Columbia University

Programming Language & Translator

2012 Fall

Final Report

ALG – A Language for Geometry

Mengqi Zhang	mz2369
Qingye Jiang	qj2116
Shiyao Zhu	sz2395
Qiang Deng	qd2114
Ainur Rysbekova	ar3005

CONTENTS

1 INTRODUCTION

1.1 Background

This Language is supposed to be used by students who are studying geometry. Students may sometimes find it is abstract to indicate attributes of a concrete geometry figure and it is even more difficult to make clear the relationship between several figures. In order to help users have a better idea of geometry, we propose this "A Language for Geometry (ALG)" language and make the study for geometry more enjoyable!

1.2 Project Overview

Our Language is actually a programming language for geometry calculation. With our Language, a user can calculate certain attributes of a geometric figure and relationship between several figures in a convenient way, also the user can verify his/her speculations on geometry rules and witness how the other attributes will change as certain conditions change.

We keep most of the data types in C++ like int, float, char, etc. and define geometry types like point, line, ellipse and polygon, so that all 2-D figures can be composed from these four types. Also, we've defined geometric operators to help users indicate the relationship between two or more geometric figures. Therefore, it will be very easy for a user with limited programming skills to get the answer he/she wants to the geometric problem. Also, one can develop some complicated algorithms with the control structure provides in the language.

Our compiler does not actually compile the source code into binary code, it translates the ALG source code into correct C++ code, and we will get the executable file through the C++ compiler. The process to translate ALG code to C++ code requires the collaboration of scanner, parser, ast, semantic check, and code generation, and to make the process reliable, we also built numerous test cases. We will discuss the details of these in the following parts.

1.3 Language Features

- The Language will judge whether a value for a certain geometry type is valid or not.
- The same operator could be used by many data types.

- The same function could be used by various input arguments.
- ALG can display the input figures.
- ALG can move figures to different positions.
- ALG can calculate attributes of figures and indicate relations like intersect, tangent, etc.

2 LANGUAGE TUTORIAL

2.1 Getting Started with the Compiler

Before installing our compiler, configure your environment first as in Section 4-Environment (including download ‘pkgconfig’ as well as install ‘opencv’);

Also, ‘ALG Compiler’ will compiler the ALG source code into executable codes, with the help of c compiler ‘gcc’. You should also install ‘gcc’ in your system, ‘gcc47’ or above is required.

2.2 Installing and Compiling the Compiler

After the procedure above, you need to put our project folder ‘ALG’ into your file system. These are source files. You need to compile the ‘ALG Compiler’ first, and then use it to compile ALG programs.

To compile the ‘ALG Compiler’, go into the project folder, and then use the following commands to build:

make clean
make

After doing so, it will produce an executable file named ‘compiler’ (It will later be used by ‘compiler.sh’). Then use ‘compiler.sh’ to compile your ALG program with the following commands:

bash ./compiler.sh inputfilename outputfilename

Note: inputfilename is the file name of ALG program, and outputfilename is the name of the output executable file. In fact, although you only need to use the executable file named ‘outputfilename’, the compiler will generate intermediate c++ code in the file ‘output.c’.

After compiler your ALG program, you can now execute the executable file called ‘outputfilename’ with the following command:

./outputfilename

2.3 A first example of ALG program

2.3.1 ALG Program: (inter_area.alg)

```
1  def int main()
2  {
3          line l1;
4          line l2;
5          line l3;
6          point p1;
7          point p2;
8          point p3;
9          polygon poly;
10         l1=[[10;10],[20;20]];
11         l2=[[0;0],[20;1]];
12         l3=[[15;20],[20;0]];
13         Draw(l1);
14         Draw(l2);
15         Draw(l3);
16         (!display the information of the three lines!)
17         display(l1);
18         display(l2);
19         display(l3);
20         p1=l1^l2; !!p1 is the intersect point of l1 and l2
21         p2=l2^l3; !!p2 is the intersect point of l2 and l3
22         p3=l3^l1;       !!p3 is the intersect point of l1 and l3
23
24         poly=[p1,p2,p3]; !! poly is the intersect area of the three lines
25         print("the area of the intersection area is ");
26         print(Area(poly)); !!print the area of the intersection part of the three lines
27         print_newline();
28 }
```

This is an ALG program to calculate the area of the intersection part of three lines.
Line 3-9: Local variable declarations, here we defined line l1, l2, l3 as three lines, p1, p2, p3 as three points, and poly as a polygon;
Line 10-12: Assign values to the three lines;
Line 13-15: Draw the three lines defined above;
Line 17-19: Display the information of the three lines;
Line 20-22: Calculate the intersection point of each pairs of the three lines, and assign them to the three points defined above. The operator ‘^’ is used to calculate the intersection point of two lines.
Line 24: Construct a polygon with vertexes p1, p2, p3, and assign it to poly. The area of the polygon is the area of the intersection part of the three lines.
Line 26: Call built-in function Area to calculate the area of the intersection part, and print it out.

2.3.2 Compile the program

To compile the program, suppose you want to name the executable file as output_alg, then use the following command:

bash ./compiler.sh inter_area.alg output_alg

2.3.3 Running the program

Then use the following command to execute output_alg:

./output_alg

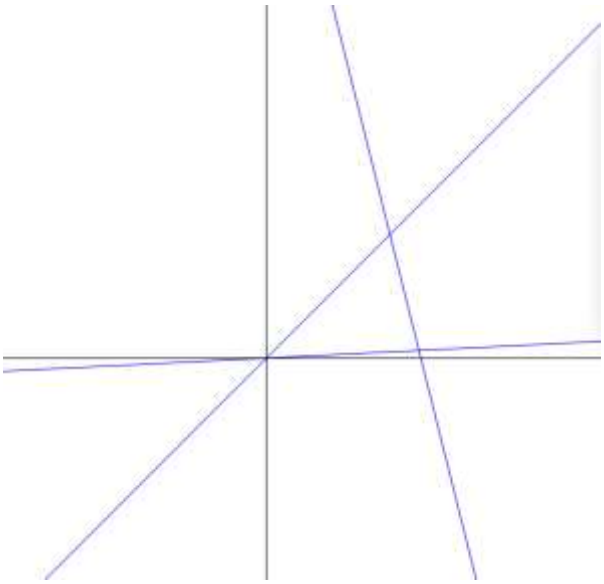
2.3.4 Result

The output of the above program is as below:
Console output:

```
./qiqi@qiqi-HP-Pavilion-dm1-Notebook-PC:~/workspace/PLT2/PLT_Project/ocaml$ ./ou
ut_alg
This is a line!
The first point is (1.000000 1.000000)
The second point is (2.000000 2.000000)
This is a line!
The first point is (0.000000 0.000000)
The second point is (2.000000 0.000000)
This is a line!
The first point is (2.000000 2.000000)
The second point is (2.000000 0.000000)
the area of the intersection area is 2
```

The output contains the information of the three lines defined in the program, and as well as the area of the intersection part of the three lines, which is 2.

Drawing output:



The drawing output shows the three lines defined in the ALG source file and their intersection part.

2.4 Additional Examples

2.4.1 Demonstrate Geometry Figures

This is an example to draw the figures in a window, and then you can clearly see the relationship of these figures.

2.4.1.1 ALG Program

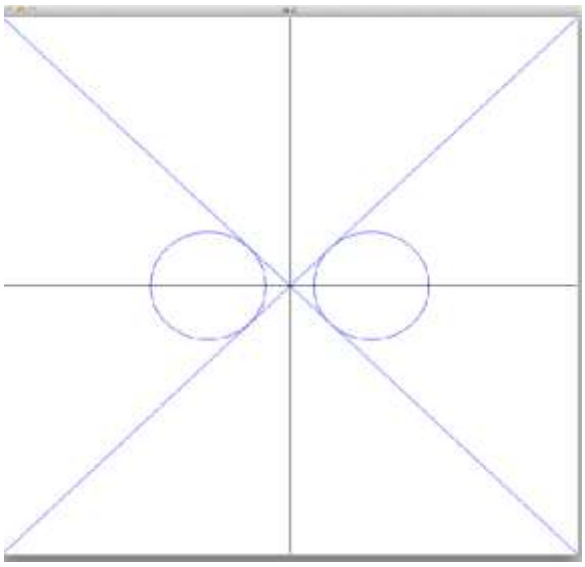
file | 16 lines (15 sloc) | 0.196 kb

```
1  def int main()
2  {
3          ellipse e1;
4          ellipse e2;
5          line l1;
6          line l2;
7          l1=[[1;1],[2;2]];
8          l2=[[1;-1],[2;-2]];
9          e1={[-14.14;0],10,10};
10         e2={[14.14;0],10,10};
11         Draw(l1);
12         Draw(l2);
13         Draw(e1);
14         Draw(e2);
15 }
```

Line 3-6: Define two ellipse e1 and e2, and two lines l1 and l2.
Line 7-10: Assign values to the four variables above. l1 is the line y=x; l2 is the line y=-x; e1 is an ellipse with center point (-14.14;0) , and axises 10, 10, which is in fact a circle; e2 is an ellipse with center point (-14.14;0), and axises 10,10, which is in fact a circle.
Line 11-14: Draw the four geometry figures in a new window.

2.4.1.2 Result

The result of the program above is as below:



2.4.2 Relationship between ellipses

This is an example to demonstrate the relationship of different ellipses.

2.4.2.1 ALG Program

```
1  def int main()
2  {
3          ellipse e1;
4          ellipse e2;
5          int i;
6          i=25;
7          e1={[0;0],10,10};
8          e2={[i;0],5,5};
9          Draw(e1);
10         while(i>=0)
11         {
12                 i=i-.5;
13                 Draw(e2);
14                 display(e2);
15                 print(e1|-e2);
16                 print_newline();
17                 Move(e2,-5,0);
18         }
19
20 }
```

In the above ALG program, we defined two ellipses e1 and e2; We make e1 immovable, and make e2 move by vector (-5,0), and then print out the positional relation between e1 and e2, as well as draw them.

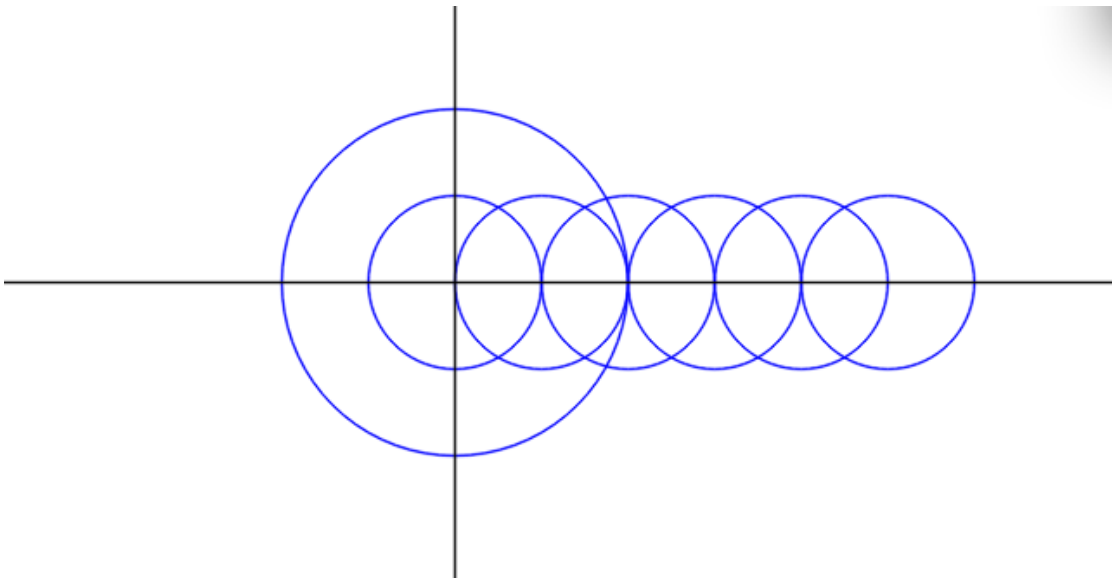
2.4.2.2 Result:

Console Output:

```
This is an ellipse!
Its focus point is (25.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
seperate
This is an ellipse!
Its focus point is (20.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
seperate
This is an ellipse!
Its focus point is (15.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
circumscribe
This is an ellipse!
Its focus point is (10.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
secant
This is an ellipse!
Its focus point is (5.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
inscribe
This is an ellipse!
Its focus point is (0.000000 0.000000)
Its major semi-axe's length is 5.000000, and its minor semi-axe's length is 5.000000
contain
```

The console output will print the information of all the ellipses e2, as well as their positional relationship.

Drawing Output:



Here are the figures of all the ellipses e2 and e1.

2.4.3 Relationship between Line and Ellipse

This demo is to show the positional relationship between line and ellipse.

2.4.3.1 ALG Program

```
def int main()
{
        ellipse e1;
        line l1;
        int i;
        i=50;
        e1={[0;0],10,10};
        l1=[[50;0],[50;1]];
        Draw(e1);
        while(i>=0)
        {
                Draw(l1);
                if((l1|-e1):="tangent")
                {
                        print("tangent");
                        print_newline();
                        display(l1);
                        done;
                }
                Move(l1,-5,0);
        }

}
```

In the above ALG program, we defined an ellipse e1 and a line l1. We want to find a line which is parallel to l1 and also tangent to e1. In the while loop, we move the l1 by vector (-5,0) in each iteration, and once we found that l1 is tangent to e1, we will quit. Meanwhile, we will print out the information of all the l1 during the while loop, as well as draw them.

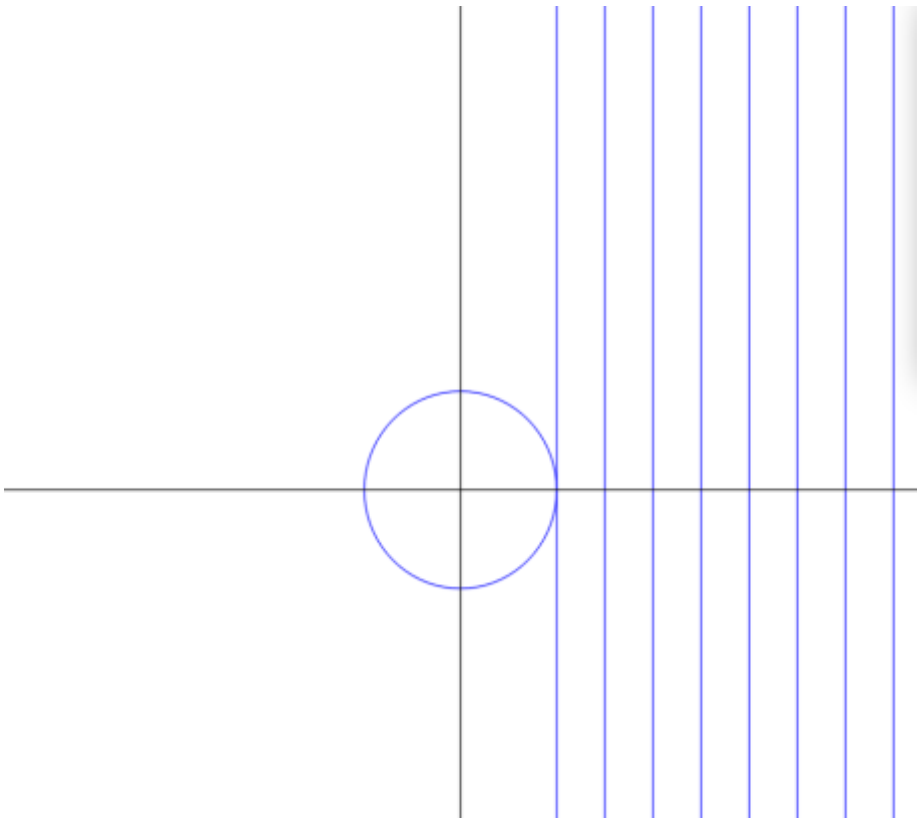
2.4.3.2 Result

Console Output:

```
qingyejiang$ ./output
tangent
This is a line!
The first point is (10.000000 0.000000)
The second point is (10.000000 1.000000)
```

The console output print out the information of the l1 which is tangent to e1 we found in our program.

Drawing Output:



In the Drawing Output, it prints out the e1, as well as all the lines in the while loop. And the leftmost line is the one we want to find.

2.4.4 Hexagon and Triangle

In this example, we calculate the area and perimeter of multiple hexagons originating from a single hexagon.

2.4.4.1 ALG Program

```
def int main()
{
        point p1;
        point p2;
        point p3;
        point p4;
        point p5;
        point p6;
        polygon poly1;
        polygon poly2;
        polygon poly3;
        polygon poly4;
        p1=[10;0];
        p2=[0;10];
        p3=[10;20];
        p4=[20;20];
        p5=[30;10];
        p6=[20;0];
        poly1=[p1,p2,p3];
        poly2=[p1,p2,p3,p4];
        poly3=[p1,p2,p3,p4,p5];
        poly4=[p1,p2,p3,p4,p5,p6];
        Draw(poly1);
        Draw(poly2);
        Draw(poly3);
        Draw(poly4);
        print("Areas ");
        print(Area(poly1));
        print_newline();
        print(Area(poly2));
        print_newline();
        print(Area(poly3));
        print_newline();
        print(Area(poly4));
        print_newline();
        print("Perimeters ");
        print(Perimeter(poly1));
        print_newline();
        print(Perimeter(poly2));
        print_newline();
        print(Perimeter(poly3));
        print_newline();
        print(Perimeter(poly4));
        print_newline();
}
```

The above program defines six points, and these six points make up a hexagon. We will use these six points to compose the four polygons, and print out the areas and perimeters of them.

