e3785ee0a77738a0c1ec69e2029359d424fc8d
Author: jack yan <dongxiang.yan@arvatosystems.com>
Date:   Wed Jul 10 16:40:39 2013 -0700

    Initial commit
(END)
```

5 Architectural Design

5.1 Overview

The overall compiler design of MDraw is shown in the following architecture figure. The main components include scanner, parser, translator and java generator. Java is used as compilation language. This means the general process is to translate MDraw language source codes to standard Java codes. After Java compilation, the program is executable.



5.2 Scanner

The file with suffix .mdraw is recognized as MDraw source file. The scanner is used to tokenize the source code file by using lexical analysis. Comments, whitespace and other unnecessary characters will be striped. If there are any illegal characters, the program will fail. The file scanner.mll is built as scanner.

5.3 Parser

The scanned tokens from scanner are output to parser. By inputting these tokens and basing on predefined syntax rules, parser will construct an AST (abstract syntax tree). The correspondent file is parser.mly.

5.4 Translator

According to AST, the translator is designed to do semantic check and generate SAST (semantically-checked abstract syntax tree). In MDraw, the semantic check evaluates the type of a statement. If there are any errors, exceptions will be thrown. The correspondent file is translate.ml.

5.5 Java Generator

The Java generator is used to transfer a program in AST into a Java program. If there are any errors, exception will be thrown, e.g., a reference to an unknown variable or function.

6 Test Plan

6.1 Test Suite

For each feature described in in the LRM, we provided a test. The comprehensive listing of our tests includes scanner test, ast test, sast test and overall test.

6.1.1 Scanner Test

This test converts the scanner (tokens) into string form.

```
open Scanner
open Parser

let token_to_string token =
  (match token with
    | SEMI ->
          (* Punctuation *)
       "SEMICOLON\n"
    | LPAREN ->
       "LEFT_PAREN\n"
    | RPAREN ->
       "RIGHT_PAREN\n"
    | LBRACE ->
       "LEFT_BRACE\n"
    | RBRACE ->
       "RIGHT_BRACE\n"
    | COMMA ->
       "COMMA\n"
    | PLUS ->
                    (* Arithmetic *)
       "PLUS\n"
    | MINUS ->
       "MINUS\n"
    | TIMES ->
       "TIMES\n"
    | DIVIDE  ->
       "DIVIDE\n"
    | ASSIGN ->
          (* Assignment *)
       "ASSIGN\n"
    | EQ ->
                              (* Comparison Logic *)
    "EQUAL\n"
| NEQ ->
    "NOT_EQUAL\n"
| LT ->
    "LESS_THAN\n"
| LEQ ->
    "LESS_THAN_OR_EQUAL\n"
| GT  ->
    "GREATER_THAN\n"
| GEQ ->
    "GREATER_THAN_OR_EQUAL\n"
| RETURN ->
                (* Control Flow*)
    "RETURN\n"
| IF ->
    "IF\n"
| ELSE ->
    "ELSE\n"
| FOR ->
    "FOR\n"
| WHILE ->
    "WHILE\n"
| INT ->
                (* Data Types *)
    "INT\n"
| BOOLEAN ->
    "BOOLEAN\n"
| STRING ->
    "STRING\n"
| VOID ->                   (* Void *)
    "VOID\n"
| PRINT ->
        (* TaML Exclusive *)
    "PRINT\n"
| POINT ->
    "POINT\n"
| LINE ->
    "LINE\n"
| ELLIPSE ->
          "ELLIPSE\n"
| CURVE ->
    "CURVE\n"
| DRAW ->
         "DRAW\n"
| GETX -> "GETX\n" | GETY -> "GETY\n"
| MPOINT ->
    "MPOINT\n"
| MLINE ->
    "MLINE\n"
| MELLIPSE ->
```

```
              "MELLIPSE\n"
     | MCURVE ->
             "MCURVE\N"
     | INTLITERAL i ->
           (* Literals/Constants *)
         "INTLITERAL: " ^ string_of_int(i) ^ "\n"
     | STRINGLITERAL s ->
         "STRINGLITERAL: " ^ s ^ "\n"
     | ID id ->       (* Variables *)
         "ID: " ^ id ^ "\n"
     | EOF ->        (* End of File *)
         "END_OF_FILE\n"           )

(* Generate a string for a set of tokens *)
let string_of_tokens tokens =
  let token_list = List.map token_to_string tokens in
  String.concat "" token_list
```

6.1.2 Ast Test

This test converts the AST into a string-readable form.

```
open Ast

let string_of_type = function
   | Int -> "TYPE_INT"
   | Point -> "TYPE_POINT"
   | Line -> "TYPE_LINE"
   | Ellipse -> "TYPE_ELLIPSE"
   | Curve -> "TYPE_CURVE"
   (*| LAYER -> "TYPE_LAYER"*)
   | String -> "TYPE_STRING"
   | Void -> "TYPE_VOID"
   | Boolean -> "TYPE_BOOLEAN"

let string_of_binop = function
  Add -> "ADD"
 | Sub -> "SUB"
 | Mult -> "MULT"
 | Div -> "DIV"
 | Equal -> "EQUAL"
 | Neq -> "NEQ"
 | Less-> "LT"
 | Leq -> "LTE"
 | Greater -> "GT"
 | Geq -> "GTE"

let rec string_of_expr = function
   IntLiteral(l) -> "EXPR_LITERAL:[LITERAL_INT:[" ^ (string_of_int l) ^ "]]"
  | StringLiteral(l) -> "EXPR_LITERAL:[LITERAL_STRING:[" ^ l ^ "]]"
  | Id(s) -> "EXPR_ID:[" ^ s ^ "]"
  (*| Dotop(s, op, e) -> "EXPR_DOTOP[" ^ s ^ "," ^ string_of_binop op ^ "," string_of_expr e
^"]"*)
  | MPoint(e1, e2) -> "EXPR_POINT[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^"]"
  | MLine(e1, e2, e3, e4) -> "EXPR_LINE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MEllipse(e1, e2, e3, e4) -> "EXPR_ELLIPSE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^
"," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MCurve(e1, e2, e3, e4, e5, e6) -> "EXPR_CURVE[" ^ string_of_expr e1^ "," ^ string_of_expr
e2 ^"," ^string_of_expr e3^ "," ^ string_of_expr e4 ^string_of_expr e5^ "," ^ string_of_expr e6
^ "]"
  | GetX(e) -> "EXPR_GETX:[" ^ string_of_expr e ^ "]"
  | GetY(e) -> "EXPR_GETY:[" ^ string_of_expr e ^ "]"
(*possibly layer*)
  | Binop(e1, op, e2) -> "EXPR_BINOP:[" ^ string_of_expr e1 ^ "," ^ string_of_binop op ^ "," ^
string_of_expr e2 ^ "]"
  | Assign(e1, e2) -> "EXPR_ASSIGN:[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "]"
  | Call(f, el) ->
     "EXPR_CALL:[" ^ f ^ ",LIST:[" ^ String.concat ", " (List.map string_of_expr el) ^ "]]"

let string_of_init = function
    NullInit(t, s) -> "INIT_NULL:[" ^ string_of_type t ^ "," ^ s ^ "]"
  | VarInit(t,s,e) -> "INIT_VAR:[" ^ string_of_type t ^ "," ^ s ^ "," ^ (string_of_expr e) ^ "]"

let rec string_of_stmt = function
    Block(stmts) ->
     "STMT_BLOCK:[LIST:[" ^ String.concat "," (List.map string_of_stmt stmts) ^ "]]"
  | Expr(expr) -> "STMT_EXPR:[" ^ string_of_expr expr ^ "]";
  | Return(expr) -> "STMT_RETURN:[" ^ string_of_expr expr ^ "]";
  | Print(expr) -> "STMT_PRINT:[" ^ string_of_expr expr ^ "]";
  | If(e, s, NoElseMark) -> "STMT_IF_NOELSE:[" ^ string_of_expr e ^ "," ^ string_of_stmt s ^ "]"
  | If(e, s1, s2) ->  "STMT_IF:[" ^ string_of_expr e ^ "," ^
     string_of_stmt s1 ^ "," ^ string_of_stmt s2 ^ "]"
  | For(e1, e2, e3, s) ->
     "STMT_FOR:[" ^ string_of_expr e1  ^ "," ^ string_of_expr e2 ^ "," ^
     string_of_expr e3  ^ "," ^ string_of_stmt s ^ "]"
  | While(e, s) -> "STMT_WHILE:[" ^ string_of_expr e ^ "," ^ string_of_stmt s ^ "]"
  | NoElseMark -> "STMT_NOELSEMARK"
  | VarDecl(s)->string_of_init s
  | Draw(e) -> "STMT_DRAW[" ^ string_of_expr e ^ "]"

let string_of_varSig = function
  VarSig(t, s) -> "VARSIG:[" ^ string_of_type t ^ "," ^ s ^ "]"

let string_of_func_decl fdecl =
  "FUNCDECL:[FUNTYPE[" ^ string_of_type fdecl.return ^ "],FUNNAME[" ^ fdecl.fname ^
"],LIST:[" ^
  String.concat "," (List.map string_of_varSig fdecl.formals) ^ "],LIST:[" ^
  string_of_stmt fdecl.body ^ "]]"

let string_of_construct = function
```

```
  GlobalVar(i) -> "CONSTRUCT_GLOBALVAR:[" ^ (string_of_init i) ^ "]"
 | FuncDef(f) -> "CONSTRUCT_FUNCDEF:[" ^ (string_of_func_decl f) ^ "]"

let string_of_program construct_list =
 "PROGRAM:[LIST:[" ^ (String.concat "," (List.map string_of_construct construct_list)) ^
"]]\n"
```

6.1.3 SAST Test

This test converts the SAST into a string-readable form.