2.4.4.2 Result

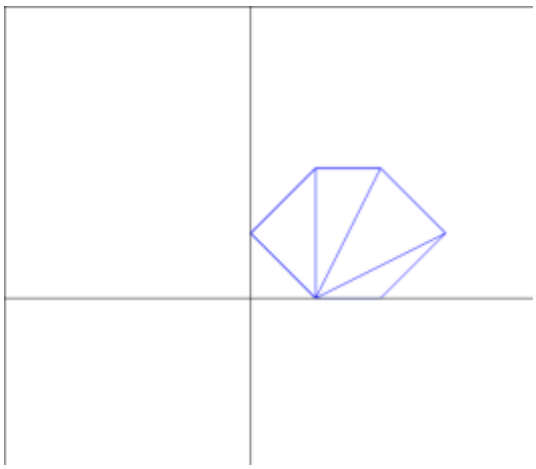
Console Output:

```
qingyejiang$ ./output
Areas
100
200
350
400
Perimeters
48.2843
60.645
74.7871
76.5685
```

The console output prints out the areas and perimeters of all the polygons we defined

above.

Drawing Output:



In the Drawing Output, it will print out the polygons we construct in the program.

2.4.5 Similar Triangles

2.4.5.1 ALG Program

```
def int main()
{
        polygon triangle;
        polygon triangle1;
        int i;
        int j;
        i=2;
        j=1;
        triangle=[[0;0],[i;0],[0;j]];
        triangle1=triangle;
        while(i<50)
        {
                Draw(triangle1);
                print(Area(triangle1));
                print_newline();
                i=i*2;
                j=j*2;
                triangle1=[[0;0],[i;0],[0;j]];

        }
        if(triangle1~triangle)
                {print("similar");
                print_newline();}
}
```

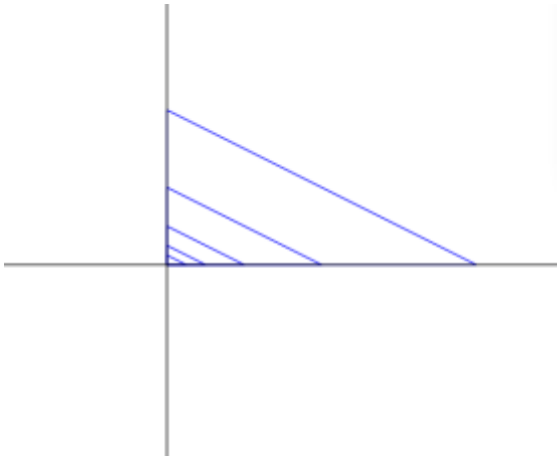
2.4.5.2 Result

Console Output:

```
qingyejiang$ ./output
1
4
16
64
256
similar
```

In the output, it prints out the area of a bunch of similar triangles. They have similarity ratio of 2.

Drawing Output:



In the Drawing Output, it will prints out all the similar triangles.

2.4.6 Triangles with the Same Height

In this demo, we will write a program to show how the area of an triangle changes along with one of its edges extending.

2.4.6.1 ALG Program

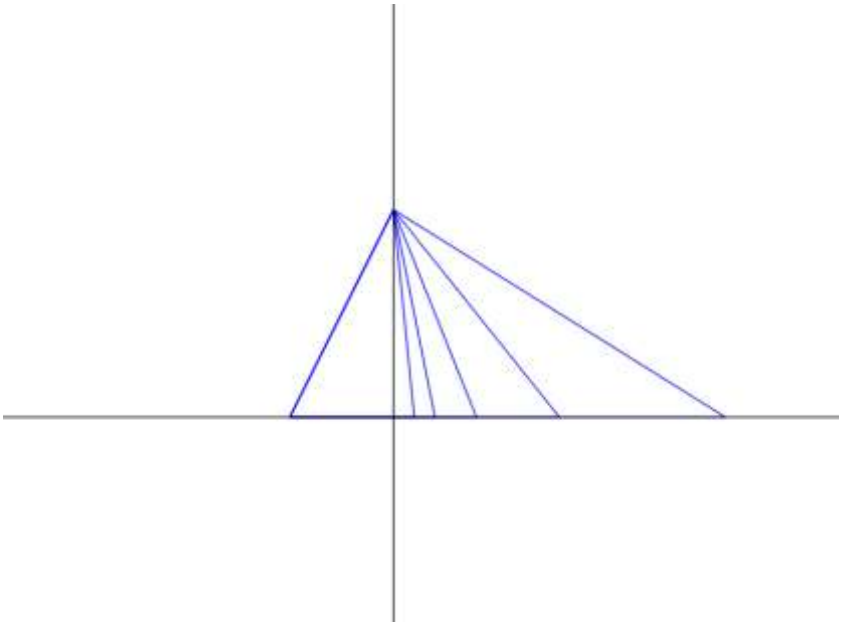
```
def int main()
{
        polygon triangle;
        int i;
        int j;
        int k;
        k=-10;
        j=20;
        i=2;
        triangle=[[k;0],[0;j],[i;0]];
        for (i=2;i<50;i=i*2)
        {
                triangle=[[k;0],[0;j],[i;0]];
                print(Area(triangle));
                print_newline();
                Draw(triangle);
        }
}
```

2.4.6.2 Result

Console Output:
In the following output, it prints out the the areas of all the triangles of the same height we defined in our ALG program.

```
qingyejiang$ ./output
120
140
180
260
420
```

Drawing Output:



3 LANGUAGE REFERENCE MANUAL

3.1 Introduction

A Language for Geometry (ALG) is a programming language for geometry calculation. With ALG, user can calculate certain attributes of geometric figures and relationship between them in a convenient way. Also user can verify his/her speculations on geometry rules and witness how the other attributes will change as certain conditions vary.

ALG programs are first compiled into "C++" modules and will be further compiled into machine binary by C Language compiler.

3.2 Lexical conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. Space, tabs and newlines are symbols of separating tokens. Every token should match the longest string that could comprise a token in the input buffer. For example, if there following two characters in the input buffer are "~" and "=", we will get the token from "~" rather than from "=".

3.2.1 Comments

Symbol "!!" is used to mark a single-line comment.
As for multi-line comments, "(!" marks the beginning of the comments, and "!)" means the end.
Here's an example:
! ! This is a single-line comment.
(! This Is a multi-line comment !)

3.2.2 Identifiers (Names)

An identifier is a sequence of and only of letters, digits, and underscores (_). An identifier must begin with a letter. ALG is a case sensitive language, so uppercase and lowercase are different for the compiler.

3.2.3 Keywords

The following terms are reserved as keywords for ALG:

int	point	for
float	line	while
string	polygon	if
ellipse	else	boolean
void	return	goon
done	def	

3.2.4 Constants

ALG has constants including Integer, float ,string and graph constants.

3.2.4.1 Integer constants

An integer constant is a sequence of digits without a decimal point.
For example:
int x = 15;

3.2.4.2 Float constants

A floating constant consists of an integer part, a decimal point, a fraction part, an e, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing. Every floating constant is taken to be double precision.
For instance, float number can be like:
0.125
1.25
123.33542000
While these numbers are illegal:
1..2 (More than one decimal)
.123 (integer part missing)

3.2.4.3 String constants

A string is a sequence of characters(‘a’-’z’, ‘A’-’Z’, ‘0’-’9’) surrounded by double quotes ‘‘ " ’’. "This is a string" defines a string in ALG. Two " must use together, none of them could be missing.

3.2.5 Graph constants

We use a new section to introduce graph constants because of their important to our language and the unique features.

3.2.5.1 point

Point is defined by a point [x, y] (x,y are floats or integers; x, y can also be variables)in the coordinate system. It is used to set a point.
Usage:
point a ;
a= [1;2];
a=[i;j];

3.2.5.2 line

Line is settled by two points. The two points should be on the line in order to define a line.
Usage:
line l1;
line l2;
point p1;
point p2;
l1 = [[0;1],[1;2]];
l2=[p1,p2]; !! the line can be defined by two points

3.2.5.3 ellipse

Ellipse is defined by a central point (not focus point) and two distance --a(the length of major semi-axex) and b(the length of minor semi-axes), and in the format as {point, a, b}(a and b are floats or integers).
Particularly, when a equals b, the ellipse is a circle.
Usage:
point p;
ell = {[1;4] 3, 5};
circleOne = {[0;0] 5, 5};
ell2={p,1,1};//defined with a point variable p

3.2.5.4 polygon

For polygon, points on each edge are used to define it. In order to clarify ambiguities, the points must be introduced clockwise. The polygon is defined as the format [point, point,point,...,point]
Usage:
triangle = [[0;0],[0;1],[4;1]];
square = [[0;0],[0;1],[1;1],[1;0]];
!!define a polygon with point variables, here either all the points are represented by variables, or all the points are represented by format [constant;constant];
triangle1=[p1,p2,p3];

3.3 Operator Characteristics

We have defined many operators to deal with Geometry Figures (e.g. "//", "|-") .These operators' operands cover a large range of objects. For example, "~=" could be applied to two operands of type ellipses or Polygons, and "|-" could be applied to two operands of type (line, ellipse), (ellipse, ellipse), etc. The details will be discussed in the section 5 of "Expression".

3.4 Program Structure

A ALG program consists of several global function declarations as well as global variable declarations (Details in section 7 of "Declarations"). Global variables and functions can be defined in any order, and global variables could be seen from all the global functions even if it is defined after the function declaration. However, functions can only see functions defined before it, or itself(recursive), those functions defined after it could not be seen. We don't allow nested function declarations, that is, a function could not be defined in another function.

3.5 Expressions

3.5.1 Primary Expressions

3.5.1.1 Identifiers

An identifier indicates a variable or function, and the example is like: rectangle1, Area.

3.5.1.2 Constant

Decimal integer, floating, boolean constants are all considered as primary expressions. The type of a constant is defined based on its form and value.

3.5.1.3 Graph-Constant

Point, line, ellipse, polygon constants are considered as primary expressions. Each geometry shape has its own patterns, like ellipse is {[1;2],1,2]}. We can get the type of a shape based on this pattern.

3.5.1.4 String

A string is a primary expression. It is a sequence of chars delimited by double quotes “ " ”.

3.5.1.5 Expression

A parenthesized expression is a primary expression. Its type and value are identical to those of the unadorned expression. The parenthesized expression is used to state the calculation priority clearly.

3.5.1.6 Primary-expression (expression-list)

A primary expression followed by a list of expressions indicated the appearance of a function call, and it is a primary expression. The primary-expression indicates which function (of which object‘s) is being adopted (the following expression list can be empty). Commas separate the expression-list in the parenthesis. The expression list includes all the parameters that are used in the calculation of the function. The returning type is decided by the function declaration.

When the function is called, a copy of all parameters is made, and the function may not change the real value of the actual parameters. However, the properties or values of an object can be changed by some built-in functions.

3.5.2 Geometry Operators

3.5.2.1 expression // expression

Both operands must be lines. The value of “l1 // l2” is true when l1 and l2 are parallel

and false when two lines are not parallel.

3.5.2.2 expression ^ expression

Both operands must be lines. The value of "s1 ^ s2" is the intersecting point of s1 and s2 when s1 and s2 are both lines, or void when they are parallel.

3.5.2.3 expression |- expression

Two operands can be line and ellipse or two ellipses.
1. line and ellipse:
For line l and ellipse e, l |- e will return one of the three strings: "secant", "separate", "tangent".
2. ellipse and ellipse:
For ellipse e1 and e2, e1 |- e2 will return one of the five strings: "secant", "separate", "contain", "inscribe", "circumscribe".

3.5.2.4 other expressions

expression =~ expression, expression ~ expression, expression~<expression, expression~>expression, expression~~<expression, expression~~>expression, expression<<expression, expression>>expression, expression<<= expression, expression>>=expression

Both the above operands should be figures. "=~" and "~" can only allow operands of two polygons, and the rest of them can allow operands of two polygons or two ellipses.

The operators ~=(two figures are the congruent), ~(two figures are similar), ~<(area is smaller than), ~>(area is greater than), ~~<(area is less than or equal to), ~~>(area is greater than or equal to), <<(perimeter is smaller than), >>(perimeter is greater than), <<=(perimeter is smaller or equal to), >>=(perimeter is greater or equal to) both yield boolean results.

3.5.3 Additive operators

The additive operators + and –. are binary and group left-to-right.

3.5.3.1 expression + expression

The result is the sum of the operands. If both expressions are integers, then the result is

an integer. If both expressions are float, then the result is a float. If one expression is float and the other one is integer, then the integer will be converted into a float and the result will be a float number. No other combinations are allowed.

3.5.3.2 expression – .expression

The result is the difference of the operands. If both expressions are integers, then the result is an integer. If both expressions are float, then the result is a float. If on expression is float and the other one is integer, then the integer will be converted into a float and the result will be float. No other combinations are allowed.

3.5.4 Multiplicative operators

The multiplicative operators *, / and % are binary and group left-to-right.

3.5.4.1 expression * expression

The result is the product of two expressions. If both expressions are integers, the result is an integer. If both expressions are float, the result is float. If one expression is integer and the other one is float, then the integer is converted into float and the result is float. No other combination is allowed.

3.5.4.2 expression/expression

The result is the division result. The same type considerations as for multiplication.

3.5.4.3 expression % expression

The result is the remainder from the division of the first expression by the second. Both operands must be integers.

3.5.5 Assignment operators

3.5.5.1 lvalue = expression

The value of object referred by lvalue will be replaced by the value of the expression. The operands' types can be divided into three situations. First, the operands are both numbers including integer and float. Second, the operands are both shapes including point, line, polygon, and ellipse. In the second case, the types of both operands should

be the same. Other combinations are not allowed. The expression's type will be converted to the type of the l-value if the operands' types are not the same in the first situation. Finally, the operands can be a variable of type string, and the value can a string constant.

3.5.6 string operator

While the string assignment operater is "=", there is a string comparison operator ":=" which judge whether the two strings are equal.

3.5.7 logical operator

"and " , "or ", "not" are three logical operators in ALG, which respectively means logic ADD, OR and NOT. While "and " and "or" are binary operator whose operands should be boolean, "not " is unary operator, whose operands should also be boolean.

3.5.8 arithmetic comparing operator

The "==" , "!=", "<", ">", "<=" and ">=" operators can are Boolean operators to judge whether two integers or floating numbers fulfill the specified relations. "==" is "equal to". "!="is "not equal to". "<" is "smaller than". ">" is "greater than". "<=" is "smaller or equal to". ">=" is "greater or equal to". In that case, both operands should be integers or floating numbers. If one expression is floating number and the other expression is integer, then the integer will be converted into a floating number and then the comparison will be done based on the converted value.

3.6 Built-in functions

There are 7 built-in functions in our language.

def void move(type-specifier a, float x, float y)
The move function takes three arguments, the first argument is the figure to be moved, which is of type point, line, polygon, ellipse. The second and third arguments are the move distances: x and y correspond to horizontal and vertical distances respectively. The return value is void.

def void print (type-specifier a)

The print function prints the argument it takes to the standard output. It's return type is void. It can print int, float and string.

def void print_newline()

The print_newline function prints a newline. It's return type is void. It doesn't take any parameters.

def void display (type-specifier a)

The display function displays the information of the argument to the standard output. The argument type is restricted to point, line, polygon and ellipse. The return type is void.

def void draw(type-specifier a)

The draw function can only accept parameters of type point, line, polygon, ellipse, and it will draw the figures of the specified variable in a window.

def float Area (type-specifier a)

The area function calculates the area of its argument. The type-specifier of the argument is either polygon or ellipse. The function returns the area of argument in type float.

def float Perimeter (type-specifier a)

The function perimeter calculates the perimeter of its argument. The type-specifier of the argument is either polygon or ellipse. The function returns the perimeter of the argument in type float.

Note: The entry function of our language should be named as "main", with return type int.

3.7 Declarations

The general format of a declaration is:

(def) type-specifier decl
The type-specifier declares type of the following decl and instructs compiler to interpret the decl. The identifier 'def' only applies to define a function, but not for variable declarations.

3.7.1 Type-specifier

Type-specifiers in ALG have similar types with C as well as its own types. They are:

int boolean
string point
float line
void ellipse
polygon

3.7.2 Function Declaration

3.7.2.1 Declarations

The type-specifier of a function specifies the type of return value from the function. The return type of the function can be each of the above types in 7.1. However, every function can only have one certain return value.

The decl part consists of a function name, arguments of the function and body of the function. A specific format of a function declaration is:

def type-specifier fname (farg-list) {fbody}

fname : is an identifier for the function which identifies the name of the function;
farg-list: contains zero or more arguments, and each argument has its own type-specifier and its name, different arguments are separated by ",";
fbody: implements the function, which consists of statements and variable declarations(see below). Particularly, in fbody, all the variable declarations appear before the statements. And no variable declarations are allowed among statements.

A common example of function declaration is:

```
def float AreaDifference (polygon a1, polygon a2)
{
    float i;
    float j;
    i=Area(a1);
        j=Area(a2);
    return (i-.j)
}
```

3.7.2.2 Recursive Functions

ALG allow recursive functions, which means inside a function definition, you can call the function itself.

3.7.3 Variable Declaration

Each variable has its own type as its type-specifier indicates. The decl part the variable declaration contain one identifier.

Here is the example of the format:
type-specifier identifier1;

Common examples of variable declaration are:

int a ;
polygon triangle1;

3.8 Statements

Note: <statement> is a general denotation of all kinds of statements in the following parts.

3.8.1 Expression statement

expression;

An expression statement is an expression followed by a semicolon. Operations like assigning value or calling functions are usually in this form. Semicolon after <statement> indicates the end of a statement.

Example:
a=3;
add(1, 2); !!add is a function with declaration ‘int add(int, int)’;

3.8.2 Block statement

{ <statement1>
<statement2>
<statement3>

}

Block statement includes one or more statements and always enclosed on both sides by curly brackets. Block statement is treated as one single statement.

3.8.3 Conditional statements

if (expression)

<block statement>

if (expression)
<block statement1>
else
<block statement2>

Conditional statements are similar to C's conditional statements.
In the first schema if expression returns a true value, the <block statement> is executed. Otherwise the block statement is ignored.
In the second schema if expression returns a true value the <block statement1> is executed. Otherwise the <block statement2> is executed.

3.8.4 Repetitive statements

while (expression)
<block statement>

for (expression1; expression2; expression3)
<block statement>

While statement runs as long as expression holds true.

For statement repeats until expression2 is true. expression1 holds initialization, expression2 contains condition and expression3 executes increment after each iteration.

3.8.5 Done statement

done;

Done statement ends the execution of a repetitive statement or a selection statement.

3.8.6 Goon Statement

goon;
Goon statement ends this round of iteration and continue executing the following rounds of iterations.

3.8.7 Return statement

return expression;

Return statement indicates an exit from the current function and returns a value to the calling function. The returned value is an expression following the return statement.

3.9 Scoping Rules

A function is only visible after or when it is declared and different functions have different names, while global variables are visible anywhere. This can be called a global scope. If a function is called before it is declared, then it will be regarded illegal.

For Example:
```
ellipse E1;
E1= {[1.5;2.0],3,4};   !! this is legal
line l1;
l1 = E1^E2;   !! this is illegal because E2 has not been declared
E2 = {[1,2],1.5,3.2};
```

4 PROJECT PLAN

4.1 Process

Our team was formed in late September. After that we meet at least three times a week, that is Monday, Wednesday and Saturday. The topics for each meeting varies as we made progress in the projects, but the procedure is standard. Every time in the meeting, we shall first check whether we have accomplished all the tasks that were assigned in the previous meetings and discuss how to solve the obstacles preventing us from moving forward. Then, we shall assign new tasks to each member of us according to the project timeline set before. Last, we will talk about the content of the PLT courses to make sure everyone is in pace with the lectures and has a good understanding of we are doing in the project.

We adopt Github to help us control the code version. Every member of the group contributes to the PLT repository once the code is accomplished and we do updates frequently to fix bugs and make optimizations. Since this is the first time we use adopt a SVN, we find it not that convenient to help share and control the code, because we usually encounter the problem of not being able to checkout the latest version and have to make clones almost every time. But still we forced us to do use github because it has become an essential skill in our future study and work. In the end of the project, we can work well with github and find it truly brings much convenience to our project. It's cool!

Our development strategy is an iterative process. To be specific, we will test our codes along the whole process in order to ensure we will not bring any independent bugs to the next step. The strategy works well in our project because this will narrow down the scope of our bug-checking. Though we have to modify previous files as we approach the finalized version, this strategy truly helps save a lot of time to concentrate on the newly-born bugs.

Besides conducting unit test for each of the file, our testing strategy mainly focuses on the semantic check phase and the final testing phase. Semantic check is trivial and we have to test whether all the legal input types will be admitted and illegal ones denied. It takes a lot of time and we need to be very careful about it. During the final testing phase we build two testing sets, one is for legal input and the other illegal input. More details about how we conduct these tests will be provided in the following corresponding chapter.

4.2 Programming Style

Even though the programming environments vary among the group due to different operating systems, still we attempt to maintain the programming style similar to that of MicroC, which is introduced in the class. Also we wrote comments for almost every function in the semantic check part, because the types to be checked are trivial and numerous, as for other parts, we keep the format neat and keep fewer comments for these files because they are relatively easy to understand.

4.3 Project Timeline

2012/9/20	Team formed
2012/9/25	Language defined
2012/9/28	Language proposal submitted
2012/10/1	Proposal modified according to TA's feedback
2012/10/31	Language Reference Manual submitted
2012/11/11	Scanner, Parser and AST completed
2012/11/16	Scanner, Parser and AST tested
2012/11/25	C++ code for built-in functions completed
2012/12/14	Semantic check completed and tested
2012/12/16	Code Generation completed
2012/12/18	Testing suit completed

4.4 Responsibility

Name	Code	Document
Shiyao Zhu (sz2395)	Scanner(ALG_Scanner.mll), Parser(ALG_Parser.mly), AST(AST.mli), Write tests for Semantic.ml, Write final tests for ALG with legal inputs	Ch 7 of LRM, Ch 1,4,8 of Final Report
Qingye Jiang (qj2116)	Semantic Analyzer (Semantics.ml), C++ code(alg.h), Write tests for Semantic.ml, Write	Ch 5 of LRM, Ch 5,6 of Final Report

	final tests for ALG with illegal inputs	
Mengqi Zhang (mz2369)	Code Generation: CodeGen.ml, Optimization for C++ code(alg.h), Write tests for CodeGen.ml, Write final tests for ALG with legal inputs, Supporting Files: makefile, compiler.ml,compiler.sh	Proposal, Ch 3,4 of LRM, Ch 2 of Final Report
Qiang Deng (qd2114)	C++ code (alg.h), Final tests with legal inputs	Ch 1,2 of LRM
Ainur Rysbekova (ar3005)	C++ code(alg.h), Final tests with illegal inputs	Ch 8,9 of LRM Ch 2 of Final Report

4.5 Environment

4.5.1 Language

We use O'Caml, C++, Shell Script and our own language ALG in our project to compose the files. Specifically, we use O'Caml to write the scanner, parser, ast, semantic check, and code generation parts; we use C++ to write the implementation of the built-in geometric functions; we use Shell Script to make the executable programs and ALG to write our testing cases.

4.5.2 Environment

Our computers run in different operating systems: Mac OS X, Ubuntu 12.04 and Windows Vista, thus we adopt different developing environments for the tasks, they are XCode, gcc, Eclipse, and gcc47 for Mac. Also, we use ocamllex for lexical analysis and ocamlyacc for the parsing. Moreover, to realize all the functions, we also need to configure OpenCV for the developing environment and get "pkgconfig" installed in Mac. (For how to configure OpenCV in Mac, Please consult http://opencv.willowgarage.com/wiki/Mac_OS_X_OpenCV_Port.)

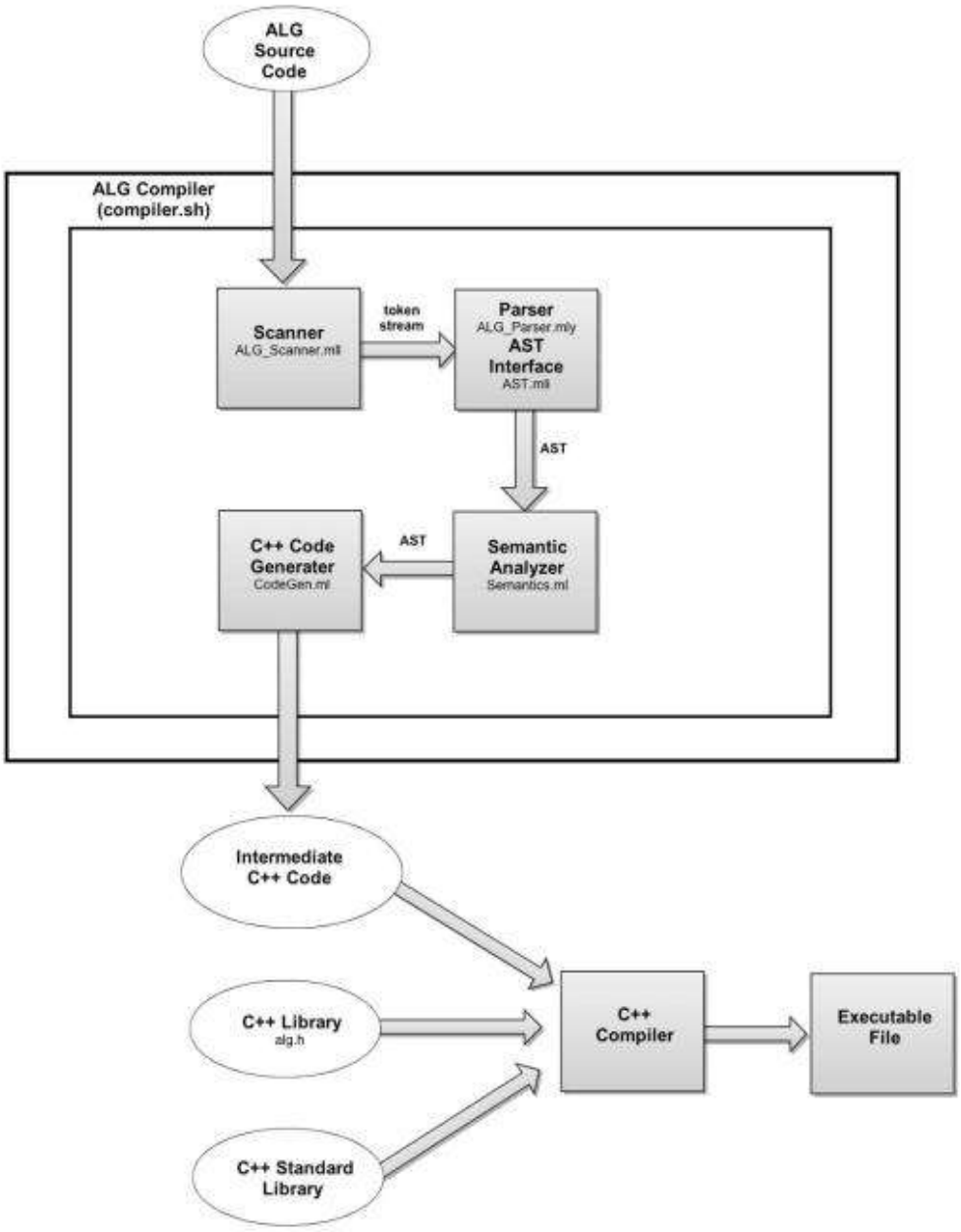
The compiler is primarily written in O'Caml and we write the alg.h header file in C++ to implement the back-end built-in functions like calculating the area and perimeter of a geometry figure, judging the position relationship of several figures and display the figures. All of these codes is written in editors like XCode, Vim and eclipse and ultimately compiled in gcc47 for Mac.

Also we adopt other tools in the project to make us get organized. To be specific, we use github to share and control different version of our codes and get everyone updated with the latest version. We use Google Doc to share documents and reading materials among group members and also established a Tencent Discuss Group talk about project problems remotely. All these tools make each of us informed about our progress and help our project work efficiently.

5 ARCHITECTURE DESIGN

5.1 Architectural Design Diagram

Here is a diagram to illustrate the main components and steps of ALG projects. Detailed explanation will be in other subsections of this section.
(Please consult the next page)



5.2 ALG Components

5.2.2 Scanner (ALG_Scanner.mll)

The scanner part will convert the raw ALG language into token stream, which will be used in the parser to compose the AST. It will also recognize the comments in the program so that the compiler will ignore them.

5.2.3 Parser and AST (ALG_Parser.mly & AST.mli)

In this part, we define our syntax and build up the AST. The input of token stream will finally be converted into two lists: a global variable declaration list, and a function declaration list. In the global variable declaration list, each global variable will be stored in a tuple whose form is (var_type * string), so that we can get access to this variable's type and its name. In the function declaration list, we store each function in a type who has the following properties: function name, function return type, function parameter list, function local variable list, and function body.

With the above two lists, we simulate an AST and in the following Semantic check part, we can recursively check the validity of AST structure.

5.2.4 Semantic Analyzer (Semantic.ml)

The semantic analyzer includes the important features like: check the duplication of variables in global/local variables, duplication of function names, validity of expression, validity of statements, validity of function declaration, etc. If there is any error in the ALG source code that has passed the parser section, it will be hijacked by the semantic analyzer and exception will be threw to inform the user which kind of error has been detected.

5.2.5 ALG C++ Library (alg.h)

We compose a C++ code to implement basic functions to define shape classes, calculate the relationship between pictures, or area or perimeter of shapes, draw pictures to illustrate their location relationship. This C++ library will directly be used by the generated C++ code from the following part.

5.2.6 C++ Code Generator (CodeGen.ml)

The C++ Code Generator will convert the AST (after semantic analyzer) into intermediate C++ code. The newly generated C++ code will be combined with the ALG C++ Library code and be given to the C++ compiler to execute.

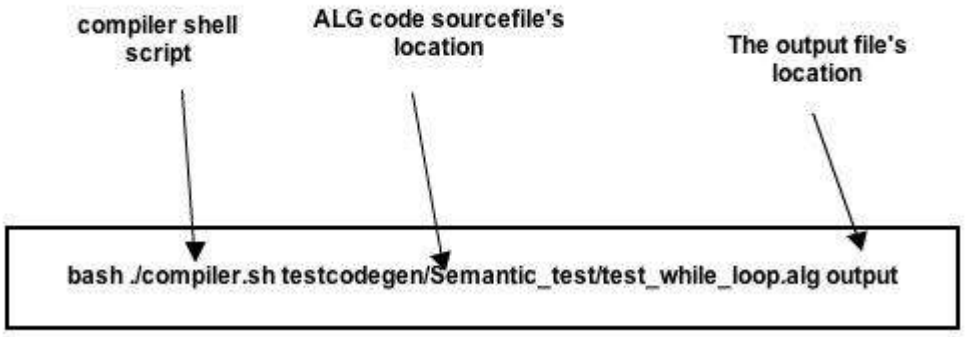
6 TEST

We adopted unit test method throughout our project in each components (Scanner, Parser, Semantic Analyser, Code Generator, C++ Library, Compiler). After the accomplishment of each component, we also did integrated testing to check whether the whole system fulfils the initial requirement, and also we went into the detail code to do white box testing to pick up bugs.

For example, in the coding process of Semantic analyzer, each low-level function will have a detailed goal. After writing one function, there should always be a test for it to see whether it will achieve our goal. One important function is "get_expr_type", which will get you the expression's type recursively. The author manually composed varieties of expressions from naive ones to complicated one which included sub-expressions inside. In this way, we can always be confident to use the old components when we try to develop new components.

After the integration of whole system, we write ALG language and give it to the compiler directly to see whether it passes the initial check and give the expected results. These test suites can be divided into three parts: Semantic Test, Basic Function Test, and Advanced Function Test.

You can just write sample ALG code in a .alg file, and run:



6.1 Semantic Test

We start from “errors” and see whether the semantic analyzer will stop those errors before the the AST is given to code generator. Typical errors are: Duplicated function name/ variable name/ parameter name, wrong number of parameters when using a function, non-defined function, wrong expression type in assign or function call, wrong statement form, lack of return statement in a function, no main function, etc. For each of these possible cases, an ALG code script is written and they are all stored in folder ./ocaml/testcodegen/Semantic_test.

A sample testing code is:

```
(! This codes include a function whose return type is wrong!)
(! The result is exception Failure("Unmatched return type with function's definition!") !)

int a;
def ellipse foo(int b, ellipse e1)
{
     b = 2;
     return b;
}

def int main()
{
     ellipse e2;
     ellipse e3;

     e3 = foo(2, e2);
     return 0;
}
```

6.2 Basic Function Test

We continue testing some basic functions of ALG language, including the variable declaration, variable value assignment, operators, built-in functions, for/while loop, drawing picture, etc. The corresponding test ALG code scripts are stored in four folders:

./ocaml/testcodegen/global
./ocaml/testcodegen/builtin
./ocaml/testcodegen/draw
./ocaml/testcodegen/basic_function

A sample code is:

```
def int main()
{
  point p1;
  line l1;
  ellipse e1;
  polygon poly1;
  p1=[0;0];
  l1=[[1;1],[2;2]];
  e1={[0;0],1,1};
  poly1=[[0;0],[1;1],[2;1],[1;0]];
  print("The original figures are as follows ");
  print_newline();
  display(p1);
  display(l1);
  display(e1);
  display(poly1);
  Move(p1,1.5,2.0);
  Move(l1,1.5,2.0);
  Move(e1,1.5,2.0);
  Move(poly1,1.5,2.0);
  print("After move they are ");
  print_newline();
  display(p1);
  display(l1);
  display(e1);
  display(poly1);
}
```

6.3 Advanced Function Test

After testing the basic functions and making sure they work well, we wanted to compose more complicated code to achieve more advanced functions. For example, given three lines, we want to get the area of the triangle that these three lines enclosed.

All this part's ALG code scripts are stored in the folder
./ocaml/testcodegen/algorithm

One sample code is:

```
def int main()
{
  line l1;
  line l2;
  line l3;
  point p1;
  point p2;
  point p3;
  polygon poly;
  l1=[[1;1],[2;2]];
  l2=[[0;0],[2;0]];
  l3=[[2;2],[2;0]];
  (!display the information of the three lines!)
  display(l1);
  display(l2);
  display(l3);
  p1=l1^l2; !!p1 is the intersect point of l1 and l2
  p2=l2^l3; !!p2 is the intersect point of l2 and l3
  p3=l3^l1; !!p3 is the intersect point of l1 and l3

  poly=[p1,p2,p3]; !! poly is the intersect area of the three lines
  print("the area of the intersection area is ");
  print(Area(poly)); !!print the area of the intersection part of the three lines
  print_newline();
}
```

7 LESSONS LEARNED

7.1 Qingye Jiang

It is such a pleasure to work with my teammates in this project. Building up a language compiler is a middle size project and required each teammate to understand almost all the components in this system. At first, I thought all I need to do is to understand how to code my own part (Semantic analyzer and C++ library). However, I found that without the knowledge of scanner, parser, and the following C++ code generator, I could not move even one step forward.

This project really helped me gain a deeper understanding of the general programming language implementation system, and finally I realized things like SLR table that we draw in the exam are not just logic games, but something really work. Coding with Ocaml for the Semantic Analyzer is convenient, even though sometimes bugs inside are not easy to find compared with C++ (maybe it is just because I am not a Ocaml pro). By viewing the code from last year programmers (especially thank to L-systems team), I concluded several kinds of common errors in our language that Semantic analyzer should specially focused on. It was also a challenge to compose the C++ library because in the geometry operations, for there were a lot of special

cases that I need to take care of, and it was a good chance for me to review high school math knowledge.