```
open Ast
open Sast

let string_of_type = function
  | Int -> "TYPE_INT"
  | Point -> "TYPE_POINT"
  | Line -> "TYPE_LINE"
  | Curve -> "TYPE_CURVE"
  | Ellipse -> "TYPE_ELLIPSE"
  (*| LAYER -> "TYPE_LAYER"*)
  | String -> "TYPE_STRING"
  | Void -> "TYPE_VOID"
  | Boolean -> "TYPE_BOOLEAN"

let string_of_binop = function
  Add -> "ADD"
 | Sub -> "SUB"
 | Mult -> "MULT"
 | Div -> "DIV"
 | Equal -> "EQUAL"
 | Neq -> "NEQ"
 | Less-> "LT"
 | Leq -> "LTE"
 | Greater -> "GT"
 | Geq -> "GTE"

let rec string_of_expr = function
  IntLiteral(l) -> "EXPR_LITERAL:[LITERAL_INT:[" ^ (string_of_int l) ^ "]]"
 | StringLiteral(l) -> "EXPR_LITERAL:[LITERAL_STRING:[" ^ l ^ "]]"
 | Id(s) -> "EXPR_ID:[" ^ s ^ "]"
 (*| Dotop(s, op, e) -> "EXPR_DOTOP[" ^ s ^ "," ^ string_of_binop op ^ "," string_of_expr e
^"]"*)
 | GetX(e) -> "EXPR_GETX[" ^ string_of_expr e ^ "]"
 | GetY(e) -> "EXPR_GETX[" ^ string_of_expr e ^ "]"
 | MPoint(e1, e2) -> "EXPR_POINT[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^"]"
 | MLine(e1, e2, e3, e4) -> "EXPR_LINE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
 | MEllipse(e1, e2, e3, e4) -> "EXPR_ELLIPSE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^
"," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
```

```
  | MCurve(e1, e2, e3, e4, e5, e6) -> "EXPR_CURVE[" ^ string_of_expr e1^ "," ^ string_of_expr
e2 ^"," ^string_of_expr e3^ "," ^ string_of_expr e4 ^string_of_expr e5^ "," ^ string_of_expr e6
^ "]"
  (*possibly layer*)
  | Binop(e1, op, e2) -> "EXPR_BINOP:[" ^ string_of_expr e1 ^ "," ^ string_of_binop op ^ "," ^
string_of_expr e2 ^ "]"
  | Assign(e1, e2) -> "EXPR_ASSIGN:[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "]"
  | Call(f, el) ->
     "EXPR_CALL:[" ^ f ^ ",LIST:[" ^ String.concat ", " (List.map string_of_expr el) ^ "]]"

let string_of_sexpr = function
    ExprType(e, t) -> "(EXPR_TYPE:" ^ string_of_type t ^ ")" ^ string_of_expr e

let string_of_init = function
    NullInit(t, s) -> "INIT_NULL:[" ^ string_of_type t ^ "," ^ s ^ "]"
  | VarInit(t,s,e) -> "INIT_VAR:[" ^ string_of_type t ^ "," ^ s ^ "," ^ (string_of_expr e) ^ "]"

let string_of_sinit = function
    SInit(i) ->  string_of_init i


let rec string_of_sstmt = function
    SBlock(stmts) ->
     "STMT_BLOCK:[LIST:[" ^ String.concat "," (List.map string_of_sstmt stmts) ^ "]]"
  | SExpr(expr) -> "STMT_EXPR:[" ^ string_of_sexpr expr ^ "]";
  | SReturn(expr) -> "STMT_RETURN:[" ^ string_of_sexpr expr ^ "]";
  | SPrint(expr) -> "STMT_PRINT:[" ^ string_of_sexpr expr ^ "]";
  | SIf(e, s, SBlock([])) -> "STMT_IF_NOELSE:[" ^ string_of_sexpr e ^ "," ^ string_of_sstmt s ^
"]"
  | SIf(e, s1, s2) ->  "STMT_IF:[" ^ string_of_sexpr e ^ "," ^
     string_of_sstmt s1 ^ "," ^ string_of_sstmt s2 ^ "]"
  | SFor(e1, e2, e3, s) ->
     "STMT_FOR:[" ^ string_of_sexpr e1  ^ "," ^ string_of_sexpr e2 ^ "," ^
     string_of_sexpr e3  ^ "," ^ string_of_sstmt s ^ "]"
  | SWhile(e, s) -> "STMT_WHILE:[" ^ string_of_sexpr e ^ "," ^ string_of_sstmt s ^ "]"
  | SNoElseMark -> "STMT_NOELSEMARK"
  | SVarDecl(s)->string_of_sinit s
  | SDraw(e) -> "STMT_DRAW:[" ^ string_of_sexpr e ^ "]"

let string_of_varSig = function
  VarSig(t, s) -> "VARSIG:[" ^ string_of_type t ^ "," ^ s ^ "]"

let string_of_sfunc_decl fdecl =
  "FUNCDECL:[FUNTYPE[" ^ string_of_type fdecl.sreturn ^ "],FUNNAME[" ^ fdecl.sfname ^
"],LIST:[" ^
  String.concat "," (List.map string_of_varSig fdecl.sformals) ^ "],LIST:[" ^
  string_of_sstmt fdecl.sbody ^ "]]"

let string_of_sconstruct = function
    SGlobalVar(i) -> "CONSTRUCT_GLOBALVAR:[" ^ (string_of_sinit i) ^ "]"
```

```
 | SFuncDef(f) -> "CONSTRUCT_FUNCDEF:[" ^ (string_of_sfunc_decl f) ^ "]"

let string_of_sprogram sconstruct_list =
  "PROGRAM:[LIST:[" ^ (String.concat "," (List.map string_of_sconstruct sconstruct_list)) ^
"]]\n"
```

6.1.4 Overall Test

This test is to ensure the basic function of our whole program can work well. The detailed cases are shown in the next section.

6.2 Test Cases

The detailed test cases information are show in the following table.

Input File	Output File	Testing Target
test_arith1.mdraw	test_arith1.out	Ensure the basic arithmetic operators
test_arith2.mdraw	test_arith2.out	
test_fib.mdraw	test_fib.out	Ensure our program can perform some basic algorithm
test_for.mdraw	test_for.out	To perform the for loop function
test_fun1.mdraw	test_fun1.out	Ensure our program can perform function definition and call
test_fun2.mdraw	test_fun2.out	
test_gcd.mdraw	test_gcd.out	Ensure our program can perform some basic algorithm
test_hello.mdraw	test_hello.out	Perform the basic print function
test_if1.mdraw	test_if1.out	To perform the if conditional statement
test_if2.mdraw	test_if2.out	
test_op.mdraw	test_op.out	Ensure the basic binary operators
test_while.mdraw	test_while.out	To perform the while loop function
test_global.mdraw	test_global.out	Ensure global variable can work in our program
test_point.mdraw	test_point.out	Ensure the MPoint function and getX and getY method can work
test_line.mdraw	test_line.out	Ensure the MLine function can work
test_curve.mdraw	test_curve.out	Ensure the MCurvefunction can work
test_ellipse.mdraw	test_ellipse.out	Ensure the MEllipse function can work

7 Lessons Learned

7.1 Jingyu Shi

There are many things I love about this project, one is knowing and applying the process of how a compiler works, another is the Ocaml language. I learnt to enjoy programming in Ocaml, it really is a lot of fun to program in, and it is so efficient and succinct.

I was drafting the scanner and parser together with the ast file at first, and run into some obstacles and I realize it is really important to break up the work to individuals and communicate at the same time.

Another thing I learnt is always read a lot of code before you get started, to understand what you want to achieve, reading the projects from past teams is very very helpful.

One of the mistakes I learnt from this project is that we should make our mind on what we want our language to look like and focus on the compile and interpreter part, because we changed in the middle and start over and go back and forth, which wasted a lot of time . Getting the scanner, parser and ast done early is really helpful. It is also very important to communicate with your teammate, because when we were writing the manual, there are many specific details we can't agree on. We decided on one way and later changed our idea, but we wasted a lot of time and effort.

7.2 Huimin Sun

This project helped me to enhance my knowledge on compilers. Implementation of this project helped me learn the various phases involved in writing an interpreter. It clarified the concepts of how the data flows between various components involved in the process and how the data is filtered for each successive component. This project also gave me an opportunity to learn O'caml, though it wasn't easy to start with.

I am responsible for the compiler, translator and the final run shell of our program. Actually I run into some problems such as the converting our language to java code. Since paint is a must-have function in java in order to use the awt geometry library, I have to split our function into paint function and basic function.

Apart from the technical aspects, this project also taught me the lessons on teamwork, code organization and time management. I am not a team leader, but I didn't do anything less than the leader. Time management is very important and starting early really helps us save a lot of time. Since we have set up some deadlines for the entire process and sticked to those headlines closely, we can finish our project in time. Of course, the professor and TA also help us a lot.

7.3 Dongxiang Yan

I really enjoy this project except the time constraints. First of all, I'd like to say thanks to my teammate Jingyu and Huimin, thanks to TA Qiuzi, and thanks to Professor Edwards. Without instruction from Professor and TA, without team collaborative works, this project cannot be completed on time. There are two main points I learned from this project.

The first point is about the knowledge of programming language translator. Before taking this course, I have no clear understanding about programming language translator & compiler. It means I haven't had a chance to pay real attention on how to design and compile a programming language. This PLT course is well prepared and organized. Besides the concepts and principles I learned in class, we also have this opportunity to design and build a real programming language. This practical experience is much valuable. A sparrow may be small, but its body has every organ it needs.

The second point is about the project management. It will never be an easy work to complete this project in such a short period without well-organized team and dedicated project plan. Scrum methodology is applied to the project progress. In total six weeks, we follow each milestones and meet all timelines. A good start is a half way to success. Although at the very beginning, we have struggled with architecture design and choosing proper compilation language, we overcome these issues by dedicated group meeting and active discussion with TA.

8 Appendix (Code Listing)

8.1 MDraw_src

8.1.1 scanner.mll