Mengqi and Shiyao did a wonderful job in finishing the scanner and parser early, and their robust code made it very comfortable for me to do some unit tests on my semantic analyzer. They also taught me how to use git, and it is efficient to work with this version control system when we worked separately. Thanks a lot to Professor Stephen Edwards and all my teammates for this wonderful class.

7.2 Mengqi Zhang

The course of PLT brings much to me during my first semester in Columbia. From the course, I learned a lot about the working principles of compiler as well as a magical functional language O'Caml. Also, I have a very nice experience collaborating with my teammates on our projects.

As for the knowledge with compiler, I learned a lot including the working principles of scanner, parser, semantic analyzer and code generator. I am always confused about how a C or Java program can be translated into executable codes, although I know how the machine codes works. And this course gives me exactly what I want to know. After I learned how the scanner transforms program elements into a stream of tokens, and how the parser will transform the tokens into AST, and how the semantic analyzer checks the validity of the AST, as well as how the Code Generator transforms the AST into executable codes, I grasp a clear conception about how the compiler works. The homework about automation and parsing methods really helps much in understanding what is going on when the compiler is working.

Meanwhile, this course is also about O'Caml. O'Caml is really a fantastic functional language. With O'Caml, we finally finished our project. Maybe at first glance it is terrible to look at the syntax of O'Caml, but it is really powerful to let us complete our project with little lines of codes. O'Caml can make us think in a very different way as compared to when we use Java or c. Also, it provides the many library functions which make our life easier.

Finally, the project makes me learn a lot of things. To understand the knowledge is one thing, and to implement it is another thing. I implemented the Code Generation part of the compiler in O'Caml, and I really encounter many problems and bugs during the process. Some of the problems can only solve upon that my teammate should change their part in Parser or Scanner process, and I should collaborate well with my teammates to achieve our goal. And this project really makes me more confident, we can solve problems with O'Caml, what else we can't do!

7.3 Shiyao Zhu

I know recursive functions before, but I understand them now. O'Caml is strange at first glance but it's a neat language with the power beyond your imagination. Almost every function can be recursive in the O'Caml world and this enables us to write "so little codes that can do so much". I've never seen functions like fold_left before, and it is brain draining to compose my own fold_left codes, but the feeling is great once I get familiar with it!

We say it's hard to find the beauty in a familiar place in Chinese, and so it is for coding. I only care about the codes but seldom the compiler in the past. It seems "nature" for compiler to generate the executable codes. But now I've learned the beauty of the systematic approach of how compilers work. Also the more obstacles I encountered, the more beauty I find. I'm lucky enough to come across several Shift/Reduce problems (set your mind, I've fixed them) in the earlier versions of my parser, and this is the right chance to adopt what we've learned in class to check the bugs in our project. It's terrific to synchronize our project with the lectures!

Tests should never be lower estimated. Even though we divided the project into several parts in the beginning, it's testing that links us together and force us to have a comprehensive understanding of all the stuff. I'm not the author of semantic part, but by semantic check testing, I got to know how the part is composed and how it cooperates with the others. Also, testing is not only about finding bugs, it can also provoke great ideas that could be added in our language.

Finally, as the captain of our ALG team, I'm grateful to work with our smart and hardworking teammates. If I may give some advice to this course, I think the name for this course should be *PLT & O'caml*, because you take one but actually get both!

7.4 Qiang Deng

In this course, firstly I learned how the language processor works. It's very interesting to know how it takes everything in and automatically generate executable code. This helps me build up a new perspective of programming languages.

Also, how to handle and process language is amazing. For example, translate tokens with Nondeterministic Finite Automata and Deterministic Finite Automata, build up LR(0) and apply rules to shift and reduce. I am totally shocked by how genius could those people be to develop these rules.

Secondly, O'Caml is weird. You have to give up what you previously knew about

programming and learn a new one, especially when there are countless loops and recursive links. It's kind of brain damaging to twist normal thinking to O'Caml thinking at the first time, (in fact I don't fully trust O'Caml at the beginning because it handles things in very few codes that I think it's unsafe) but it turns out to be a very useful language then.

Third, Professor Edwards' lectures are very helpful. They are well prepared and excellently organized, also full of fun.

Finally, working with great team members will be great fun. I'm very happy to work with my talented teammates. We actively set up meetings to discuss our project in which we enjoy a lot. We learned to use github to share and communicate code. We not only developed the project together, but also had fun together.

7.5 Ainur Rysbekova

By taking the PLT course I learned the structure of the compiler and gained experience with functional language like OCaml, which was quite different from other languages I learned before. The hardest part of this project was to learn the functional language, especially when you didn't have a lot of experience in programming. This project was very helpful for me in terms of understanding how to design and develop your own language. But the most important part of this project was that I significantly improved my programming skills and I learned a huge amount of new programming concepts that I didn't know before. To write complier was much much difficult than I thought before. Studying OCaml was very painful, but finally rewarding. In addition, I would like to note that the amount of knowledge I got from this course was very high. I really enjoyed learning fundamentals of translation and techniques used in translation and compiling. I really appreciate that I had so great team members. Working together was very efficient and collaborating with my teammates has been a good learning experience in terms how to plan your time and work for achieving a common goal. Each person brought different perspectives to a project what made our project enjoyable and feasible.

7.6 Advice for future group

Every previous team would suggest starting early and so do we. But you don't need to rush to the codes, please consult your TA once submitted the Proposal and LRM for detailed feedback, because good design will save you whole bunch of time.

Do invite all the team member to the meeting or you will spend extra time explaining everything to the person who missed the last meeting. It hurts the efficiency.

In the earlier stage tasks should be divided to perform a parallel approach to the destination, however once most of the modules are finished, we suggest to do cross testing, so that everyone can have a comprehensive understanding of the whole project rather than limit his/her knowledge to the divided part. Also this method is good for finding bugs ignored by the original author.

As tech guys, you need to make full use of as much tools as possible. Tools like SVN, Google Docs, discuss groups, etc. will save you a lot of time and enable all group members to fully collaborate with each other. Though we do not keep a complete project log, but it's wise of you to do that. It's a good habit to keep everything neat and organized.

7.7 Special Thanks

Thanks Stephen for guiding us in fighting with the Dragon of compiler!
Thanks our TA Bonan Liu to give us feedbacks and instructions in our project!
Thanks authors of MicroC, LSystem for providing us with an outline of how the project would be like!

8 Appendix

```
*********************************************************************************
```

ALG_Scanner.mll Originally authored by Shiyao Zhu (sz2369)

```
*********************************************************************************
{
    open Printf
    open ALG_Parser
}

let digit = ['0'-'9']
let id = ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_']*
let string=['a'-'z' 'A'-'Z' '0'-'9' '_' ' ']
let ignore=[' ' '\t' '\r' '\n']

rule token=parse
| ignore {token lexbuf}
| "(!"  {comment lexbuf}
| "!!"  {comment1 lexbuf}

(*type key words*)
| "int" {INT_KEY}|"float" { FLOAT_KEY } |"string" {STRING_KEY}
| "array" {ARRAY_KEY}|"boolean" {BOOLEAN_KEY}|"void" {VOID_KEY}

(*geometry type key words*)
| "point" {POINT_KEY}|"line" {LINE_KEY}|"polygon" {POLYGON_KEY}|"ellipse"
{ELLIPSE_KEY}

(*control key words*)
| "for" {FOR}|"while" {WHILE}|"if" {IF}
| "else" {ELSE}|"elseif" {ELSEIF}|"case" {CASE}|"return" {RETURN}
| "switch" {SWITCH}|"goon" {CONTINUE}|"done" {BREAK}
| "def" {DEF}|"default" {DEFAULT}
| "and" {AND} |"not" {NOT}|"or" {OR}
| "\"" string* "\"" as str {String(str)}
| "+" {PLUS} |"-." {MINUS}
| id as identifier {ID(identifier)}
| "-"?digit+ as int_num {INT(int_of_string int_num)}
| "-"?(digit+'.'digit*)('e'['+' '-']?digit+)?
| "-"?digit+'e'['+' '-']?digit + as float_num {NUM(float_of_string float_num)}
```

```
(*punctuation*)
| "(" {LPAREN} | ")" {RPAREN} | "{" {LBRACE} | "}" {RBRACE}
| "[" {LBRACKET } | "]" {RBRACKET } |"," {COMMA } | ";" {SEMI}

(*Geometry operators*)
| "//" {PARA}|"^" {INTERS}|"|-" {RELAT}
| "==" {EE}|"!=" {NE}
| "<<=" {SSE}|">>=" {LLE}
| "<<" {SS}|">>" {LL}
| "<=" {SE}|">=" {LE}
| "<" {S}|">" {L}
| "~>" {TL}|"~<"{TS}|"~~>"{TTL}|"~~<"{TTS}|"~="{TE}
| "~" {T}
| ":=" {QE}

(*basic operators*)
| "*" {MUL}
| "/" {DIV}|"%" {PERC} |"=" {E}
| eof   {EOF}
| _ as char {raise (Failure("SCANNER:illegal input"^Char.escaped char))}

and comment =parse
| "!)" {token lexbuf}
| _     {comment lexbuf}

and comment1=parse
| "\n" {token lexbuf}
| _ {comment1 lexbuf}
```

****

ALG_Parser.mly Originally authored by Shiyao Zhu (sz2369)

****

```
%{
    open Printf
    open AST
%}
%token AND OR NOT
%token TL TS TTL TTS TE T QE
%token EOF
%token FOR LPAREN RPAREN COMMA LBRACKET RBRACKET LBRACE RBRACE SEMI
%token INT_KEY FLOAT_KEY ARRAY_KEY STRING_KEY VOID_KEY BOOLEAN_KEY
```

```
%token POINT_KEY LINE_KEY POLYGON_KEY ELLIPSE_KEY
%token FOR WHILE IF ELSE ELSEIF CASE RETURN SWITCH BREAK CONTINUE DEF DEFAULT
%token <string> ID
%token <float> NUM
%token <int> INT
%token <string> String
%token PARA INTERS RELAT EE NE  SE LE S L  E PLUS MINUS MUL DIV PERC SSE LLE
SS LL

%nonassoc NOELSE
%nonassoc ELSE

%right E
%left PLUS MINUS
%left MUL DIV PERC
%left AND OR
%nonassoc NOT
%left PARA INTERS RELAT TE SSE LLE  SS LL TL TS TTL TTS  T QE
%left EE NE SE LE S L


%start program
%type <AST.program> program


%% /* Grammar rules and actions follow */

program:
/* nothing */ { [], [] }
| var_decl program { ($1 :: fst $2), snd $2 }
| fdecl program { fst $2, ($1 :: snd $2) }

stmt_list:
| { [] }
| stmt stmt_list { $1 :: $2 }

stmt:
| expr SEMI {ExStmt($1)}
| RETURN expr SEMI { Return($2) }
| LBRACE stmt_list RBRACE { Block($2) }
| IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
| IF LPAREN expr RPAREN stmt ELSE stmt{ If($3, $5, $7) }
```

```
| FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt{ For($3, $5, $7, $9) }
| WHILE LPAREN expr RPAREN stmt{ While($3, $5)}
| BREAK SEMI {BREAK}
| CONTINUE SEMI {CONTINUE}

var_type:
| INT_KEY   {INT_TYPE}
| FLOAT_KEY  {FLOAT}
| ARRAY_KEY  {ARRAY}
| STRING_KEY {STRING}
| VOID_KEY  {VOID}
| BOOLEAN_KEY {BOOLEAN}
| POINT_KEY {POINT}
| LINE_KEY {LINE}
| POLYGON_KEY {POLYGON}
| ELLIPSE_KEY {ELLIPSE}

/*function declaration*/
formal_decl:
| var_type ID {$1,$2}

var_decl:
| var_type ID SEMI {$1,$2}

formal_list_opt:
| {[]}
| formal_list {$1}

formal_list:
| formal_decl  { [$1] }
| formal_decl COMMA formal_list { $1 :: $3 }

vdecl_list:
| {[]}
| var_decl vdecl_list  { $1 :: $2 }

fdecl:
| DEF var_type ID LPAREN formal_list_opt RPAREN LBRACE vdecl_list stmt_list
RBRACE {{ftype=$2;fname=$3;formal_list=$5;locals=$8;body=$9}}

/*end function declaration*/
```

```
/*ex:
| expr  {GeoEx($1)}
| expr  {Ex($1)}
| ID LPAREN actuals_opt RPAREN {Call($1,$3)}
*/

actuals_opt:
| { [] }
| actuals_list { $1 }

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

/*expression*/

/*expr:
| String  {String($1)}
| ellipse {EllipseEx($1)}
| polygon {PolygonEx($1)}
| line    {LineEx($1)}
| point   {PointEx($1)}
| expr INTERS expr {GeoBinop($1,INTERS,$3)}
| expr PARA expr {GeoBinop($1,PARA,$3)}
| expr RELAT expr {GeoBinop($1,RELAT,$3)}
| expr TE expr {GeoBinop($1,TE,$3)}
| expr SS expr {GeoBinop($1,SS,$3)}
| expr LL expr {GeoBinop($1,LL,$3)}
| expr SSE expr {GeoBinop($1,SSE,$3)}
| expr LLE expr {GeoBinop($1,LLE,$3)}
| expr EE expr {GeoBinop1($1,EE,$3)}
| expr NE expr {GeoBinop1($1,NE,$3)}
| expr SE expr {GeoBinop1($1,SE,$3)}
| expr LE expr {GeoBinop1($1,LE,$3)}
| expr L expr {GeoBinop1($1,L,$3)}
| expr S expr {GeoBinop1($1,S,$3)}
| ID E expr {GeoAssign($1,$3)}
| LPAREN expr RPAREN {$2}
| expr T expr {GeoBinop($1,T,$3)}
| expr TS expr {GeoBinop($1,TS,$3)}
| expr TL expr {GeoBinop($1,TL,$3)}
| expr TTS expr {GeoBinop($1,TTS,$3)}
```

```
| expr TTL expr {GeoBinop($1,TTL,$3)}
*/


expr:
| NUM {NUM($1)}
| INT {INT($1)}
| ID {ID($1)}
| String  {String($1)}
| point   {PointEx($1)}
| polygon {PolygonEx($1)}
| ellipse {EllipseEx($1)}
| line    {LineEx($1)}
| expr INTERS expr {Binop($1,INTERS,$3)}
| expr PARA expr {Binop($1,PARA,$3)}
| expr RELAT expr {Binop($1,RELAT,$3)}
| expr TE expr {Binop($1,TE,$3)}
| expr SS expr {Binop($1,SS,$3)}
| expr LL expr {Binop($1,LL,$3)}
| expr SSE expr {Binop($1,SSE,$3)}
| expr LLE expr {Binop($1,LLE,$3)}
| expr PLUS expr  {Binop($1,PLUS, $3)}
| expr MINUS expr {Binop($1,MINUS, $3)}
| expr MUL expr   {Binop($1,MUL, $3)}
| expr DIV expr   {Binop($1,DIV, $3)}
| expr PERC expr  {Binop($1,PERC, $3)}
| expr EE expr  {Binop($1,EE, $3)}
| expr NE expr  {Binop($1,NE, $3)}
| expr LE expr  {Binop($1,LE, $3)}
| expr SE expr  {Binop($1,SE, $3)}
| expr L expr  {Binop($1,L, $3)}
| expr S expr  {Binop($1,S, $3)}
| expr T expr {Binop($1,T,$3)}
| expr TS expr {Binop($1,TS,$3)}
| expr TL expr {Binop($1,TL,$3)}
| expr TTS expr {Binop($1,TTS,$3)}
| expr TTL expr {Binop($1,TTL,$3)}
| expr QE expr {Binop($1,QE,$3)}
| expr AND expr {Binop($1,AND,$3)}
| expr OR expr {Binop($1,OR,$3)}
| NOT expr {Not($2)}
| ID E expr {Assign($1,$3)}
```

```
| LPAREN expr RPAREN {$2}
| ID LPAREN actuals_opt RPAREN {Call($1,$3)}

ellipse:
| LBRACE point COMMA expr COMMA expr RBRACE  {Ellipse($2,$4,$6)}
| LBRACE ID COMMA expr COMMA expr RBRACE {EllipseID($2,$4,$6)}

polygon:
| LBRACKET points RBRACKET   {Polygon($2)}
| LBRACKET pointsid RBRACKET   {PolygonID($2)}

line:
| LBRACKET point COMMA point RBRACKET  {Line($2,$4)}
| LBRACKET ID COMMA ID RBRACKET  {LineID($2,$4)}

pointsid:
| ID COMMA ID COMMA ID {[$1;$3;$5]}
| ID COMMA pointsid          {$1::$3}

points:
| point COMMA point COMMA point {[$1;$3;$5]}
| point COMMA points          {$1::$3}

point:
| LBRACKET expr SEMI expr RBRACKET {Point($2,$4)}
/*expression*/
%%
```

****

AST.mli Originally authored by Shiyao Zhu (sz2369)

****

```
type op =
QE|EE|NE|SE|LE|S|L|PARA|INTERS|RELAT|SS|LL|SSE|LLE|PLUS|MINUS|MUL|DIV|PERC|
TL|TS|TTL|TTS|TE|T|AND|OR

type var_type=
| INT_TYPE
| FLOAT
| ARRAY
| STRING
| VOID
| BOOLEAN
```

```
| POINT
| LINE
| POLYGON
| ELLIPSE


type expr=
| NUM of float
| INT of int
| ID of string
| Binop of expr * op * expr
| Not of expr
| Assign of string * expr
| Call of string * expr list
| String of string
| PointEx of point
| LineEx of line
| PolygonEx of polygon
| EllipseEx of ellipse

and point=
| Point of expr * expr

and line=
| Line of point * point
| LineID of string * string

and polygon=
| Polygon of point list
| PolygonID of string list

and ellipse=
| Ellipse of point * expr * expr
| EllipseID of string * expr * expr

and stmt=
| ExStmt of expr
| Return of expr
| Block of stmt list
| If of expr * stmt * stmt
| For of expr * expr * expr * stmt
| While of expr * stmt
```

```
| BREAK
| CONTINUE

and fdecl=
{
    ftype:var_type;
    fname:string;
    formal_list: (var_type * string) list;
    locals: (var_type * string) list;
    body: stmt list
}

type program=((var_type * string) list) * (fdecl list)

**********************************************************************
```

Semantics.ml Originally authored by Qingye Jiang (qj2116)

```
**********************************************************************
open AST


type env = {
    mutable functions : fdecl list;
    variables : (var_type * string) list;
    }



(*this is a function used in other functions*)
(*It is used to test whether a function's name is equal to a string 'name'*)

let func_equal_name name = function
    | func -> func.fname = name

(*This function is to check whether a function's name has been defined more than
once*)

let fun_exist func env =
    let name = func.fname in
       try
              let _ = List.find (func_equal_name name) env.functions in
                  let e = "Function whose name is "^ name ^" has been defined
more than once" in
```

```
                        raise (Failure e)
            with Not_found -> false

(*This function is to check whether a function's name exist in the env*)
let func_name_exist name env = List.exists (func_equal_name name) env.functions

(*This function will directly give you the function object if you give its name*)
let return_func_given_name name env =
    try
          let result = List.find (func_equal_name name) env.functions in
                result
  with Not_found -> raise(Failure("Function "^ name ^ " has not been declared!"))

(*check whether a 'fpname' is in the formal parameter list of a function*)
let exist_formal_para func fpname =
    let check func fpname = List.exists (function (a,b) -> b = fpname)
func.formal_list in
      check func fpname

(*check whether a 'vname' is declared in the local variable list of a function*)

let exist_local_variable func vname =
    let check func vname = List.exists (function (a,b) -> b= vname) func.locals
in
      check func vname


(*check whether a 'vname' is declared in the global variable list*)

let exist_global_variable env vname =
    let check env vname = List.exists (function (a,b) -> b= vname) env.variables
in
      check env vname

(*get the type of a 'vname' in the global variable list*)

let get_global_variable_type env vname =
    try
        let fpara = List.find(function (_,b) -> b = vname) env.variables in
          let(c,_) = fpara in
              c
    with Not_found ->raise(Failure("Cannot find a variable called "^ vname ^
```

```
" in global variable list!"))

(*The following function will get you the type of a parameter in a function if
you give parameter's name*)

let get_para_type func fpname =
    try
          let para  =List.find (function (a,b) -> b = fpname) func.formal_list
in (*check whether fname is in  func's formal_list*)
              let (para_type,_) = para in
                   para_type
    with Not_found ->raise(Failure("In the function" ^ func.fname ^" does not
have the parameter "^ fpname))


(*The following function will get you the type of a local variable in a function
if you give variable's name*)

let get_var_type func vname =
    try
        let var  = List.find (function (a,b) -> b = vname) func.locals
in(*search in function's local variable list*)
          let (var_type,_) = var in
              var_type
  with Not_found -> raise(Failure("In the function"^ func.fname ^" there is not
a loval variable called " ^ vname ^ " defined!"))

(*A function to check whether a function has a parameter called fpname*)
let check_exist_para_in_fun func fpname = List.exists (function (_,b)->b =
fpname) func.formal_list

(*A function to check whether a function has a local parameter called vname*)
let check_exist_var_in_fun func vname = List.exists (function (_,b) -> b = vname)
func.locals



(*The function to get the type of a variable "name" to see whether it appears
in parameter or local variable list or global variable list*)

let get_type env func name =
    if check_exist_var_in_fun func name
```

```
          then get_var_type func name
    else
           if check_exist_para_in_fun func name
                  then get_para_type func name
              else
                  if exist_global_variable env name
                      then get_global_variable_type env name
                      else
                   raise(Failure("Variable" ^ name^" is used but not declared
in function "^ func.fname))


(*The function will check whether a variable "name" exists in a function's para
list or local list*)
let exist_id name func env= (check_exist_para_in_fun func name) or
(check_exist_var_in_fun func name) or (exist_global_variable env name)


(*The function will check whether a function name "fun_name" has a corresponding
function in the environment*)

let find_func func_name env =
    try
         let _ = List.find (func_equal_name func_name) env.functions in
             true
    with Not_found -> raise(Failure("Function "^ func_name ^" is not found in
the function defination list"))


(*This function will check whether a (var_type*string) has a para_name that
appears more than once in a function's parameter list*)
let count_fpara func = function (_,b)
  -> let f count (_,c) =
          if c=b then count+1
            else count
            in
              let count = List.fold_left f 0 func.formal_list in
                 if count > 1
                    then raise(Failure("Duplicate parameter in function " ^
func.fname))
                    else
                        count
```

```
(*This function will automatically check whether there is parameter duplication
in function defination*)

let check_fpara_duplicate func =
    List.map (count_fpara func) func.formal_list

(*This function will check whether a (var_type*string) has a var_name that
appears more than once in a function's local variable list*)
let count_var func = function (_,b)
  -> let f count (_,c) =
         if c=b then count+1
           else count
           in
             let count = List.fold_left f 0 func.locals in
                if count > 1
                   then raise(Failure("Duplicate parameter "^ b ^ " in function
" ^ func.fname))
                   else
                       count

(*This function will automatically check whether a function has local variable
duplication*)

let check_var_duplicate func =
    List.map (count_var func) func.locals

(*This function will get you the return type of a function if you give me the
fname*)
let get_func_return_type func_name env =
    try
         let func = List.find (func_equal_name func_name) env.functions in
             func.ftype
    with Not_found -> raise(Failure("Function "^ func_name ^" is not found in
the function defination list"))


(*This function will help check whether an ellipse's axis length is positive*)
let check_ellipse_axis expr =
    match expr with
    | NUM(a) ->if a>0.0 then true else raise(Failure("The axis length of an
```

```
ellipse mut be positive!"))
  | INT(a) ->if a>0 then true else raise(Failure("The axis length of an ellipse
mut be positive!"))
    | _ -> true

(*This function will return the type of an expression*)
(*env: the environment including the global variable list and function list*)
(*expr: the expression we are exploring*)
(*func:the function we are in now*)
let rec get_expr_type expr func env =
    match expr with
    | String(s) ->STRING
    | ID(s) -> get_type env func s (*we need to modify it later*)
    | INT(s) -> INT_TYPE

    | PointEx(Point(expr1,expr2))
       ->
              let t1 = get_expr_type expr1 func env and t2 = get_expr_type expr2
func env in
             begin
                   match t1,t2 with
                   | INT_TYPE, INT_TYPE -> POINT
                   | INT_TYPE, FLOAT -> POINT
                   | FLOAT,INT_TYPE -> POINT
                   | FLOAT, FLOAT ->POINT
                   | _, _ -> raise(Failure("Error in Point expression!"))
                   end
    | LineEx(Line(Point(expr1,expr2),Point(expr3,expr4)))
       -> let check_temp_expr temp_expr =
                let t = get_expr_type temp_expr func env (*get the type of one
expression*)
                in
                    begin
                    match t with
                      | INT_TYPE -> true
                      | FLOAT -> true (*the function check_temp_expr will
check whether an expression's type is INT_TYPE or FLOAT*)
                      | _ -> false
                       end
             in
                   if check_temp_expr expr1 && check_temp_expr expr2 &&
check_temp_expr expr3 && check_temp_expr expr4
```

```
                    then LINE
                    else raise(Failure("Error in Line expression"))

    | EllipseEx(Ellipse(Point(expr1,expr2),expr3,expr4)) ->
        let check_temp_expr temp_expr =
            let _check1 = check_ellipse_axis expr3 and _check2 =
check_ellipse_axis expr4 in
                let t = get_expr_type temp_expr func env (*get the type of one
expression*)
                in
                 begin
                   match t with
                     | INT_TYPE -> true
                     | FLOAT -> true (*the function check_temp_expr will
check whether an expression's type is INT_TYPE or FLOAT*)
                     | _ -> false
                     end
             in
                  if check_temp_expr expr1 && check_temp_expr expr2 &&
check_temp_expr expr3 && check_temp_expr expr4
                     then ELLIPSE
                     else raise(Failure("Error in Ellipse expression"))


    | PolygonEx(Polygon(l))
       -> let check_point_validation pp =
                     begin
                     match pp with

                     | Point(expr1,expr2) ->

                    let t1 =  get_expr_type expr1 func env  and t2 =
get_expr_type expr2 func env
                  in
                    begin
                     match t1,t2 with
                      | INT_TYPE,INT_TYPE ->true
                      | INT_TYPE,FLOAT ->true
                      | FLOAT,INT_TYPE -> true
                      | FLOAT,FLOAT -> true
                      | _,_ ->false
                     end
```

```
                    |_ -> raise(Failure("Do Point Check to a non-Point
object!"))
                    end(*end of definition of check_point_validation*)
                    in let final = List.for_all check_point_validation l in
                    if final then POLYGON else raise(Failure("Error in Polygon
expression!"))(*check whether each point is valid*)

    | PolygonEx(PolygonID(listid)) -> POLYGON
        | LineEx(LineID(str1,str2)) -> LINE
        | EllipseEx(EllipseID(str,expr1,expr2)) -> let type1  = get_expr_type
expr1 func env and type2 = get_expr_type expr2 func env in
          begin
            match type1 ,type2 with
            | INT_TYPE,FLOAT
            | FLOAT,INT_TYPE
            | INT_TYPE,INT_TYPE
            | FLOAT,FLOAT -> ELLIPSE
            | _,_ -> raise(Failure("Error in Ellipse expression"))
          end
    | NUM(s) ->FLOAT
        | Not(expr1) -> let type1 = get_expr_type expr1 func env in if type1 =
BOOLEAN then BOOLEAN else raise(Failure("The type of expression in Not operator
should be boolean!"))
    | Binop(expr1,op,expr2) -> let temp1 = get_expr_type expr1 func env and temp2
= get_expr_type expr2 func env
          in
                begin
                    match temp1,op,temp2 with
                                    | BOOLEAN, OR, BOOLEAN-> BOOLEAN
                                    | BOOLEAN, AND, BOOLEAN -> BOOLEAN
                    | BOOLEAN, EE, BOOLEAN -> BOOLEAN(*EE*)
                    | INT_TYPE, EE, INT_TYPE -> BOOLEAN
                    | INT_TYPE, EE, FLOAT -> BOOLEAN
                    | FLOAT, EE, INT_TYPE -> BOOLEAN
                    | FLOAT, EE, FLOAT -> BOOLEAN
                    | POLYGON,TE,POLYGON -> BOOLEAN
                    | ELLIPSE, TE, ELLIPSE -> BOOLEAN
                    | LINE, EE, LINE -> BOOLEAN
                    | POINT, EE, POINT -> BOOLEAN
                    | STRING, QE, STRING -> BOOLEAN
                    | BOOLEAN, NE, BOOLEAN->BOOLEAN      (*NE*)
                    | INT_TYPE, NE, INT_TYPE -> BOOLEAN
```

```
  | INT_TYPE, NE, FLOAT -> BOOLEAN
  | FLOAT, NE, INT_TYPE -> BOOLEAN
  | FLOAT, NE, FLOAT -> BOOLEAN
  | POLYGON,NE,POLYGON -> BOOLEAN
  | ELLIPSE, NE, ELLIPSE -> BOOLEAN
  | LINE, NE, LINE -> BOOLEAN
  | POINT, NE, POINT -> BOOLEAN
  | STRING, NE, STRING -> BOOLEAN
  | INT_TYPE, SE, INT_TYPE -> BOOLEAN   (*SE,LE,S,L*)
  | INT_TYPE, SE, FLOAT -> BOOLEAN
  | FLOAT, SE, INT_TYPE -> BOOLEAN
  | FLOAT, SE, FLOAT -> BOOLEAN
  | POLYGON, TTS, POLYGON -> BOOLEAN
  | POLYGON, TTS, ELLIPSE -> BOOLEAN
  | ELLIPSE, TTS, POLYGON -> BOOLEAN
  | ELLIPSE, TTS, ELLIPSE -> BOOLEAN
| INT_TYPE, LE, INT_TYPE -> BOOLEAN
  | INT_TYPE, LE, FLOAT -> BOOLEAN
  | FLOAT, LE, INT_TYPE -> BOOLEAN
  | FLOAT, LE, FLOAT -> BOOLEAN
  | POLYGON, TTL, POLYGON ->BOOLEAN
  | POLYGON, TTL, ELLIPSE -> BOOLEAN
  | ELLIPSE, TTL, POLYGON -> BOOLEAN
  | ELLIPSE, TTL, ELLIPSE -> BOOLEAN
  | INT_TYPE, S, INT_TYPE -> BOOLEAN
  | INT_TYPE, S, FLOAT -> BOOLEAN
  | FLOAT, S, INT_TYPE -> BOOLEAN
  | FLOAT, S, FLOAT -> BOOLEAN
  | POLYGON, TS, POLYGON -> BOOLEAN
  | POLYGON, TS, ELLIPSE -> BOOLEAN
  | ELLIPSE, TS, POLYGON -> BOOLEAN
  | ELLIPSE, TS, ELLIPSE -> BOOLEAN
  | INT_TYPE, L, INT_TYPE -> BOOLEAN
  | INT_TYPE, L, FLOAT -> BOOLEAN
  | FLOAT, L, INT_TYPE -> BOOLEAN
  | FLOAT, L, FLOAT -> BOOLEAN
  | POLYGON, TL, POLYGON -> BOOLEAN
  | POLYGON, TL, ELLIPSE -> BOOLEAN
  | ELLIPSE, TL, POLYGON -> BOOLEAN
  | ELLIPSE, TL, ELLIPSE -> BOOLEAN
  | LINE, PARA, LINE -> BOOLEAN(*PARA*)
  | LINE, INTERS, LINE -> POINT(*INTERS*)
```

```
(*| LINE, INTERS, ELLIPSE -> ARRAY*)
(*| ELLIPSE, INTERS, LINE -> ARRAY*)
| ELLIPSE, INTERS, ELLIPSE ->LINE
| LINE,RELAT, LINE -> STRING (*RELAT*)
| LINE, RELAT, ELLIPSE -> STRING
| ELLIPSE, RELAT, LINE -> STRING
| ELLIPSE, RELAT, ELLIPSE -> STRING
| POLYGON, TE, POLYGON -> BOOLEAN (*TE*)
| POLYGON, T, POLYGON -> BOOLEAN (*T*)
| POLYGON, SS, POLYGON -> BOOLEAN(*SS,LL,SSE,LLE*)
| POLYGON, SS, ELLIPSE -> BOOLEAN
| ELLIPSE, SS, POLYGON -> BOOLEAN
| ELLIPSE, SS, ELLIPSE -> BOOLEAN
| POLYGON, LL, POLYGON -> BOOLEAN
| POLYGON, LL, ELLIPSE -> BOOLEAN
| ELLIPSE, LL, POLYGON -> BOOLEAN
| ELLIPSE, LL, ELLIPSE -> BOOLEAN
| POLYGON, SSE, POLYGON -> BOOLEAN
| POLYGON, SSE, ELLIPSE -> BOOLEAN
| ELLIPSE, SSE, POLYGON -> BOOLEAN
| ELLIPSE, SSE, ELLIPSE -> BOOLEAN
| POLYGON, LLE, POLYGON -> BOOLEAN
| POLYGON, LLE, ELLIPSE -> BOOLEAN
| ELLIPSE, LLE, POLYGON -> BOOLEAN
| ELLIPSE, LLE, ELLIPSE -> BOOLEAN
| INT_TYPE, PLUS, INT_TYPE -> INT_TYPE
| INT_TYPE, PLUS, FLOAT -> FLOAT
| FLOAT, PLUS, INT_TYPE -> FLOAT
| FLOAT, PLUS, FLOAT -> FLOAT
| INT_TYPE, MINUS, INT_TYPE -> INT_TYPE
| INT_TYPE, MINUS, FLOAT -> FLOAT
| FLOAT, MINUS, INT_TYPE -> FLOAT
| FLOAT, MINUS, FLOAT -> FLOAT
| INT_TYPE, MUL, INT_TYPE -> INT_TYPE
| INT_TYPE, MUL, FLOAT -> FLOAT
| FLOAT, MUL, INT_TYPE -> FLOAT
| FLOAT, MUL, FLOAT -> FLOAT
| INT_TYPE, DIV, INT_TYPE -> FLOAT
| INT_TYPE, DIV, FLOAT -> FLOAT
| FLOAT, DIV, INT_TYPE -> FLOAT
| FLOAT, DIV, FLOAT -> FLOAT
| INT_TYPE, PERC, INT_TYPE ->INT_TYPE
```

```
                    | _,_,_ ->raise(Failure("Illegal type used in a binop
expression"))
                end(*end of binop consideration*)
            |Assign(id,expr) -> get_expr_type expr func env
            |Call(fname,expr) -> get_func_return_type fname env(*We want to
check the return type of the function fname*)
            |_ ->raise(Failure("An unexpected error occured!"))

(*The following function will judge whether an expression is assign or call*)

let is_assign_call func = function
    | Assign(_,_) ->true
    | Call(_,_) ->true
    | _ ->false


(*This function is to check whether when calling function fname with a parameter
list exprlist, it matches*)
(*all the required parameter type as claimed in the function declaration*)
(*fname: the function name being called*)
(*exprlist: the expression list which performs as the parameters when called*)
(*func: the function in which fname is called*)
(*env: the big enviornment*)

let check_func_paralist_type fname exprlist func1 env =
   let l = List.length exprlist in
        if fname = "display" && l= 1 then let expr1 = List.hd exprlist in let
v_type = get_expr_type expr1 func1  env in

        match v_type with
        | POINT -> 1
        | ELLIPSE -> 1
        | POLYGON -> 1
        | LINE -> 1
        | _ -> raise(Failure("Wrong usage in using display function by choosing
variable type "))

    else if fname = "Draw" && l= 1 then let expr1 = List.hd exprlist in let v_type
= get_expr_type expr1 func1  env in

        match v_type with
        | POINT -> 1
```

```
        | ELLIPSE -> 1
        | POLYGON -> 1
        | LINE -> 1
        | _ -> raise(Failure("Wrong usage in using draw function by choosing
variable type "))

       else if l=1 && ((fname = "Perimeter")||(fname = "Area")) then
       let expr1 = List.hd exprlist in let v_type = get_expr_type expr1 func1
env in

       match v_type with
      |ELLIPSE -> 1
      |POLYGON -> 1
      | _ -> raise(Failure("Wrongly use function "^ fname))
     else if l == 0 &&fname = "print_newline" then 1
           else if l == 1 && fname = "print" then
                let expr1 = List.hd exprlist in let v_type = get_expr_type expr1
func1  env in

                    match v_type with
                       |FLOAT -> 1
                       | INT_TYPE -> 1
                       | STRING ->1
                       | _ -> raise(Failure("print function can only print int,
float, string!"))

           else if l == 2 && fname  = "getAngle" then
                let expr1 = List.hd exprlist and expr2 = List.hd (List.tl
exprlist)
        in let v_type1 = get_expr_type expr1 func1  env and v_type2
=get_expr_type expr2     func1 env in
         match v_type1,v_type2 with
                   | LINE,LINE -> 1
                   | _,_ -> raise(Failure("getAngle's parameters must be two
lines!"))
           else if l == 2  && fname = "getDis" then
                let expr1 = List.hd exprlist and expr2 = List.hd (List.tl
exprlist)
        in let v_type1 = get_expr_type expr1 func1  env and v_type2
=get_expr_type expr2     func1 env in
         match v_type1,v_type2 with
                   | POINT,POINT -> 1
```

```
                    | _,_ ->raise(Failure("getDis's parameters must be two
points!"))
      else if l = 3 && fname = "Move" then
      let expr1 = List.hd exprlist and expr2 = List.hd (List.tl exprlist) and
expr3 = List.hd (List.tl (List.tl exprlist))
        in let v_type1 = get_expr_type expr1 func1  env and v_type2
=get_expr_type expr2 func1 env  and v_type3  = get_expr_type expr3 func1 env
in
        match v_type1,v_type2,v_type3 with
        |ELLIPSE, INT_TYPE, INT_TYPE -> 1
        |ELLIPSE, INT_TYPE, FLOAT -> 1
        |ELLIPSE, FLOAT, INT_TYPE -> 1
        |ELLIPSE, FLOAT, FLOAT -> 1
        |POLYGON, INT_TYPE, INT_TYPE -> 1
        |POLYGON, INT_TYPE, FLOAT -> 1
        |POLYGON, FLOAT, INT_TYPE -> 1
        |POLYGON, FLOAT, FLOAT -> 1
        |LINE, INT_TYPE, INT_TYPE ->1
        |LINE, FLOAT, INT_TYPE ->1
        |LINE, INT_TYPE, FLOAT ->1
        |LINE, FLOAT, FLOAT ->1
        |POINT, INT_TYPE, INT_TYPE ->1
        |POINT, FLOAT, INT_TYPE ->1
        |POINT, INT_TYPE, FLOAT ->1
        |POINT, FLOAT, FLOAT ->1
        |_,_,_ -> raise(Failure("Wrong usage of function "^ "Move"))

       else
    let func = return_func_given_name fname env in
    let arg_type_list = List.map (fun(e) -> get_expr_type e func1 env) exprlist
(*When you do the function call, you need to cehck the expr list matches every
claimed parameter of the function*)
      in
        if(List.length arg_type_list != List.length func.formal_list)
           then raise(Failure("The numebr of parameters given when calling
function "^ func.fname ^" is worng!"))
        else
           let check_one_by_one count arg_type =  (*arg_type is the type of this
variable when being called*)
               let para = List.nth func.formal_list count in(*get this variable
in the formal list*)
                 let (real_var_type,para_name) = para in (*get its real type*)
```

```
                    if real_var_type = arg_type (*if they are the same, just
pass*)
                       then  count+1
                      else
                          begin
                          match real_var_type, arg_type with
                          | INT_TYPE,FLOAT-> count+1
                          | FLOAT,INT_TYPE -> count+1
                          | _ ->raise(Failure("Type does not all match in the
call expression of "^ fname))
                          end  (*end of check_one_by_one*)
             in
                   List.fold_left check_one_by_one 0 arg_type_list
(*The following function check whether a statements list end with a return
statement*)

let has_return_stmt stmt_list =
    if List.length stmt_list = 0
      then false
    else match (List.hd (List.rev stmt_list)) with
    | Return(_) -> true
    | _->false

(*Check is a statement list, whether one if statements has a return value in
it*)
let if_has_return stmt_list =
    let if_stmt_list = List.filter (function If(_,_,_)->true |_ ->false)
stmt_list in
       let check_result = List.map (
            function
                If(_,s1,s2) ->
                       begin
                           match s1,s2 with
                          | Block(st1), Block(st2) ->(has_return_stmt st1) &&
(has_return_stmt st2)
                          | _ -> raise(Failure("The statements after if or else
should be incldued in {}!"))
                          end
             |_ -> false
           ) if_stmt_list in(*get a list "cehck_result" of boolean to tell
whether each if statement has a return *)
           List.fold_left (fun a b -> b||a) false check_result
```

```
(*The following function will judge whether an expression is valid in a fund
and expr*)

let rec expr_valid func expr env =
    match expr with
    |Assign(id, e1) -> if exist_id id func env
                        then let type1 = get_type env func id and _ = expr_valid
func e1 env and type2 = get_expr_type e1 func env in
                                                if type1 = type2 then true
                                                else
                                                    begin
                                                    match type1, type2 with
                                                    | INT_TYPE,FLOAT -> true
                                                    | FLOAT,INT_TYPE ->true
                                                    | _
->raise(Failure"Unmathed type in Assign operation!")
                                                    end
                                        else raise(Failure("Undeclared
identifier " ^ id ^ " is used!"))
    |Call(fname,exprlist) ->
             if func_name_exist fname env
                      then let _fulfill_valid_exprs = List.map (fun e ->
expr_valid func e env) exprlist in
                           let _check_type = check_func_paralist_type fname
exprlist func env in
                            true
                   else raise(Failure("Undefined function: "^ fname ^ "is
used!"))
    |_ ->
        try
            let _ = get_expr_type expr func env in true
        with Not_found -> raise(Failure("Invalid expression!"))


(*The following function will tell you whether a function's body is valid*)

let check_valid_body func env =
   let rec check_stmt =
        function
```

```
            |BREAK->true
            |CONTINUE->true
            | Block(stmt_list) ->
                let _  = List.map (fun(x) -> check_stmt x) stmt_list in
true(*block case*)
            | ExStmt(expr) -> let vldexpr = expr_valid func expr env and
assign_call = is_assign_call func expr in
                    begin
                                match vldexpr,assign_call with
                                | true, true ->true
                                | true, false -> raise(Failure("There is a
statement in function "^func.fname^" which is not a function call or assignment,
which is invalid!"))
                                | false, _ -> raise(Failure("There is an invalid
expession in function "^ func.fname))
                            end
                |Return(expr) -> let ret  = get_expr_type expr func env in
                                    let real_type  = func.ftype in
                                                    if ret = real_type then
true else raise(Failure("Unmatched return type with function's defination!"))
                |If(expr,stmt1,stmt2) ->
                    let expr_type = get_expr_type expr func env in
                       let _check_expr_type =
                            begin
                               match expr_type with
                                | BOOLEAN -> true
                                | _ -> raise(Failure("expression in If(..)
should be type BOOLEAN!"))
                            end
                                in
                                if (check_stmt stmt1) && (check_stmt stmt2)
                                then true
                                else raise(Failure("Invalid statement in the if
statement in function: "^ func.fname))
                |For(a,b,c,stmt1) -> if expr_valid func a env && expr_valid func
b env && expr_valid func c env && check_stmt stmt1 then true else
raise(Failure("Invalid statement or expressions in For statement in function:"^
func.fname))
                |While(a,stmt1)-> if expr_valid func a env && check_stmt stmt1
then true else raise(Failure("Invalid statement in While statement in
function:"^ func.fname))
                in (*end of check_stmt*)
```

```
      let _ = List.map (check_stmt) func.body in
            true


(*check whether a function's body has a return statement*)
let func_body_has_return func =
    let stmt_list = func.body in
       let result = List.exists (function stmt -> match stmt with |Return(_)
->true |_ ->false) stmt_list   in
            result
(*get all the return statements' corresponding expression type*)
let fun_get_all_return_type func env=
    let stmt_list = func.body in
     let f result_list stmt =
        match stmt with
        |Return(expr) -> (get_expr_type expr func env)::result_list
        |_ -> result_list in
      let result = List.fold_left f [] stmt_list in
          result

(*check whether a function's return statement's type all fulfills the real return
type of this function*)
let check_return func env =
    match func.ftype with
    |  VOID -> if func_body_has_return func then raise(Failure("There should
not be return statement in a void function: "^ func.fname)) else true
    |  f_type ->
        let return_type_list  =  fun_get_all_return_type func env in
          let f a  =  a= f_type in
                if List.for_all  f return_type_list  then true else
raise(Failure("At least one return type does not match the function's defination
requriement!"))

(*THis will check each function's validity*)
let check_func f env =
        let _dup_name = fun_exist f env in
                  let _ = env.functions <- (f) ::env.functions in
           let _dup_formals = check_fpara_duplicate f in
               let _dup_vlocals = check_var_duplicate f in
                   let _vbody = check_valid_body f env in
                       let _check_return_result = check_return f env in
                                                  true
```

```
(*check whether there is a main function*)


let exists_main env =
    if func_name_exist "main" env
       then true else raise(Failure("No Main Function exist!"))
(*the following three functions will judge whether there is global variables
redefiend!*)
let equal_variable_name (a,b) (c,d) =
    b=d

let exist_v_name vlist vdecl =
    let new_fun count x =
        if(equal_variable_name vdecl x) then count+1 else count in
         let result = List.fold_left new_fun 0 vlist in
           if result <=1 then true else raise(Failure("Global Variable has been
redefined!"))

let dup_in_global env =
     List.for_all (exist_v_name env.variables) env.variables


(*define all the built-in functions*)
let f1 = {ftype = VOID;fname = "display";formal_list = [(POINT,"a")];locals =
[];body = []}
let f2 = {ftype = VOID;fname = "display";formal_list = [(LINE,"a")];locals =
[];body = []}
let f3 = {ftype = VOID;fname = "display";formal_list = [(ELLIPSE,"a")];locals
= [];body = []}
let f4 = {ftype = VOID;fname = "display";formal_list = [(POLYGON,"a")];locals
= [];body = []}

let f5 = {ftype = VOID;fname = "Draw";formal_list = [(POINT,"a")];locals =
[];body = []}
let f6 = {ftype = VOID;fname = "Draw";formal_list = [(LINE,"a")];locals = [];body
= []}
let f7 = {ftype = VOID;fname = "Draw";formal_list = [(ELLIPSE,"a")];locals =
[];body = []}
let f8 = {ftype = VOID;fname = "Draw";formal_list = [(POLYGON,"a")];locals =
[];body = []}
```

```
let f9 = {ftype = VOID;fname = "print";formal_list = [(STRING,"a")];locals =
[];body = []}
let f10 = {ftype = FLOAT ;fname = "Perimeter";formal_list =
[(ELLIPSE,"a")];locals = [];body = []}
let f11 = {ftype = FLOAT ;fname = "Area";formal_list = [(ELLIPSE,"a")];locals
= [];body = []}
let f12 = {ftype = VOID ;fname = "Move";formal_list =
[(ELLIPSE,"a");(FLOAT,"b");(FLOAT ,"c")];locals = [];body = []}
let f16 = {ftype = VOID ;fname = "Move";formal_list =
[(POLYGON,"a");(FLOAT,"b");(FLOAT ,"c")];locals = [];body = []}
let f17 = {ftype = VOID ;fname = "Move";formal_list =
[(LINE,"a");(FLOAT,"b");(FLOAT ,"c")];locals = [];body = []}
let f18 = {ftype = VOID ;fname = "Move";formal_list =
[(POINT,"a");(FLOAT,"b");(FLOAT ,"c")];locals = [];body = []}
let f14 = {ftype = FLOAT ;fname = "getAngle";formal_list =
[(LINE,"a");(LINE,"b")];locals = [];body = []}
let f13 = {ftype = VOID ;fname = "print_newline";formal_list = [];locals = [];body
= []}
let f15 = {ftype = FLOAT ;fname = "getDis";formal_list =
[(POINT,"a");(POINT,"b")];locals = [];body = []}
let built_in =
[f1;f2;f3;f4;f5;f6;f7;f8;f9;f10;f11;f12;f13;f14;f15;f16;f17;f18]
let check_program (var_list,fun_list) =
    let env = {functions = built_in;variables = var_list} in
        let _global_check = dup_in_global env in
    let _dovalidation = List.map (fun f -> check_func f env) fun_list in
       let  _mainexist = exists_main env in
           let _ = print_endline "\nThe semantic check has been finished!\n" in
            true
```

```
*****************************************************************************
```