```
{ open Parser}  (* Get the token types *)

rule token = parse
          [' ' '\t' '\r' '\n'] { token lexbuf }  (* Whitespace *)
| "/*"    { comment lexbuf }          (* Comments *)
| '('    { LPAREN }
| ')'    { RPAREN }
| '{'    { LBRACE }
| '}'    { RBRACE }
| ';'    { SEMI }
| ','    { COMMA }

(* binary operations *)
```

```
| '+'     { PLUS }
| '-'     { MINUS }
| '*'     { TIMES }
| '/'     { DIVIDE }
| '='     { ASSIGN }

(*arithmetic operations *)
| "=="    { EQ }
| "!="    { NEQ }
| '<'     { LT }
| "<="    { LEQ }
| ">"     { GT }
| ">="    { GEQ }

(*control flow *)
| "if"    { IF }
| "else"  { ELSE }
| "for"   { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "print" {PRINT}

| "int"   { INT }
| "Point"  { POINT }
| "Line"   { LINE }
| "Ellipse"{ ELLIPSE }
| "Curve"  { CURVE }
| "string" {STRING}
| "void"   {VOID}
| "boolean" {BOOLEAN}
| ".draw()" {DRAW}
| "MEllipse" {MELLIPSE}
| "MLine" {MLINE}
| "MPoint" {MPOINT}
| "MCurve" {MCURVE}
| ".getX()" {GETX}
| ".getY()" {GETY}

| ['0'-'9']+ as lxm { INTLITERAL(int_of_string lxm) }
| ['\"'] ([^ '\"']* as lit) ['\"'] { STRINGLITERAL(lit) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }  (* End-of-comment *)
| _   { comment lexbuf }  (* Eat everything else *)
```

8.1.2 scanner_test.ml

```
(* scanner_utils.ml - This utility is used for testing purposes.    *)
(* It outputs the results of the scanner (tokens) into string form   *)


open Scanner
open Parser

(* generate a string for each token. *)
let token_to_string token =
  (match token with
    | SEMI ->
          (* Punctuation *)
       "SEMICOLON\n"
    | LPAREN ->
       "LEFT_PAREN\n"
    | RPAREN ->
       "RIGHT_PAREN\n"
    | LBRACE ->
       "LEFT_BRACE\n"
    | RBRACE ->
       "RIGHT_BRACE\n"
    | COMMA ->
       "COMMA\n"
    | PLUS ->
                    (* Arithmetic *)
       "PLUS\n"
    | MINUS ->
       "MINUS\n"
    | TIMES ->
       "TIMES\n"
    | DIVIDE  ->
       "DIVIDE\n"
    | ASSIGN ->
          (* Assignment *)
       "ASSIGN\n"
    | EQ ->         (* Comparison Logic *)
       "EQUAL\n"
    | NEQ ->
       "NOT_EQUAL\n"
    | LT ->
       "LESS_THAN\n"
    | LEQ ->
       "LESS_THAN_OR_EQUAL\n"
    | GT  ->
       "GREATER_THAN\n"
    | GEQ ->
       "GREATER_THAN_OR_EQUAL\n"
    | RETURN ->           (* Control Flow*)
       "RETURN\n"
    | IF ->
        "IF\n"
    | ELSE ->
        "ELSE\n"
    | FOR ->
        "FOR\n"
    | WHILE ->
        "WHILE\n"
    | INT ->        (* Data Types *)
        "INT\n"
    | BOOLEAN ->
        "BOOLEAN\n"
    | STRING ->
        "STRING\n"
    | VOID ->         (* Void *)
            "VOID\n"
    | PRINT ->    (* TaML Exclusive *)
            "PRINT\n"
    | POINT ->
        "POINT\n"
    | LINE ->
        "LINE\n"
    | ELLIPSE ->
             "ELLIPSE\n"
    | CURVE ->
        "CURVE\n"
    | DRAW ->
            "DRAW\n"
    | GETX -> "GETX\n" | GETY -> "GETY\n"
    | MPOINT ->
        "MPOINT\n"
    | MLINE ->
        "MLINE\n"
    | MELLIPSE ->
             "MELLIPSE\n"
    | MCURVE ->
            "MCURVE\N"
    | INTLITERAL i ->
           (* Literals/Constants *)
        "INTLITERAL: " ^ string_of_int(i) ^ "\n"
    | STRINGLITERAL s ->
        "STRINGLITERAL: " ^ s ^ "\n"
    | ID id ->      (* Variables *)
        "ID: " ^ id ^ "\n"
    | EOF ->       (* End of File *)
        "END_OF_FILE\n"          )

(* Generate a string for a set of tokens *)
let string_of_tokens tokens =
  let token_list = List.map token_to_string tokens in
  String.concat "" token_list
```

8.1.3 parser.mly

```
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR WHILE
%token INT POINT LINE CURVE ELLIPSE STRING VOID BOOLEAN
%token <int> INTLITERAL
%token <string> STRINGLITERAL
%token <string> ID
%token MPOINT MLINE MELLIPSE MCURVE
%token ASSIGN
%token PRINT
%token DRAW GETX GETY
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE

%start program
%type <Ast.program> program

%%

program:
   program_r { List.rev $1 }

program_r:
  /* nothing */ { [] }
 | program_r fdecl { FuncDef($2)::$1 }
 | program_r vdecl { GlobalVar($2)::$1 }