CodeGen.ml Originally authored by Mengqi Zhang (mz2369)

```
*****************************************************************************
open AST
open Printf

let gen_type=function
| INT_TYPE -> "int"
| FLOAT -> "float"
| VOID -> "void"
| BOOLEAN -> "bool"
| POINT -> "point *"
```

```
| LINE -> "line *"
| POLYGON-> "polygon *"
| ELLIPSE   -> "ellipse *"
| ARRAY ->"array"
| STRING->"char *"


let rec gen_expr expr=
    match expr with
    | NUM(f) ->string_of_float f
    | INT(i)-> string_of_int i
    | ID(id)-> id
    | Not(expr)->"!"^(gen_expr expr)
    | Binop( expr1,op ,expr2)->
        begin
            match op with
            | QE->"!strcmp("^(gen_expr expr1)^","^(gen_expr expr2)^")"
            | TTS->"(*"^(gen_expr expr1)^")<=(*"^(gen_expr expr2)^")"
            | TTL->"(*"^(gen_expr expr1)^")>=(*"^(gen_expr expr2)^")"
            | TS->"(*"^(gen_expr expr1)^")<(*"^(gen_expr expr2)^")"
            | TL->"(*"^(gen_expr expr1)^")>(*"^(gen_expr expr2)^")"
            | EE->"("^(gen_expr expr1)^")==("^(gen_expr expr2)^")"
            | NE->"("^(gen_expr expr1)^")!=("^(gen_expr expr2)^")"
            | SE-> "("^(gen_expr expr1)^")<=("^(gen_expr expr2)^")"
            | LE->"("^(gen_expr expr1)^")>=("^(gen_expr expr2)^")"
            | S->"("^(gen_expr expr1)^")<("^(gen_expr expr2)^")"
            | L->"("^(gen_expr expr1)^")>("^(gen_expr expr2)^")"
            | PLUS->"("^(gen_expr expr1)^")+("^(gen_expr expr2)^")"
            | MINUS->"("^(gen_expr expr1)^")-("^(gen_expr expr2)^")"
            | AND->"("^(gen_expr expr1)^")&&("^(gen_expr expr2)^")"
            | OR->"("^(gen_expr expr1)^")||("^(gen_expr expr2)^")"
            | MUL->"("^(gen_expr expr1)^")*("^(gen_expr expr2)^")"
            | DIV->"("^(gen_expr expr1)^")/("^(gen_expr expr2)^")"
            | PERC ->"("^(gen_expr expr1)^")%("^(gen_expr expr2)^")"
            | PARA->"(isParallel(*"^gen_expr expr1^",*"^gen_expr expr2^"))"
            | INTERS->"(getIntersect(*"^gen_expr expr1^",*"^gen_expr
expr2^"))"
            | RELAT->"(getRelation(*"^gen_expr expr1^",*"^gen_expr expr2^"))"
            | T->"(isSimilar(*"^gen_expr expr1^",*"^gen_expr expr2^"))"
            | TE->"(isCongruent(*"^gen_expr expr1^",*"^gen_expr expr2^"))"
            | SS->"(isPerimeterLessThan(*"^gen_expr expr1^",*"^gen_expr
expr2^"))"
```

```
            | LL->"(isPerimeterLargerThan(*"^gen_expr expr1^",*"^gen_expr
expr2^"))"
            | SSE->"(isPerimeterLessOrEqual(*"^gen_expr expr1^",*"^gen_expr
expr2^"))"
            | LLE->"(isPerimeterLargerOrEqual(*"^gen_expr expr1^",*"^gen_expr
expr2^"))"
        end
        | Assign(id,expr)->id^"="^(gen_expr expr)
        | Call(funcname, actual_list)->

        begin
            match funcname with
            | "Area"-> funcname^"(*"^(gen_actual_list actual_list)^")"
            | "Perimeter"-> funcname^"(*"^(gen_actual_list actual_list)^")"
            | "Move"->funcname^"("^(gen_move_list actual_list)^")"
            | "print"->"cout<<"^"("^(gen_actual_list actual_list)^")"
            | "print_newline"->"cout<<endl;"
            | "getAngle"->"getAngle"^(gen_angle_list actual_list)
            | "getDis"->"getDis"^(gen_actual_list actual_list)
            | "Draw"->"Draw(Image1,*"^(gen_actual_list actual_list)^")"
            | _ ->funcname^(gen_actual_list actual_list)
        end
        | String(str)->str
        | PointEx(Point(expr1,expr2))->"new point((float) "^(gen_expr
expr1)^",(float) "^(gen_expr expr2)^")"
        | LineEx(Line(point1,point2))->"new line("^(gen_point
point1)^","^(gen_point point2)^")"
        | LineEx(LineID(p1,p2))->"new line("^p1^","^p2^")"
        | PolygonEx(Polygon(point_list))->"new polygon("^(string_of_int
(List.length point_list))^",new point*["^(string_of_int (List.length
point_list))^"]"^(gen_points point_list)^")"
        | PolygonEx(PolygonID(p_list))->"new polygon("^(string_of_int
(List.length p_list))^",new point*["^(string_of_int (List.length
p_list))^"]"^(gen_pointsid p_list)^")"
        | EllipseEx(Ellipse(point, expr1,expr2))->"new ellipse("^(gen_point
point)^",(float) "^(gen_expr expr1)^",(float) "^(gen_expr expr2)^")"
        | EllipseEx(EllipseID(point, expr1,expr2))->"new
ellipse("^point^",(float) "^(gen_expr expr1)^",(float) "^(gen_expr expr2)^")"

        and gen_actual_list =function
            | actual_list ->"("^(String.concat "," (List.map gen_expr
actual_list))^")"
```

```
        and gen_move_list=function
            | point::list->"*"^(gen_expr point)^","^(String.concat ","
(List.map gen_expr list))
            | _->""

        and gen_angle_list=function
            | list->"(*"^(String.concat ",*" (List.map gen_expr list))^")"

        and gen_point=function
            | Point(expr1,expr2)->"new point((float) "^(gen_expr
expr1)^",(float) "^(gen_expr expr2)^")"

        and gen_points point_list=
            "{"^(String.concat "," (List.map gen_point point_list))^"}"

        and gen_pointsid p_list=
            "{"^(String.concat "," p_list)^"}"

let rec gen_stmt stmt=
    match stmt with
    | ExStmt(expr)->(gen_expr expr)^";"
    | Return(expr)->"return "^(gen_expr expr)^";"
    | Block(stmt_list)->"{"^(String.concat "\n" (List.map gen_stmt
stmt_list))^"}"
    | If(expr,stmt1,stmt2)->"if("^(gen_expr expr)^")\n"^(gen_stmt
stmt1)^"\nelse\n"^(gen_stmt stmt2)
    | For(expr1,expr2,expr3,stmt)->"for("^(gen_expr expr1)^";"^(gen_expr
expr2)^";"^(gen_expr expr3)^")\n"^(gen_stmt stmt)
    | While(expr,stmt)->"while("^(gen_expr expr)^")\n"^(gen_stmt stmt)
    | BREAK-> "break;"
    | CONTINUE->"continue;"

let gen_var_decl_list var_list=
    let gen_var_decl (var_type, id)=
    (gen_type var_type)^" "^id
    in
    String.concat "," (List.map gen_var_decl var_list)

let gen_locals local_list=
    let gen_var_decl (var_type, id)=
    (gen_type var_type)^" "^id
```

```
    in
    match local_list with
    | []->""
    | _->(String.concat ";\n" (List.map gen_var_decl local_list) )^";"

let gen_fdecl fdecl=
    let ftype=gen_type (fdecl.ftype) in
    let fname=fdecl.fname in
    let formal_list=(gen_var_decl_list fdecl.formal_list) in
    let locals=gen_locals fdecl.locals in
    let body=String.concat "\n" (List.map gen_stmt fdecl.body) in
    match fname with
    | "main"->ftype^" "^fname^" "^"("^formal_list^")\n{\n"^locals^"\nCvSize
ImageSize1=cvSize(1000,1000);\nImage1=cvCreateImage(ImageSize1,IPL_DEPTH_8U
,3);\ncvNot(Image1,Image1);\n"^body^"\n\nDrawAx(Image1);\ncvNamedWindow(\"A
LG\",1);\ncvShowImage(\"ALG\",Image1);\ncvWaitKey(0);\n}"
    | _->ftype^" "^fname^" "^"("^formal_list^")\n{"^locals^"\n"^body^"\n}"


let gen_program (var_list, fdecl_list)=
    let vars=gen_locals var_list in
    let fdecls= String.concat "\n" (List.map gen_fdecl fdecl_list) in
    let header="#include \"alg.h\"\n#include
\"stdio.h\"\n#include\"string.h\"\n" in
    let _ = print_endline "\nThe Code Generation has been finished!\n" in
    header^vars^"\n"^fdecls
```

```
/*******************************************************************************
```

alg.h Originally authored by Qingye Jiang (qj2116), Qiang Deng (qd2114), Ainur Rysbekova (ar3005)

```
*******************************************************************************
using namespace std;
#include "stdio.h"
#include <iostream>
#include <cmath>
#include "string.h"
#include "stdlib.h"
#include </opt/local/include/opencv/cv.h>
#include </opt/local/include/opencv/highgui.h>

/*************************************************************/
```

```
//Draw a picture
IplImage *Image1;
int Position=500;
int Color=255;
int Shift=0;


class  point
{
public:
    float x;
    float y;
    point(float a,float b);
};

point::point(float a,float b){x=a;y=b;}

class line
{
public:
    point * p1;
    point  * p2;
    line(point *  a,point *  b);
};
line::line(point* a,point*  b){p1=a;p2=b;}

struct ellipse
{
    point * focusPoint;
    float a;//the length of major semi-axes
    float b;//the length of minot semi-axes
    ellipse(point * p, float x, float y);
};

ellipse::ellipse(point * p, float x, float y){focusPoint=p;a=x;b=y;}

class polygon
{
public:
    int pointNum;
    point ** pointCollection;
    polygon(int pointNum, point ** pointCollection);
```

```
};
polygon::polygon(int pointNum1, point **
pointCollection1){pointNum=pointNum1;pointCollection=pointCollection1;}

/*************************************************************/
//Draw a point
void Draw(IplImage *Image1, point p ){
    cvCircle(Image1, cvPoint(Position+p.x*10, Position-p.y*10), 2,
CV_RGB(0,0,0),CV_FILLED);
}

void Draw(IplImage *Image1, line l){
    float x1 = 0;
    float y1 = 0;
    float x2 = 0;
    float y2 = 0;
    float xax1 = 0;
    float yax1 = 0;
    float xax2 = 0;
    float yax2 = 0;
    x1 = Position+l.p1->x*10;
    y1 = Position-l.p1->y*10;
    x2 = Position+l.p2->x*10;
    y2 = Position-l.p2->y*10;
    if((l.p2->x-l.p1->x)==0){
        xax1 = x1 + (1000-y1)*(x2-x1)/(y2-y1);
        yax1 = 1000;
        xax2 = x1 + (0-y1)*(x2-x1)/(y2-y1);
        yax2 = 0;
    }
    else{
        xax1 = 1000;
        xax2 = 0;
        yax1 = (y2-y1)*(1000-x1)/(x2-x1) + y1;
        yax2 = (y2-y1)*(0-x1)/(x2-x1) + y1;
    }

    cvLineAA(Image1, cvPoint(xax1, yax1), cvPoint(xax2, yax2), Color, Shift);
}


//Draw an ellipse
```

```
void Draw(IplImage *Image1, ellipse e){
    cvEllipseAA(Image1, cvPoint(Position+e.focusPoint->x*10,
Position-10*e.focusPoint->y), cvSize(e.a*10, e.b*10), 0, 0, 360, Color, Shift);
}

//Draw a polygon
void Draw(IplImage *Image1, polygon py){
    int pointNum = py.pointNum;
    int i = 0;
    point **pointCollection = py.pointCollection;
    while (i != pointNum-1){
        cvLineAA(Image1, cvPoint(Position+(*(pointCollection[i])).x*10,
Position-10*(*pointCollection[i]).y),
cvPoint(Position+(*pointCollection[i+1]).x*10,
Position-10*(*pointCollection[i+1]).y), Color, Shift);
        i ++ ;
    }
    cvLineAA(Image1, cvPoint(Position+10*(*pointCollection[pointNum-1]).x,
Position-10*(*pointCollection[pointNum-1]).y),
cvPoint(Position+10*(*pointCollection[0]).x,
Position-10*(*pointCollection[0]).y), Color, Shift);

}

//Draw the x,y axes
void DrawAx(IplImage *Image1){
    cvLineAA(Image1, cvPoint(0,500),cvPoint(1000,500),0,Shift);
    cvLineAA(Image1, cvPoint(500,0),cvPoint(500,1000),0,Shift);
}

/*************************************************************/



/************************The following code are used to judge the relation between
two lines******************/
void display(point *p)
{
    printf("Point:%f %f\n", p->x,p->y);

}
```

```
void display(line *l)
{
    printf("This is a line!\n");
    printf("The first point is (%f %f) \n",l->p1->x,l->p1->y);
    printf("The second point is (%f %f) \n", l->p2->x,l->p2->y);
}
void display(polygon *p)
{
    printf("This is a polygon with %d edges! \n",p->pointNum);
    printf("The points of this polygon is: \n");
    int num = p->pointNum,i;
    point ** C = p->pointCollection;
    for(i = 0;i<num;i++)
    {
        printf("(%f %f)\n", C[i]->x, C[i]->y);
    }
}
void display(ellipse *e)
{
    printf("This is an ellipse!\nIts focus point is (%f %f)\nIts major semi-axe's
length is %f, and its minor semi-axe's length is %f\n", e->focusPoint->x,
e->focusPoint->y, e->a, e->b);
}

point * getPoint(float f1, float f2)
{
    point *p=new point(f1,f2);
    return p;
}

point *getIntersect(line l1, line l2)
{
    bool isParallel(line l1, line l2);
    point *result = getPoint(0,0);
    if(isParallel(l1,l2))
    {
        printf("these two linse are parallel!\n");
        return result;// if these two lines are parallel, just return the original
point with a warning.
    }
    else
    {
```

```
        float x1 = l1.p1->x, y1 = l1.p1->y, x2 = l1.p2->x,y2 = l1.p2->y;
        float h1 = l2.p1->x, k1 = l2.p1->y, h2 = l2.p2->x, k2 = l2.p2->y;
        float temp2,temp1,delta1,delta2;
    float resultX,resultY;
        delta1 = h1-h2;
        delta2 = x1-x2;
        if(delta1 == 0 && delta2==0)
         return result;
    else if(delta1==0)
        {resultX=h1;resultY=(y2-y1)*(resultX-x1)/(x2-(float)x1)+y1;
    result = getPoint(resultX, resultY);
        return result;}
        else if(delta2 == 0)
         {resultX=x1;
    resultY=(k2-k1)*(resultX-h1)/(h2-(float)h1)+k1;
    result = getPoint(resultX, resultY);
        return result;}


        temp2 = (k1-k2)/delta1; temp1 = (y1-y2)/delta2;

        float a = temp1-temp2,b = k1-y1-h1*temp2+x1*temp1;
         resultX = b/a;
        resultY = temp1*(resultX-x1) +y1;
        result = getPoint(resultX, resultY);
        return result;
    }

}// get the intersect point of two lines



bool isParallel(line l1, line l2)
{
    float deltaX1 = l1.p1->x-l1.p2->x,deltaY1 = l1.p1->y-l1.p2->y;
    float deltaX2 = l2.p1->x-l2.p2->x, deltaY2 = l2.p1->y-l2.p2->y;
    if(deltaX1 == 0 && deltaX2 == 0)
        return true;
    if(deltaX1*deltaX2 == 0 && deltaX1+deltaX2!=0)
        return false;
    if(deltaY1/deltaX1 != deltaY2/deltaX2)
        return false;
```

```
    return true;
}//judge whether two lines are parallel
float getDis(point *p1, point *p2)
{
    return
(float)sqrt(pow((double)(p1->x-p2->x),2.0)+pow((double)(p1->y-p2->y),2.0));
}

char * getRelation(ellipse  circle1, ellipse  circle2)
{
     char * str=(char*)malloc(sizeof(char)*30);
    if(circle1.a!=circle1.b || circle2.a!=circle2.b)
    {
        printf("These two shapes are not both circles!\n");
     strcpy(str,"");
        return str;//inform a mistake
    }

    float r1 = circle1.a, r2 = circle2.a;

    float dis = getDis(circle1.focusPoint, circle2.focusPoint);
    if(fabs(dis-r1-r2)<=0.0001)
        strcpy(str,"circumscribe");// they are circumscribe
    else if(fabs(dis-fabs(r1-r2))<=0.0001)
        strcpy(str,"inscribe");// they are inscribe
    else if(dis<fabs(r1-r2))
        strcpy(str,"contain");// one circle contains another one
    else if(dis>r1+r2)
          strcpy(str,"seperate");//they are seperated
    else if(r1+r2>dis && dis > fabs(r1-r2))
       strcpy(str,"secant");// they are secant

    return str;

}


char * getRelation(line l, ellipse e)
{
    float y1 = l.p1->y, y2 = l.p2->y, x1 = l.p1->x, x2 = l.p2->x;
    float h = e.focusPoint->x, k = e.focusPoint->y, a = e.a, b= e.b;
    float k1 = y1-y2, k2 = x2-x1, k3 = 2*x1*y1-x1*y2-y1*x2;
```

```
    float Ba = (a*a*k1*k1)/(k2*k2)+b*b;
    float Bb = (2*a*a*k*k1)/k2 - (2*a*a*k1*k3)/(k2*k2)-2*b*b*h;
    float Bc = (a*a*k3*k3)/(k2*k2) +a*a*k*k -(2*a*a*k*k3)/(k2)+b*b*h*h-a*a*b*b;
    //cout<<Ba<<endl;
    char * str=(char*)malloc(sizeof(char)*30);
    if(k2 == 0)// special case for the vertical line
    {
       if(fabs((k3/k1)-h) == a)
           {strcpy(str,"tangent");return str;}//tangent
       else if((k3/k1)>h-a && (k3/k1)<h+a)
         { strcpy(str,"secant");return str;}//secant
        else
           {strcpy(str,"seperate");return str;}//seperate

    }
    float temp = Bb*Bb-4*Ba*Bc;
  //  cout<<temp<<endl;
   // cout<<temp<<endl;
    if(temp>0)
        {strcpy(str,"secant");return str;}//secant
    else if(temp==0)
        {strcpy(str,"tangent");return str;}//tangent
    else
       { strcpy(str,"seperate");return str;}//seperate
}


line * getLine(point P1, point P2)
{
    line * l=new line(&P1,&P2);
    return l;

}

float getAngle(line l1, line l2)// get the angle between two lines
{
    float Vx1 = l1.p1->x - l1.p2->x;
    float Vy1 = l1.p1->y - l1.p2->y;
    float Vx2 = l2.p1->x - l2.p2->x;
    float Vy2 = l2.p1->y - l2.p2->y;
    return acos((Vx1*Vx2+Vy1*Vy2)/(getDis(l1.p1,l1.p2)*getDis(l2.p1,l2.p2)));
}
```

```
bool naiveSimilar(polygon shape1, polygon shape2,int start1, int start2)// assume
we know where to start to check whether two polygond are similar
{

    int i = 0,j = 0;
    int num1 = shape1.pointNum,num2 = shape2.pointNum;
    point ** C1 = shape1.pointCollection;
    point ** C2 = shape2.pointCollection;

    while(i<num1 && j<num2)
    {
            line *shape1Line1 =
getLine(*C1[(start1+i)%num1],*C1[(start1+i-1+num1)%num1]);
            line *shape1Line2 =
getLine(*C1[(start1+i)%num1],*C1[(start1+i+1)%num1]);
            line *shape2Line1 =
getLine(*C2[(start2+j)%num2],*C2[(start2+j-1+num2)%num2]);
            line *shape2Line2 =
getLine(*C2[(start2+j)%num2],*C2[(start2+j+1)%num2]);
            float shape1Line1Length =
getDis(C1[(start1+i)%num1],C1[(start1+i-1+num1)%num1]);
            float shape1Line2Length =
getDis(C1[(start1+i)%num1],C1[(start1+i+1)%num1]);
            float shape2Line1Length =
getDis(C2[(start2+j)%num2],C2[(start2+j-1+num2)%num2]);
            float shape2Line2Length =
getDis(C2[(start2+j)%num2],C2[(start2+j+1)%num2]);
            if(fabs(getAngle(*shape1Line1,*shape1Line2) -
getAngle(*shape2Line1,*shape2Line2)) >= 0.001)
            {

                return false;
            }

if(fabs((shape1Line1Length/shape2Line1Length)-(shape1Line2Length/shape2Line2Len
gth)) >=0.001)
            {
                return false;//we should also check whether the ratio of edges' length
are the same
```

```
            }

        i++;
        j++;
    }//endwhile


    return true;

}

bool naiveSimilar2(polygon shape1, polygon shape2,int start1, int start2)// assume
we know where to start to check whether two polygond are similar
{

    int i = 0,j = 0;
    int num1 = shape1.pointNum,num2 = shape2.pointNum;
    point **C1 = shape1.pointCollection;
    point ** C2 = shape2.pointCollection;
    while(i<num1 && j<num2)
    {

        line *shape1Line1 =
getLine(*C1[(start1+i)%num1],*C1[(start1+i-1+num1)%num1]);
        line *shape1Line2 = getLine(*C1[(start1+i)%num1],*C1[(start1+i+1)%num1]);
        line *shape2Line1 =
getLine(*C2[(start2-j+num2)%num2],*C2[(start2-j+1+num2)%num2]);
        line *shape2Line2 =
getLine(*C2[(start2-j+num2)%num2],*C2[(start2-j-1+num2)%num2]);
        float shape1Line1Length =
getDis(C1[(start1+i)%num1],C1[(start1+i-1+num1)%num1]);
        float shape1Line2Length =
getDis(C1[(start1+i)%num1],C1[(start1+i+1)%num1]);
        float shape2Line1Length =
getDis(C2[(start2-j+num2)%num2],C2[(start2-j+1+num2)%num2]);
        float shape2Line2Length =
getDis(C2[(start2-j+num2)%num2],C2[(start2-j-1+num2)%num2]);
        if(fabs(getAngle(*shape1Line1,*shape1Line2) -
getAngle(*shape2Line1,*shape2Line2)) >= 0.001)
        {
            return false;
        }
```

```
if(fabs((shape1Line1Length/shape2Line1Length)-(shape1Line2Length/shape2Line2Len
gth)) >=0.001)
        {
            return false;//we should also check whether the ratio of edges' length
are the same
        }

        i++;
        j++;
    }//end while

    return true;

}// counter clockwise check

bool isSimilar(polygon shape1, polygon shape2)//judge whether two polygons are
similar
{
    int i;
    int num1 = shape1.pointNum;
    int num2 = shape2.pointNum;
    if(num1!=num2)
        {
     return false;// if the two polygon have different point numbers, then they are
not similar
     }
    else
    {

       for(i = 0;i<num1;i++)
        {
         //printf("ok");
            if(naiveSimilar(shape1,shape2,i,0) || naiveSimilar2(shape1, shape2, i,
0))
                return true;
       }
    }
}
```

```
float Perimeter(polygon p)
{
    int num = p.pointNum;
    float result = 0;
    point ** C = p.pointCollection;
    for(int i = 0;i<num;i++)
    {
        result = result +getDis(C[i%num],C[(i+1)%num]);
    }
    return result;
}


float Perimeter(ellipse e)
{
    return 3.14159*(3*(e.a+e.b)-sqrt((3*e.a+e.b)*(e.a+3*e.b)));
}

bool isPerimeterLessThan(polygon p1,polygon p2)
{
    float perimeter1=Perimeter(p1);
    float perimeter2=Perimeter(p2);
    return (perimeter1<perimeter2);
}

bool isPerimeterLessThan(ellipse e1,ellipse e2)
{
    float perimeter1=Perimeter(e1);
    float perimeter2=Perimeter(e2);
    return (perimeter1<perimeter2);
}


bool isPerimeterLessOrEqual(polygon p1,polygon p2)
{
    float perimeter1=Perimeter(p1);
    float perimeter2=Perimeter(p2);
    return (perimeter1<=perimeter2);
}

bool isPerimeterLessOrEqual(ellipse e1,ellipse e2)
```

```
{
    float perimeter1=Perimeter(e1);
    float perimeter2=Perimeter(e2);
    return (perimeter1<=perimeter2);
}

bool isPerimeterLargerThan(polygon p1,polygon p2)
{
    float perimeter1=Perimeter(p1);
    float perimeter2=Perimeter(p2);
    return (perimeter1>perimeter2);
}

bool isPerimeterLargerThan(ellipse e1,ellipse e2)
{
    float perimeter1=Perimeter(e1);
    float perimeter2=Perimeter(e2);
    return (perimeter1>perimeter2);
}

bool isPerimeterLargerOrEqual(polygon p1,polygon p2)
{
    float perimeter1=Perimeter(p1);
    float perimeter2=Perimeter(p2);
    return (perimeter1>=perimeter2);
}

bool isPerimeterLargerOrEqual(ellipse e1,ellipse e2)
{
    float perimeter1=Perimeter(e1);
    float perimeter2=Perimeter(e2);
    return (perimeter1>=perimeter2);
}




float getTriangleArea(polygon triangle)
{
    if(triangle.pointNum != 3)
    {
        printf("error in getting triangle area! This is not a triangle!\n");
```

```
        return 0;
    }
    point **C = triangle.pointCollection;

    float result = 0;
    result = result + (*C[0]).x*((*C[1]).y-(*C[2]).y);
    result = result + (*C[1]).x*((*C[2]).y-(*C[0]).y);
    result = result + (*C[2]).x*((*C[0]).y-(*C[1]).y);
    return fabs(result/2);
}

polygon * getPolygon(int num, point ** collection)
{
    polygon * result=new polygon(num,collection);
    return result;
}

float Area(polygon p)
{
    int num =p.pointNum;
    float result =0;
    point ** C = p.pointCollection;
    if(num<=2)
        return 0;
    if(num == 3)
        return getTriangleArea(p);
    else
    {
        point * C1[3] = {C[num-2],C[num-1],C[0]};
        point * C2[3] = {C[0],C[1],C[2]};
        polygon t1 = *getPolygon(3,C1);
        polygon t2 = *getPolygon(3,C2);
        result = result + getTriangleArea(t1);
        result = result + getTriangleArea(t2);//first add the two trianlges on the
edge
        for(int i = 2;i<num-2;i++)
        {
            point *tempC[3] = {C[0],C[i],C[i+1]};
            polygon tempT = *getPolygon(3, tempC);
            result = result+getTriangleArea(tempT);

        }
```

```
        return result;

    }
}

bool operator<= (polygon p1,polygon p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1<=area2);
}

bool operator< (polygon p1,polygon p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1<area2);
}

bool operator>= (polygon p1,polygon p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1>=area2);
}

bool operator> (polygon p1,polygon p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1>area2);
}

float Area(ellipse e)
{
    return 3.14159*e.a*e.b;
}


bool operator<= (ellipse p1,ellipse p2)
{
    float area1=Area(p1);
```

```
    float area2=Area(p2);
    return (area1<=area2);
}

bool operator< (ellipse p1,ellipse p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1<area2);
}

bool operator>= (ellipse p1,ellipse p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    printf("%f%f\n",area1,area2);
    return (area1>=area2);
}

bool operator> (ellipse p1,ellipse p2)
{
    float area1=Area(p1);
    float area2=Area(p2);
    return (area1>area2);
}



bool naiveCongruent(polygon shape1, polygon shape2,int start1, int start2)// assume
we know where to start to check whether two polygond are similar
{
    int i = 0,j = 0;
    int num1 = shape1.pointNum,num2 = shape2.pointNum;
    point ** C1 = shape1.pointCollection;
    point ** C2 = shape2.pointCollection;
    while(i<num1 && j<num2)
    {
        line *shape1Line1 =
getLine(*C1[(start1+i)%num1],*C1[(start1+i-1+num1)%num1]);
        line *shape1Line2 = getLine(*C1[(start1+i)%num1],*C1[(start1+i+1)%num1]);
        line *shape2Line1 =
getLine(*C2[(start2+j)%num2],*C2[(start2+j-1+num2)%num2]);
```

```
        line *shape2Line2 = getLine(*C2[(start2+j)%num2],*C2[(start2+j+1)%num2]);
        float shape1Line1Length =
getDis(C1[(start1+i)%num1],C1[(start1+i-1+num1)%num1]);
        float shape1Line2Length =
getDis(C1[(start1+i)%num1],C1[(start1+i+1)%num1]);
        float shape2Line1Length =
getDis(C2[(start2+j)%num2],C2[(start2+j-1+num2)%num2]);
        float shape2Line2Length =
getDis(C2[(start2+j)%num2],C2[(start2+j+1)%num2]);
        if(fabs(getAngle(*shape1Line1,*shape1Line2) -
getAngle(*shape2Line1,*shape2Line2)) >= 0.0001)
        {
            return false;
        }
        if(fabs(shape1Line1Length-shape2Line1Length)>0.0001 ||
fabs(shape1Line2Length-shape2Line2Length)>0.0001)
        {
            return false;//we should also check whether the ratio of edges' length
are the same
        }
        i++;
        j++;
    }
    return true;

}



bool naiveCongruent2(polygon shape1, polygon shape2,int start1, int start2)//
assume we know where to start to check whether two polygond are congruent
{
    int i = 0,j = 0;
    int num1 = shape1.pointNum,num2 = shape2.pointNum;
    point ** C1 = shape1.pointCollection;
    point ** C2 = shape2.pointCollection;
    while(i<num1 && j<num2)
    {
        line *shape1Line1 =
getLine(*C1[(start1+i)%num1],*C1[(start1+i-1+num1)%num1]);
        line *shape1Line2 = getLine(*C1[(start1+i)%num1],*C1[(start1+i+1)%num1]);
        line *shape2Line1 =
```

```
getLine(*C2[(start2-j+num2)%num2],*C2[(start2-j+1+num2)%num2]);
        line *shape2Line2 =
getLine(*C2[(start2-j+num2)%num2],*C2[(start2-j-1+num2)%num2]);
        float shape1Line1Length =
getDis(C1[(start1+i)%num1],C1[(start1+i-1+num1)%num1]);
        float shape1Line2Length =
getDis(C1[(start1+i)%num1],C1[(start1+i+1)%num1]);
        float shape2Line1Length =
getDis(C2[(start2-j+num2)%num2],C2[(start2-j+1+num2)%num2]);
        float shape2Line2Length =
getDis(C2[(start2-j+num2)%num2],C2[(start2-j-1+num2)%num2]);
        if(fabs(getAngle(*shape1Line1,*shape1Line2) -
getAngle(*shape2Line1,*shape2Line2)) >= 0.0001)
        {
            return false;
        }
        if(fabs(shape1Line1Length-shape2Line1Length)>0.0001 ||
fabs(shape1Line2Length!=shape2Line2Length)>0.0001)
        {
            return false;//we should also check whether the ratio of edges' length
are the same
        }

        i++;
        j++;
    }
    return true;

}// counter clockwise check


bool isCongruent(polygon shape1, polygon shape2)//judge whether two polygons are
similar
{
    int i;
    int num1 = shape1.pointNum,num2 = shape2.pointNum;
    if(num1 != num2)
        return false;// if the two polygon have different point numbers, then they
are not similar
    else
    {
        for(i = 0;i<num1;i++)
```

```
        {
            if(naiveCongruent(shape1,shape2,i,0) || naiveCongruent2(shape1, shape2,
i, 0))
                return true;
        }
        return false;

    }
}


void Move(point &p, float x, float y)// move a point based on the vector (x,y)
{
    p.x = p.x+x;
    p.y = p.y+y;
}
void Move(line &l, float x, float y)// move a line based on the vector (x,y)
{
    l.p1->x = l.p1->x+x;
    l.p1->y = l.p1->y+y;
    l.p2->x = l.p2->x+x;
    l.p2->y = l.p2->y+y;
}
void Move(polygon &poly, float x, float y)// move a polygon based on the vector (x,y)
{
    int num = poly.pointNum,i;
    point ** pointCollection=poly.pointCollection;
    for(i = 0;i<num;i++)
    {
        pointCollection[i]->x = pointCollection[i]->x+x;
        pointCollection[i]->y = pointCollection[i]->y+y;
    }

}

void Move(ellipse &e, float x, float y)// move an ellipse based on the vector (x,y)
{
    Move(*(e.focusPoint), x, y);
}
```

```
*****************************************************************************
```