/* List of all return types */
Type:
     INT {Int}
   | POINT {Point}
   | LINE {Line}
   | ELLIPSE {Ellipse}
   | CURVE {Curve}
   /*| LAYER {Layer}*/
   | STRING {String}  /* Need void here?*/
   | BOOLEAN {Boolean}
   | VOID {Void}

fdecl:
   Type ID LPAREN formals_opt RPAREN block_stmt
    { {
       return = $1;
       fname = $2;
       formals = $4;
       body = $6; } }


formals_opt:
   /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
   Type ID { [VarSig($1,$2)] }
  | formal_list COMMA Type ID { VarSig($3,$4) :: $1 }

vdecl:
    Type ID SEMI { NullInit($1,$2) }   /*default initialization*/
  | Type ID ASSIGN expr SEMI { VarInit($1,$2,$4) }   /*explicit initialization*/

block_stmt:
   LBRACE stmt_list RBRACE { Block (List.rev $2) }

stmt_list:
   /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
   expr SEMI { Expr($1) }
  | RETURN expr SEMI { Return($2) }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, NoElseMark) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr SEMI expr SEMI expr RPAREN block_stmt
    { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | PRINT expr SEMI { Print($2)}
  | expr DRAW SEMI      { Draw($1)}
  | vdecl {VarDecl($1)}


expr:
  | INTLITERAL         { IntLiteral($1) }
  | STRINGLITERAL       { StringLiteral($1) }
  | ID                  { Id($1) }
  | MPOINT LPAREN expr COMMA expr RPAREN { MPoint($3, $5)}
  | MLINE LPAREN expr COMMA expr COMMA expr COMMA expr RPAREN { MLine($3, $5, $7,
$9)}
  | MELLIPSE LPAREN expr COMMA expr COMMA expr COMMA expr RPAREN { MEllipse($3,
$5, $7, $9)}
  | MCURVE LPAREN expr COMMA expr COMMA expr COMMA expr COMMA expr COMMA
expr RPAREN { MCurve($3, $5, $7, $9, $11, $13)}
  | expr GETX {GetX($1)}
  | expr GETY {GetY($1)}
  | expr PLUS   expr       { Binop($1, Add,   $3) }
  | expr MINUS  expr       { Binop($1, Sub,   $3) }
  | expr TIMES  expr       { Binop($1, Mult,  $3) }
  | expr DIVIDE expr       { Binop($1, Div,   $3) }
  | MINUS  expr            { Binop(IntLiteral(0),  Sub,   $2) }
  | expr EQ     expr       { Binop($1, Equal, $3) }
  | expr NEQ    expr       { Binop($1, Neq,   $3) }
  | expr LT     expr       { Binop($1, Less,  $3) }
  | expr LEQ    expr       { Binop($1, Leq,   $3) }
  | expr GT     expr       { Binop($1, Greater,  $3) }
  | expr GEQ    expr       { Binop($1, Geq,   $3) }
  | expr ASSIGN expr       { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN
                           { Call($1, $3) }    /* function call*/
  | LPAREN expr RPAREN          { $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

8.1.4 parser_test.ml

```
(* parser_test.ml - This is used for testing purposes.    *)
(* It outputs the results of an AST in string-readable form        *)

open Ast

let string_of_type = function
    | Int -> "TYPE_INT"
    | Point -> "TYPE_POINT"
    | Line -> "TYPE_LINE"
    | Ellipse -> "TYPE_ELLIPSE"
    | Curve -> "TYPE_CURVE"
    (*| LAYER -> "TYPE_LAYER"*)
    | String -> "TYPE_STRING"
    | Void -> "TYPE_VOID"
    | Boolean -> "TYPE_BOOLEAN"
```

```
let string_of_binop = function
   Add -> "ADD"
 | Sub -> "SUB"
 | Mult -> "MULT"
 | Div -> "DIV"
 | Equal -> "EQUAL"
 | Neq -> "NEQ"
 | Less-> "LT"
 | Leq -> "LTE"
 | Greater -> "GT"
 | Geq -> "GTE"

let rec string_of_expr = function
    IntLiteral(l) -> "EXPR_LITERAL:[LITERAL_INT:[" ^ (string_of_int l) ^ "]]"
  | StringLiteral(l) -> "EXPR_LITERAL:[LITERAL_STRING:[" ^ l ^ "]]"
  | Id(s) -> "EXPR_ID:[" ^ s ^ "]"
  (*| Dotop(s, op, e) -> "EXPR_DOTOP[" ^ s ^ "," ^ string_of_binop op ^ "," string_of_expr e
^"]"*)
  | MPoint(e1, e2) -> "EXPR_POINT[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^"]"
  | MLine(e1, e2, e3, e4) -> "EXPR_LINE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MEllipse(e1, e2, e3, e4) -> "EXPR_ELLIPSE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^
"," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MCurve(e1, e2, e3, e4, e5, e6) -> "EXPR_CURVE[" ^ string_of_expr e1^ "," ^ string_of_expr
e2 ^"," ^string_of_expr e3^ "," ^ string_of_expr e4 ^string_of_expr e5^ "," ^ string_of_expr e6
^ "]"
  | GetX(e) -> "EXPR_GETX:[" ^ string_of_expr e ^ "]"
  | GetY(e) -> "EXPR_GETY:[" ^ string_of_expr e ^ "]"
(*possibly layer*)
  | Binop(e1, op, e2) -> "EXPR_BINOP:[" ^ string_of_expr e1 ^ "," ^ string_of_binop op ^ "," ^
string_of_expr e2 ^ "]"
  | Assign(e1, e2) -> "EXPR_ASSIGN:[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "]"
  | Call(f, el) ->
     "EXPR_CALL:[" ^ f ^ ",LIST:[" ^ String.concat ", " (List.map string_of_expr el) ^ "]]"

let string_of_init = function
    NullInit(t, s) -> "INIT_NULL:[" ^ string_of_type t ^ "," ^ s ^ "]"
  | VarInit(t,s,e) -> "INIT_VAR:[" ^ string_of_type t ^ "," ^ s ^ "," ^ (string_of_expr e) ^ "]"

let rec string_of_stmt = function
    Block(stmts) ->
     "STMT_BLOCK:[LIST:[" ^ String.concat "," (List.map string_of_stmt stmts) ^ "]]"
  | Expr(expr) -> "STMT_EXPR:[" ^ string_of_expr expr ^ "]";
  | Return(expr) -> "STMT_RETURN:[" ^ string_of_expr expr ^ "]";
  | Print(expr) -> "STMT_PRINT:[" ^ string_of_expr expr ^ "]";
  | If(e, s, NoElseMark) -> "STMT_IF_NOELSE:[" ^ string_of_expr e ^ "," ^ string_of_stmt s ^ "]"
  | If(e, s1, s2) ->  "STMT_IF:[" ^ string_of_expr e ^ "," ^
     string_of_stmt s1 ^ "," ^ string_of_stmt s2 ^ "]"
  | For(e1, e2, e3, s) ->
     "STMT_FOR:[" ^ string_of_expr e1  ^ "," ^ string_of_expr e2 ^ "," ^
     string_of_expr e3  ^ "," ^ string_of_stmt s ^ "]"
  | While(e, s) -> "STMT_WHILE:[" ^ string_of_expr e ^ "," ^ string_of_stmt s ^ "]"
  | NoElseMark -> "STMT_NOELSEMARK"
  | VarDecl(s)->string_of_init s
  | Draw(e) -> "STMT_DRAW[" ^ string_of_expr e ^ "]"

let string_of_varSig = function
 VarSig(t, s) -> "VARSIG:[" ^ string_of_type t ^ "," ^ s ^ "]"

let string_of_func_decl fdecl =
  "FUNCDECL:[FUNTYPE[" ^ string_of_type fdecl.return ^ "],FUNNAME[" ^ fdecl.fname ^
"],LIST:[" ^
  String.concat "," (List.map string_of_varSig fdecl.formals) ^ "],LIST:[" ^
  string_of_stmt fdecl.body ^ "]]"

let string_of_construct = function
    GlobalVar(i) -> "CONSTRUCT_GLOBALVAR:[" ^ (string_of_init i) ^ "]"
  | FuncDef(f) -> "CONSTRUCT_FUNCDEF:[" ^ (string_of_func_decl f) ^ "]"

let string_of_program construct_list =
  "PROGRAM:[LIST:[" ^ (String.concat "," (List.map string_of_construct construct_list)) ^
"]]\n"
```

8.1.5 ast.mli

```
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq

type data_type =
    Int
  | Point
  | Line
  | Curve
  (*| Layer  *)
  | String
  | Boolean
  | Void
  | Ellipse


type expr =
   IntLiteral of int
 | StringLiteral of string
 | Id of string
 | MPoint of expr * expr
 | MLine of expr * expr * expr * expr
 | MEllipse of expr * expr * expr * expr
 | MCurve of expr * expr * expr * expr * expr * expr
 | Binop of expr * op * expr
```

```
  | Assign of expr * expr
  | Call of string * expr list
  | GetX of expr
  | GetY of expr
  (*| Layer of expr list*)

(*type and name of a variable*)
type varSig = VarSig of data_type * string

(*initialization of variable*)
type init =
  | NullInit of data_type * string   (*assign null/default value*)
  | VarInit of data_type * string * expr   (*assgin a specific value to the new variable*)


(*statement declairation*)
type stmt =
  | Block of stmt list              (* LBRACE Statement_list RBRACE *)
  | Expr of expr                    (* expr SEMICOLON *)
  | Return of expr
  | If of expr * stmt * stmt          (* IF LPAREN expr RPAREN Statement NOELSE SEMICOLON
*)
  | For of expr * expr * expr * stmt    (* FOR LPAREN expr SEMICOLON expr SEMICOLON expr
Statement SEMICOLON *)
  | While of expr * stmt             (* WHILE LPAREN expr RPAREN Statement SEMICOLON *)
  | Print of expr
  | NoElseMark
  | VarDecl of init
  | Draw of expr


(*function declairation*)
type func_decl = {
   return : data_type;
   fname : string;
   formals : varSig list;
   body : stmt;
}


(* Bascially, the parts of a program *)
type construct =
   GlobalVar of init
  | FuncDef of func_decl


type program = construct list (* global vars, funcs *)
```

8.1.6 sast.mli

```
(* sast.mli -- Semanitcally Checked and Typed Abstract Syntax Tree *)

open Ast

(* Expressions *)
type sexpr =
  ExprType of expr * data_type

type  sinit =
  SInit of init

type sstmt =
   SBlock of sstmt list
  | SExpr of sexpr                                                        (*foo =
bar + 3; *)
  | SReturn of sexpr                                                      (*
return 42 also includes return function_name *)
  | SPrint of sexpr
  | SIf of sexpr * sstmt * sstmt                          (* if (foo == 42) {} else {} *)
  | SFor of sexpr * sexpr * sexpr * sstmt           (* for loop *)
  | SWhile of sexpr * sstmt                                           (* while (i<10)
{ i = i + 1 } *)
  | SNoElseMark
                    (* For loops without else*)
  | SVarDecl of sinit
  | SDraw of sexpr

type sfunc_decl = {
   sreturn : data_type;
                    sfname : string;
                    sformals : varSig list;    (* Formal arguments, type & names *)
                    sbody : sstmt;
}

(* Bascially, the parts of a program *)
type sconstruct =
   SGlobalVar of sinit
  | SFuncDef of sfunc_decl

type sprogram = sconstruct list (* global vars, funcs *)
```

8.1.7 compile.ml

```
(* Translate a program in AST form into a java program.  Throw an
   exception if something is wrong, e.g., a reference to an unknown
   variable or function *)

open Ast
open Sast
```

```
let imports = "import java.awt.Graphics;\n" ^ "import java.awt.Point;\n" ^ "import
java.awt.geom.*;\n" ^ "import java.awt.Graphics2D;\n"

let default_paint = "public void paint(Graphics g)"

let default_keyword = "public void "(*put before  global variables and functions*)
let default_implicit_param = "null"                    (*for null declaration*)

let main_func = "public static void main(String args[]) {\n" ^ "test t = new test();\n" ^
"t.setVisible(true);\n"
   ^ "t.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);\n" ^
"t.setSize(600, 600);\n" ^ "}\n"

(*translate binary operators*)
let string_of_operator = function
             Add -> "+"
           | Sub -> "-"
           | Mult -> "*"
           | Div -> "/"
   | Equal -> "=="
   | Neq -> "!="
   | Less -> "<"
   | Leq -> "<="
           | Greater -> ">"
           | Geq -> ">="

(*translate types*)
let string_of_type =function
     Int ->"int"
   | String -> "String"
   | Point -> "Point2D"
   | Line -> "Line2D"
   | Ellipse -> "Ellipse2D"
   | Curve -> "Arc2D"
   | Boolean -> "Boolean"



(*translate expression*)
let rec string_of_expr = function
   |IntLiteral(l) -> string_of_int l
   |StringLiteral(l) -> "\"" ^ l ^ "\""
   |MLine(e1, e2, e3, e4) -> ".setLine(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^ ")"
   |MEllipse(e1, e2, e3, e4) -> ".setFrame(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^ ")"
   |MCurve(e1, e2, e3, e4, e5, e6) -> ".setArc(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^
"," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^ "," ^ string_of_expr e5 ^ "," ^ string_of_expr
e6 ^ ", Arc2D.OPEN)"
   |Id(s) -> s
   |MPoint(e1, e2) -> ".setLocation(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ ")"
   |GetX(e) -> "(int) " ^ string_of_expr e ^ ".getX()"
   |GetY(e) -> "(int) " ^ string_of_expr e ^ ".getY()"
   |Binop(e1, op, e2) -> string_of_expr e1 ^ " " ^ string_of_operator(op) ^ " " ^ string_of_expr
e2
   |Assign(s, MLine(e1, e2, e3, e4)) -> string_of_expr s ^ ".setLine(" ^ string_of_expr e1 ^ "," ^
string_of_expr e2 ^ "," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^ ")"
   |Assign(s, MPoint(e1, e2)) -> string_of_expr s ^ ".setLocation(" ^ string_of_expr e1 ^ "," ^
string_of_expr e2 ^ ")"
   |Assign(s, MEllipse(e1, e2, e3, e4)) -> string_of_expr s ^ ".setFrame(" ^ string_of_expr e1 ^
"," ^ string_of_expr e2 ^ "," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^ ")"
   |Assign(s, MCurve(e1, e2, e3, e4, e5, e6)) -> string_of_expr s ^ ".setArc(" ^ string_of_expr
e1 ^ "," ^ string_of_expr e2 ^ "," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^ "," ^
string_of_expr e5 ^ "," ^ string_of_expr e6 ^ ", Arc2D.OPEN)"
   |Assign(s, e) -> string_of_expr s ^ " = " ^ string_of_expr e
   |Call(s, e_list) -> s ^ "(" ^ String.concat "," (List.map string_of_expr e_list) ^ ")"    |_ ->
"never get there!"

(*translate the expression from Sast*)
let rec string_of_exprs = function
   | ExprType(e,_) -> string_of_expr e

(*translation of init from Sast*)
let string_of_init = function
   | NullInit(Point, name) -> "Point2D " ^ name ^ " = new Point2D.Float()"
   | NullInit(Line, name) -> "Line2D " ^ name ^ " = new Line2D.Float()"
   | NullInit(Ellipse, name) -> "Ellipse2D " ^ name ^ " = new Ellipse2D.Float()"
   | NullInit(Curve, name) -> "Arc2D " ^ name ^ " = new Arc2D.Float()"
   | NullInit(t, s) -> string_of_type t ^ " " ^ s
   | VarInit(Point, name, MPoint(e1, e2)) -> "Point2D " ^ name ^ " = new Point2D.Float();\n"
            ^ name ^ ".setLocation(" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2
   | VarInit(Line, name, MLine(e1, e2, e3, e4)) -> "Line2D" ^ name ^ " = new
Line2D.Float();\n"
            ^ name ^ ".setLine(" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2 ^ ", " ^
string_of_expr e3 ^ ", " ^ string_of_expr e4
   | VarInit(Ellipse, name, MEllipse(e1, e2, e3, e4)) -> "Ellipse2D" ^ name ^ " = new
Ellipse2D.Float();\n"
   | VarInit(Curve, name, MCurve(e1, e2, e3, e4, e5, e6)) -> "Arc2D" ^ name ^ " = new
Arc2D.Float();\n"
   | VarInit(t, s, e) -> string_of_type t ^ " " ^ s ^ " = " ^ string_of_expr e

let rec string_of_inits = function
   SInit(i) -> string_of_init (i)



(*translate statement*)
let rec string_of_stmt = function
   |SBlock(stmts) -> "{\n" ^ String.concat "" (List.map string_of_stmt stmts)^ "}\n"
   |SExpr(e) -> string_of_exprs e ^ ";\n"
   |SNoElseMark -> "{;}\n"
   |SReturn(e) -> "return " ^ string_of_exprs e ^ ";\n"
   |SIf(e, s1, SNoElseMark) -> "if (" ^ string_of_exprs e ^ ")\n" ^ string_of_stmt s1
   |SIf(e, s1, s2) -> "if (" ^ string_of_exprs e ^ ")\n" ^ string_of_stmt s1 ^ "else\n" ^
string_of_stmt s2
   |SFor(e1, e2, e3, stmt) -> "for (" ^ string_of_exprs e1  ^ ";" ^string_of_exprs e2 ^ ";" ^
string_of_exprs e3 ^ ")\n" ^ string_of_stmt stmt ^ "\n"
   |SWhile(e, s) -> "while (" ^ string_of_exprs e ^ ")\n" ^ string_of_stmt s ^ ";\n"
   |SPrint(e) -> "System.out.println(" ^ string_of_exprs e ^ ");\n"
   |SVarDecl(l)-> string_of_inits l ^ ";\n"
   |SDraw(e) -> "g2.draw(" ^ string_of_exprs e ^ ");\n"


(*translation of variable signature*)
let string_of_varsig = function
            VarSig (t, id) ->string_of_type t ^ " " ^ id


(*translate function declairation*)
let string_of_func_decl func_decl =
   match func_decl.sfname with
    | "basic" -> "public void basic()\n{\n" ^ string_of_stmt func_decl.sbody^ "}\n"
    | "paint" -> "public void paint(Graphics g)\n{\nGraphics2D g2 = (Graphics2D) g;\n" ^
string_of_stmt func_decl.sbody ^ "}\n"
    | _ ->  "public " ^ string_of_type func_decl.sreturn ^ " " ^ func_decl.sfname ^ "(" ^
(String.concat "," (List.map string_of_varsig func_decl.sformals))
                    ^ ")\n{\n" ^ string_of_stmt func_decl.sbody ^ "}\n"


(*translation of constructs*)
let string_of_construct =function
   SGlobalVar(l)->"public static " ^ string_of_inits l ^ ";\n"
  | SFuncDef (f)->string_of_func_decl f

(*translation of construct list and the generate whole java code*)
let string_of_class class_name programs =
            "\n" ^ imports ^ "\n" ^ "\npublic class " ^ class_name ^ " extends
javax.swing.JFrame{\n" ^ String.concat "\n" (List.map string_of_construct programs) ^
"public static void main(String args[]){\n" ^ class_name ^ " t = new " ^ class_name ^
"();\nt.setVisible(true);\nt.basic();\nt.setDefaultCloseOperation(javax.swing.WindowConst
ants.EXIT_ON_CLOSE);\nt.setSize(600,600);\n}\n}\n"
```

8.1.8 translate.ml

```
open Ast
open Sast

type mode = Quiet | Verbose (* compiler directives *)

let print_warning mode w = match mode with
  | Verbose -> print_string("/* WARNING: " ^ w ^ "*/\n")
```

```
 | Quiet   -> print_string ""

(* Exception types *)
exception ReturnException of string (* Missing return statement in function *)
exception VariableNotFoundException of string (* Using an undeclared variable *)
exception FunctionNotFoundException of string (* Using an undeclared function *)
exception BinopException of string (* Mismatching types in binop *)
exception OtherException of string (* Misc exception *)
exception AssignmentException of string (* Assigning incompatable things *)
exception FunctionException of string (* Incorrect call to built in functions *)
exception WhileExprException of string (* Incorrect specification of while *)
exception IfExprException of string (* Incorrect specification of if *)
exception ReturnTypeException of string (* Mismatching return statement *)
exception MainNotFoundException of string (* Missing main function *)
exception Error of string (* Placeholder - TODO remove *)

(* Convert data types into string*)
let string_of_type tp = match tp with
     Int -> "Int"
   | String -> "String"
   | Boolean -> "Boolean"
   | Point -> "Point"
   | Curve -> "Curve"
   | Line -> "Line"
   | Ellipse -> "Ellipse"
   | Void -> "Void"

(* Define a symbol table*)
type symbol_table = {
  parent : symbol_table option;
  variables : (string * data_type) list   (* name and type*)
}

(* Define a function table*)
(* all function names are global, no nested funcs allowed, so just build a table of all functions
upon program initialization*)
type function_table = {
  functions: (string * data_type * data_type list * stmt) list  (*function name, return type,
argument type list, statement *)
}


(* Define the environment *)
type translation_environment = {
  return_type: data_type;            (* Function's return type *)
  return_seen: bool;                                    (* If there is a return statement *)
  var_scope: symbol_table;   (* symbol table for vars *)
  fun_scope: function_table; (* list of functions *)
}
```

```
(* Add variables' name and type to an envirionment's var_scope*)
let add_var_env env n t =
 let new_vars = (n,t) :: env.var_scope.variables in
 let new_sym_table = {parent = env.var_scope.parent; variables = new_vars} in
 let new_env = {env with var_scope = new_sym_table} in
 new_env

(* Search for variable in the symbol tables *)
let rec find_var_table (var_scope : symbol_table) name =
 try
  List.find (fun (n,_) -> n = name) var_scope.variables
 with Not_found ->
  match var_scope.parent with
    Some(parent) -> find_var_table parent name
   | _ -> raise Not_found

(* Search for function in environment*)
let rec find_function (fun_scope : function_table) name =
  List.find (fun (s,_,_,_) -> s = name) fun_scope.functions

(* get type of string literal and int literal *)
let get_int_literal_type l = Int
let get_string_literal_type l = String

(* check whether variable v has type t *)
let check_var_type env v t =
 let (id, id_t) = find_var_table env.var_scope v in
 if (id_t = t) then true else false

(* check whether both sides of binop are semantically compatible*)
let check_compatable_types_binop t1 t2 = match (t1,t2) with
  (Int, Int)                  -> true
 | (String, String)                      -> true
 | (Point, Point)                  -> true
 | (Line, Line)                -> true
 | (Curve, Curve)                            -> true
 | (Ellipse, Ellipse)          -> true
 | _ -> false

(* check both sides of assign are semantically compatible*)
let check_compatable_types_assign t1 t2 = match (t1,t2) with
  (Int, Int)   -> true
 | (String, String)       -> true
 | (Point, Point) -> true
 | (Line, Line) -> true
 | (Ellipse, Ellipse) -> true
 | (Curve, Curve)       -> true
 | (tp, fp) -> if (tp = fp) then true else false

(* Get type of an expression and semantically check it *)
```

```
let rec get_type_for_expr env e = match e with
  | IntLiteral(i) -> let t = get_int_literal_type i in t
  | StringLiteral(s) -> let t = get_string_literal_type s in t
  | Id(s) -> let (_, t) =
      try find_var_table env.var_scope s
      with Not_found -> raise (Error("Undeclared Identifier " ^ s)) in t
  | GetX(e) ->
            let t = get_type_for_expr env e in (if not (t = Point) then raise (Error("This is not a
point"))); Int
  | GetY(e) ->
            let t = get_type_for_expr env e in (if not (t = Point) then raise (Error("This is not a
point"))); Int
  | MPoint(e1, e2) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 in
      (if not (t1 = Int) || not (t2 = Int)
         then raise (Error("This is not an integer point" ))); Point
  | MLine(e1, e2, e3, e4) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 and
        t3 = get_type_for_expr env e3 and t4 = get_type_for_expr env e4 in
      (if not (t1 = Int) || not (t2 = Int) || not (t3 = Int) || not (t4 = Int)
         then raise (Error("This is not an integer line" ))); Line
  | MEllipse(e1, e2, e3, e4) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 and
        t3 = get_type_for_expr env e3 and t4 = get_type_for_expr env e4 in
      (if not (t1 = Int) || not (t2 = Int) || not (t3 = Int) || not (t4 = Int)
         then raise (Error("This is not an integer ellipse" ))); Ellipse
  | MCurve(e1, e2, e3, e4, e5, e6) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 and
        t3 = get_type_for_expr env e3 and t4 = get_type_for_expr env e4 and t5 =
get_type_for_expr env e5 and t6 = get_type_for_expr env e6 in
      (if not (t1 = Int) || not (t2 = Int) || not (t3 = Int) || not (t4 = Int) || not (t5 = Int) || not (t6 =
Int)
         then raise (Error("This is not an integer curve" ))); Curve
  | Binop(e1, op, e2) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 in
      (if not (check_compatable_types_binop t1 t2) then
        raise (Error("Mismatch in types for binary operator: " ^ string_of_type t1 ^ ":" ^
string_of_type t2)));
      if op = Equal || op = Neq || op = Less || op = Leq || op = Greater || op = Geq then Boolean
else t1
  | Assign(e1, e2) ->
      let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 in
      (if not (check_compatable_types_assign t1 t2) then
         raise (Error("Mismatch in types for assignment"))); t1
  | Call(s, el) ->
      let (fname, fret, farg, fbody) = try find_function env.fun_scope s with Not_found ->
        raise (Error("Undeclared Function " ^ s)) in
      let el_type = List.map (fun s -> get_type_for_expr env s) el in
      let arg_type = farg in
```

```
   (if not (el_type = arg_type) then raise (Error("Mismatching types in function call for: " ^
s))); fret


(*define semantically checked expression*)
let rec get_sexpr_for_expr env e = let t = get_type_for_expr env e in ExprType(e,t)

(*get type of a variable's signature*)
let get_varsig_type vs = let VarSig(t,s) = vs in t

(*get name and type of a variable's signature*)
let get_varsig_name_type vs = let VarSig(t,s) = vs in (s, t)

(*Add function to env and return updated env*)
let add_func_to_env env f =
 let ftable = env.fun_scope in let old_func = ftable.functions in let fun_name = f.fname in
 let argtype = List.map (fun vs -> get_varsig_type vs) f.formals in let fun_stmt = f.body in
 let fun_ret = f.return in let new_func = (fun_name, fun_ret, argtype, fun_stmt) :: old_func in
 let new_fun_scope = {functions = new_func} in let new_env = {env with fun_scope =
new_fun_scope} in
 new_env

(* Add a newly initialized var to env and return updated env *)
let add_init_var env ivar =
 let env = match ivar with
  | NullInit(t, s) -> add_var_env env s t
  | VarInit(t, s, e) -> add_var_env env s t in
 let (t, env) = match ivar with
  | NullInit(t, s) -> (t, env)
  | VarInit(t, s, e) -> (t, env) in (t, env)

(* Combine two environments. it is not used*)
let rec combine_env env lenv =
 let ret_seen = List.fold_left (fun a e -> a && e.return_seen) true lenv in
 let new_env = {env with return_seen = ret_seen} in
 new_env

(* Get type of a statement and semantically check it *)
let rec get_env_for_stmt env st = match st with
 | Block(sl) ->
  let new_var_scope = {parent = Some(env.var_scope); variables = [] } in
  let new_env = {env with var_scope = new_var_scope} in
  let help(env, acc) s = let (st, e) = get_env_for_stmt env s in (e, st::acc) in
  let (nenv, st) = List.fold_left (fun e s -> help e s) (new_env, []) sl in
  let revst = List.rev st in (SBlock(revst), nenv)
 | Expr(e) ->
  let _ = get_type_for_expr env e in (SExpr(get_sexpr_for_expr env e), env)
 | Return(e) ->
  let t = get_type_for_expr env e in
  (if not (t = env.return_type) then raise (Error("Incompatable Return type")));
```

```
   let new_env = {env with return_seen = true} in (SReturn(get_sexpr_for_expr env e),
new_env)
  | If(e, s1, s2) ->
   let t = get_type_for_expr env e in (if not (t = Boolean) then

            raise (Error("Non-boolean IF predicate")));
   let (st1, new_env1) = get_env_for_stmt env s1 and (st2, new_env2) = get_env_for_stmt env
s2 in
   let ret_seen = (new_env1.return_seen && new_env2.return_seen) in
   let new_env = {env with return_seen = ret_seen} in (SIf((get_sexpr_for_expr env e), st1,
st2), new_env)
  | For(e1, e2, e3, s) ->
   let t1 = get_type_for_expr env e1 and t2 = get_type_for_expr env e2 and t3 =
get_type_for_expr env e3 in
   (if not (t1 = Int && t2 = Boolean && t3 = Int) then raise (Error("Bad FOR loop, not integer
or boolean bounds")));
   (if not (t1 = t3) then raise (Error("Bad FOR loop, can not assign to iterator")));
   let (st, new_env) = get_env_for_stmt env s in
   (SFor((get_sexpr_for_expr env e1), (get_sexpr_for_expr env e2), (get_sexpr_for_expr env
e3), st), new_env)
  | While(e, s) ->
   let t = get_type_for_expr env e in
   (if not (t = Boolean) then raise (Error("Bad WHILE loop, not boolean bounds")));
   let (st, new_env) = get_env_for_stmt env s in (SWhile((get_sexpr_for_expr env e), st),
new_env)
  | Print(e) ->
   let _ = get_type_for_expr env e in (SPrint(get_sexpr_for_expr env e), env)
  | NoElseMark -> (SNoElseMark, env)
  | VarDecl(i) ->
   let (si, new_env) = add_init_var env i in (SVarDecl(SInit(i)), new_env)
  | Draw(e) -> let _ = get_type_for_expr env e in (SDraw(get_sexpr_for_expr env e), env)

(* Add all global vars and functions to our environment *)
let initialize_globals env constr = match constr with
  | GlobalVar(i) ->
    let (_, new_env) = add_init_var env i in
    new_env
  | FuncDef(f) ->
    let new_env = add_func_to_env env f in
    new_env

(* Check that return statements are in order, etc. *)
let check_final_env env =
  (if (false = env.return_seen && env.return_type <> Void) then
   raise (Error("Missing Return Statement")));
  true

(* Semantic checking on a function *)
let check_funcs env ct = match ct with
  | GlobalVar(v) ->
```

```
   let (t, _) = add_init_var env v in
   SGlobalVar(SInit(v))
 | FuncDef(f) ->
   let new_variables = List.fold_left (fun a vs -> (get_varsig_name_type vs)::a) [] f.formals in
   let new_var_scope = {parent = Some(env.var_scope); variables = new_variables} in
   let new_env = {return_type = f.return; return_seen = false; var_scope = new_var_scope;
fun_scope = env.fun_scope} in
   let (stbody, final_env) = get_env_for_stmt new_env f.body in
   let _ = check_final_env final_env in
   let sfunc = {sreturn = f.return; sfname = f.fname; sformals = f.formals; sbody = stbody} in
   SFuncDef(sfunc)

(* Empty initializations for tables/environments *)
let empty_symbol_table = {parent = None; variables = []}
let empty_function_table = {functions = []}
let empty_env = {return_type = Void; return_seen = false; var_scope = empty_symbol_table;
fun_scope = empty_function_table}


(* MAIN RUNNER FUNCTION *)
let translate prog =
 let env = List.fold_left (fun env ct -> initialize_globals env ct) empty_env prog in
 let res = List.map (fun ct -> check_funcs env ct) prog in
 res
```

8.1.9 translate_test.ml

```
open Ast
open Sast

let string_of_type = function
  | Int -> "TYPE_INT"
  | Point -> "TYPE_POINT"
  | Line -> "TYPE_LINE"
  | Curve -> "TYPE_CURVE"
  | Ellipse -> "TYPE_ELLIPSE"
  (*| LAYER -> "TYPE_LAYER"*)
  | String -> "TYPE_STRING"
  | Void -> "TYPE_VOID"
  | Boolean -> "TYPE_BOOLEAN"


let string_of_binop = function
  Add -> "ADD"
 | Sub -> "SUB"
 | Mult -> "MULT"
 | Div -> "DIV"
 | Equal -> "EQUAL"
 | Neq -> "NEQ"
 | Less-> "LT"
 | Leq -> "LTE"
 | Greater -> "GT"
 | Geq -> "GTE"

let rec string_of_expr = function
   IntLiteral(l) -> "EXPR_LITERAL:[LITERAL_INT:[" ^ (string_of_int l) ^ "]]"
  | StringLiteral(l) -> "EXPR_LITERAL:[LITERAL_STRING:[" ^ l ^ "]]"
  | Id(s) -> "EXPR_ID:[" ^ s ^ "]"
  (*| Dotop(s, op, e) -> "EXPR_DOTOP[" ^ s ^ "," ^ string_of_binop op ^ "," string_of_expr e
^"]"*)
  | GetX(e) -> "EXPR_GETX[" ^ string_of_expr e ^ "]"
  | GetY(e) -> "EXPR_GETX[" ^ string_of_expr e ^ "]"
  | MPoint(e1, e2) -> "EXPR_POINT[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^"]"
  | MLine(e1, e2, e3, e4) -> "EXPR_LINE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^ "," ^
string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MEllipse(e1, e2, e3, e4) -> "EXPR_ELLIPSE[" ^ string_of_expr e1^ "," ^ string_of_expr e2 ^
"," ^ string_of_expr e3 ^ "," ^ string_of_expr e4 ^"]"
  | MCurve(e1, e2, e3, e4, e5, e6) -> "EXPR_CURVE[" ^ string_of_expr e1^ "," ^ string_of_expr
e2 ^"," ^string_of_expr e3^ "," ^ string_of_expr e4 ^string_of_expr e5^ "," ^ string_of_expr e6
^ "]"
  (*possibly layer*)
  | Binop(e1, op, e2) -> "EXPR_BINOP:[" ^ string_of_expr e1 ^ "," ^ string_of_binop op ^ "," ^
string_of_expr e2 ^ "]"
  | Assign(e1, e2) -> "EXPR_ASSIGN:[" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ "]"
  | Call(f, el) ->
     "EXPR_CALL:[" ^ f ^ ",LIST:[" ^ String.concat ", " (List.map string_of_expr el) ^ "]]"

let string_of_sexpr = function
   ExprType(e, t) -> "(EXPR_TYPE:" ^ string_of_type t ^ ")" ^ string_of_expr e

let string_of_init = function
   NullInit(t, s) -> "INIT_NULL:[" ^ string_of_type t ^ "," ^ s ^ "]"
  | VarInit(t,s,e) -> "INIT_VAR:[" ^ string_of_type t ^ "," ^ s ^ "," ^ (string_of_expr e) ^ "]"

let string_of_sinit = function
   SInit(i) ->  string_of_init i


let rec string_of_sstmt = function
   SBlock(stmts) ->
     "STMT_BLOCK:[LIST:[" ^ String.concat "," (List.map string_of_sstmt stmts) ^ "]]"
  | SExpr(expr) -> "STMT_EXPR:[" ^ string_of_sexpr expr ^ "]";
  | SReturn(expr) -> "STMT_RETURN:[" ^ string_of_sexpr expr ^ "]";
  | SPrint(expr) -> "STMT_PRINT:[" ^ string_of_sexpr expr ^ "]";
  | SIf(e, s, SBlock([])) -> "STMT_IF_NOELSE:[" ^ string_of_sexpr e ^ "," ^ string_of_sstmt s ^
"]"
  | SIf(e, s1, s2) ->  "STMT_IF:[" ^ string_of_sexpr e ^ "," ^
     string_of_sstmt s1 ^ "," ^ string_of_sstmt s2 ^ "]"
  | SFor(e1, e2, e3, s) ->
     "STMT_FOR:[" ^ string_of_sexpr e1  ^ "," ^ string_of_sexpr e2 ^ "," ^
```

```
    string_of_sexpr e3  ^ "," ^ string_of_sstmt s ^ "]"
  | SWhile(e, s) -> "STMT_WHILE:[" ^ string_of_sexpr e ^ "," ^ string_of_sstmt s ^ "]"
  | SNoElseMark -> "STMT_NOELSEMARK"
  | SVarDecl(s)->string_of_sinit s
  | SDraw(e) -> "STMT_DRAW:[" ^ string_of_sexpr e ^ "]"

let string_of_varSig = function
  VarSig(t, s) -> "VARSIG:[" ^ string_of_type t ^ "," ^ s ^ "]"

let string_of_sfunc_decl fdecl =
  "FUNCDECL:[FUNTYPE[" ^ string_of_type fdecl.sreturn ^ "],FUNNAME[" ^ fdecl.sfname ^
"],LIST:[" ^
  String.concat "," (List.map string_of_varSig fdecl.sformals) ^ "],LIST:[" ^
  string_of_sstmt fdecl.sbody ^ "]]"

let string_of_sconstruct = function
   SGlobalVar(i) -> "CONSTRUCT_GLOBALVAR:[" ^ (string_of_sinit i) ^ "]"
  | SFuncDef(f) -> "CONSTRUCT_FUNCDEF:[" ^ (string_of_sfunc_decl f) ^ "]"

let string_of_sprogram sconstruct_list =
  "PROGRAM:[LIST:[" ^ (String.concat "," (List.map string_of_sconstruct sconstruct_list)) ^
"]]\n"
```

8.1.10 MDraw.ml

```
(* mdraw.ml -- Main runner file *)

open Parser
open Scanner

type action =  Scanner | Ast | Sast | Compile

let _ =
        (* Read argument and assign to "action" *)
  let action = if Array.length Sys.argv > 1 then
   List.assoc Sys.argv.(1) [ ("-a", Ast);
                  ("-s", Scanner);
                  ("-c", Compile);
                 ]
        (* User "Scanner" by default if no argument is provided *)
  else Scanner in

  match action with
   Scanner -> let lexbuf = Lexing.from_channel stdin in
        let rec loop token  =
          (match token with
            | EOF -> []
            | _ as t -> t::loop (Scanner.token lexbuf)) in
        let tokens = loop (Scanner.token lexbuf) in
        let output = Scanner_test.string_of_tokens tokens in
```

```
          print_string output
  | Ast ->   let lexbuf = Lexing.from_channel stdin in
          let abstract_syntax_tree = Parser.program Scanner.token lexbuf in
          let output = Parser_test.string_of_program abstract_syntax_tree in
          print_string output
  | Sast ->
          let lexbuf = Lexing.from_channel stdin in
          let abstract_syntax_tree = Parser.program Scanner.token lexbuf in
          let sastree = Translate.translate abstract_syntax_tree in
          let output = Translate_test.string_of_sprogram sastree in
          print_string output
  | Compile ->
          let lexbuf = Lexing.from_channel stdin in
          let abstract_syntax_tree = Parser.program Scanner.token lexbuf in
          let sastree = Translate.translate abstract_syntax_tree in
          let output = Compile.string_of_class (Sys.argv.(2)) sastree in
          print_string output
```

8.1.11 Makefile

```
OBJS = parser.cmo scanner.cmo scanner_test.cmo parser_test.cmo translate.cmo
translate_test.cmo compile.cmo MDraw.cmo

taml : $(OBJS)
        ocamlc -o ../MDraw $(OBJS)

scanner.ml : scanner.mll
        ocamllex scanner.mll

parser.ml parser.mli : parser.mly
        ocamlyacc parser.mly

%.cmo : %.ml
        ocamlc -c $<

%.cmi : %.mli
        ocamlc -c $<

.PHONY : clean

clean :

        rm -f scanner.ml scanner.mli scanner_test.mli \

        parser.ml parser.mli parser_test.mli translate.mli \

        translate_test.mli compile.mli MDraw.mli \

        test_results.out \
        *.cmo *.cmi *.out *.diff *.annot

# Generated by ocamldep *.ml *.mli
jpp.cmo:  ast.cmi
jpp.cmx:  ast.cmi
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
parser_utils.cmo:  ast.cmi
parser_utils.cmx:  ast.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
scanner_test.cmo: scanner.cmo parser.cmi
scanner_test.cmx: scanner.cmx parser.cmx
semantic_check.cmo:  ast.cmi
semantic_check.cmx:  ast.cmi
MDraw.cmo: translate_test.cmo translate.cmo scanner_test.cmo scanner.cmo \
   parser_test.cmo parser.cmi compile.cmo
MDraw.cmx: translate_test.cmx translate.cmx scanner_test.cmx scanner.cmx \
   parser_test.cmx parser.cmx compile.cmx
translate.cmo: sast.cmi ast.cmi
translate.cmx: sast.cmi ast.cmi
translate_test.cmo: sast.cmi ast.cmi
translate_test.cmx: sast.cmi ast.cmi
ast.cmi:
parser.cmi: ast.cmi
sast.cmi: ast.cmi
```

8.2 run.sh

```
# Based on your shell, may need to edit this path to Bourne shell
#!/usr/bin/sh
#dos2unix testall.sh                                        # For windows users
echo "Compiling: "$1
export CLASS_NAME=`echo $1 | sed 's/\.mdraw//'`
export JAVA_NAME=`echo $1 | sed 's/\.mdraw/\.java/'`
./MDraw -c $CLASS_NAME < $1 > java_lib/$JAVA_NAME
cd java_lib
javac $JAVA_NAME
java $CLASS_NAME
```

8.3 test

8.3.1 test_arith1.mdraw

```
void basic(){
print (2+3);
}
```

8.3.2 test_arith2.mdraw

```
void basic(){
print (1+2*3+4);
}
```

8.3.3 test_fib.mdraw

```
int fib(int x){
if (x<2) {return 1;}
return fib(x-1) + fib(x-2);
}
void basic(){
print (fib(0));
print (fib(1));
print (fib(2));
print (fib(3));
print (fib(4));
print (fib(5));
}
```

8.3.4 test_for.mdraw

```
void basic()
{
 int i;
 for (i = 0 ; i < 5 ; i = i + 1) {
  print(i);
 }
 print(42);
}
```

8.3.5 test_fun1.mdraw

```
int add(int a, int b)
{ return a+b; }
void basic()
{
  int a;
a = add(3,5);
print a;
}
```

8.3.6 test_fun2.mdraw

```
int fun(int x,int y)
{
  return 0;
}
void basic()
{
  int i;
  i = 1;
  i = fun(i=2, i=i+1);
 print(i);
}
```

8.3.7 test_gcd.mdraw

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

void basic()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
}
```

8.3.8 test_global.mdraw

```
int a;
int b;

int printa()
{
  print(a);
  return 0;
}

int printb()
{
  print(b);
  return 0;
}

int incab()
{
  a = a + 1;
  b = b + 1;
  return 0;
}

void basic()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
}
```

8.3.9 test_hello.mdraw

```
void basic()

{

  print(42);

  print(71);

  print(1);

}
```

8.3.10 test_if1.mdraw

```
void basic()
{
  if (3>2) print(42);
  print(17);
}
```

8.3.11 test_if2.mdraw

```
void basic()
```

```
{
  if (3<2) {print(42);} else {print 8;}
  print(17);
}
```

8.3.12 test_op.mdraw

```
void basic()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  print(1 == 2);
  print(1 == 1);
  print(99);
  print(1 != 2);
  print(1 != 1);
  print(99);
  print(1 < 2);
  print(2 < 1);
  print(99);
  print(1 <= 2);
  print(1 <= 1);
  print(2 <= 1);
  print(99);
  print(1 > 2);
  print(2 > 1);
  print(99);
  print(1 >= 2);
  print(1 >= 1);
  print(2 >= 1);
}
```

8.3.13 test_var.mdraw

```
void basic()
{
  int a;
  a = 42;
  print(a);
}
```

8.3.14 test_while.mdraw

```
void basic(){
int i;
i=10;
while(i>0){print i; i=i-1;}
}
```

8.3.15 test_point.mdraw

```
int x;
int y;
void paint(){
Point p;
p = MPoint(100,100);
x = p.getX();
y = p.getY();
}

void basic(){
print x;
print y;
}
```

8.3.16 test_line.mdraw

```
void paint(){
Line l;
l = MLine(100,100,200,200);
l.draw();
}

void basic(){}
```

8.3.17 test_curve.mdraw

```
void paint(){
Curve c;
c = MCurve(100,100,100,200,0,180);
c.draw();
}

void basic(){}
```

8.3.18 test_ellipse.mdraw

```
void paint(){
Ellipse e;
e = MEllipse(100,100,200,300);
e.draw();
}

void basic(){}
```