compiler.ml Originally authored by Mengqi Zhang (mz2369)

```
*****************************************************************************
open Printf

    let _ =
    let lexbuf = Lexing.from_channel (open_in "input.alg") in
    let parse_prog=ALG_Parser.program ALG_Scanner.token lexbuf in
    let _=Semantics.check_program parse_prog in
    let prog=CodeGen.gen_program parse_prog in
    let out_channel = open_out "output.c" in
    output_string out_channel prog


*****************************************************************************
```

compiler.sh Originally authored by Mengqi Zhang (mz2369)

```
*****************************************************************************
#!/bin/bash

cat $1 >input.alg
export PKG_CONFIG_PATH=/opt/local/lib/pkgconfig
./compiler
g++ -o $2 output.c `pkg-config --cflags --libs opencv`


*****************************************************************************
```

makefile Originally authored by Mengqi Zhang (mz2369)

```
*****************************************************************************
all: AST.cmi compiler
compiler: CodeGen.cmo Semantics.cmo ALG_Parser.cmo ALG_Scanner.cmo
compiler.cmo
    ocamlc  -o compiler CodeGen.cmo Semantics.cmo ALG_Parser.cmo
ALG_Scanner.cmo  compiler.cmo

compiler.cmo : compiler.ml
    ocamlc -c compiler.ml

ALG_Scanner.cmo : ALG_Scanner.ml
    ocamlc -c ALG_Scanner.ml

ALG_Scanner.ml : ALG_Scanner.mll
    ocamllex ALG_Scanner.mll
```

```
ALG_Parser.cmo:ALG_Parser.ml
    ocamlc -c ALG_Parser.ml
ALG_Parser.ml: ALG_Parser.mli
    ocamlc -c ALG_Parser.mli
ALG_Parser.mli:  ALG_Parser.mly
    ocamlyacc ALG_Parser.mly

Semantics.cmo: Semantics.ml
    ocamlc -c Semantics.ml

CodeGen.cmo:CodeGen.ml
    ocamlc -c CodeGen.ml


AST.cmi: AST.mli
    ocamlc -c AST.mli
clean:
    rm -f *.cmo *.cmi ALG_Parser.mli ALG_Parser.ml ALG_Scanner.ml AST.cmi
```

****

Legal Functional Test Set_1 Originally authored by Mengqi Zhang (mz2369)

****

```
!!cal_para_line.alg
def int main()
{
    int i;
    line l1;
    line l2;
    i=0;
    for(i=0;i<5;i=i+1)
    {
        l1=[[1;1],[2;2]];
        l2=[[1;2],[2;i]];
        if(l1//l2)
        {
        print(i);
        print_newline();
        print("l1 parallel l2");
        print_newline();
        display(l2);
```

```
        }
    }
}

!!inter_area.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    point p1;
    point p2;
    point p3;
    polygon poly;
    l1=[[1;1],[2;2]];
    l2=[[0;0],[2;0]];
    l3=[[2;2],[2;0]];
    (!display the information of the three lines!)
    display(l1);
    display(l2);
    display(l3);
    p1=l1^l2; !!p1 is the intersect point of l1 and l2
    p2=l2^l3; !!p2 is the intersect point of l2 and l3
    p3=l3^l1;   !!p3 is the intersect point of l1 and l3

    poly=[p1,p2,p3]; !! poly is the intersect area of the three lines
    print("the area of the intersection area is ");
    print(Area(poly)); !!print the area of the intersection part of the three
lines
    print_newline();
}

!!angle_dis.alg
def int main()
{
    line l1;
    line l2;
    point p1;
    point p2;
    p1=[0;0];
    p2=[1;1];
    l1=[[0;0],[2;0]];
```

```
    l2=[[1;1],[2;2]];
    print("the angle of the two lines are ");
    print(getAngle(l1,l2));
    print_newline();
    print("the distance of the two points are ");
    print(getDis(p1,p2));
    print_newline();
}

!!area.alg
def int main()
{
    ellipse e1;
    polygon poly1;
    e1={[2.3;1],1,1};
    poly1=[[0;0],[2;0],[1;1]];
    print("the area of the ellipse is ");
    print(Area(e1));
    print_newline();
    print("the area of the polygon is ");
    print(Area(poly1));
    print_newline();
}

!!display.alg
def int main()
{
    line l1;
    ellipse e1;
    polygon poly1;
    point p1;
    p1=[1;2];
    l1=[[1;1],[1;2]];
    e1={[1;1],2,1};
    poly1=[[1;1],[2;3],[3;5],[4;6],[3;4]];
    display(l1);
    display(e1);
    display(p1);
    display(poly1);
}

!!draw.alg
```

```
def int main()
{
    line l1;
    ellipse e1;
    polygon poly1;
    point p1;
    p1=[1;2];
    l1=[[1;1],[1;2]];
    e1={[1;1],2,1};
    poly1=[[1;1],[2;3],[3;5],[4;6],[3;4]];
    Draw(l1);
    Draw(e1);
    Draw(p1);
    Draw(poly1);
}
```

```
!!input_builtin.alg
def int main()
{
    line l1;
    ellipse e1;
    line l2;
    polygon poly1;
    point p1;
    point p2;
    float f;
    p1=[1;2];
    p2=[1;3];
    l2=[[1;2],[3;3]];
    l1=[[1;1],[1;2]];
    e1={[1;1],2,1};
    poly1=[[1;1],[2;2],[3;3],[5;6],[7;8]];
    display(l1);
    Move(l1,1,2);
    display(l1);
    display(e1);
    Move(e1,1,2);
    display(e1);
    display(p1);
    Move(p1,1,2);
    display(p1);
    display(poly1);
```

```
    Move(poly1,1,2);
    display(poly1);
    Area(poly1);
    Area(e1);
    Perimeter(poly1);
    Perimeter(e1);
    f=getAngle(l1,l2);
    f=getDis(p1,p2);
}

!!move.alg
def int main()
{
    point p1;
    line l1;
    ellipse e1;
    polygon poly1;
    p1=[0;0];
    l1=[[1;1],[2;2]];
    e1={[0;0],1,1};
    poly1=[[0;0],[1;1],[2;1],[1;0]];
    print("The original figures are as follows ");
    print_newline();
    display(p1);
    display(l1);
    display(e1);
    display(poly1);
    Move(p1,1.5,2.0);
    Move(l1,1.5,2.0);
    Move(e1,1.5,2.0);
    Move(poly1,1.5,2.0);
    print("After move they are ");
    print_newline();
    display(p1);
    display(l1);
    display(e1);
    display(poly1);
}

!!perimeter.alg
def int main()
{
```

```
    ellipse e1;
    polygon poly1;
    e1={[2.3;1],1,1};
    poly1=[[0;0],[2;0],[1;1]];
    print("the perimeter of the ellipse is ");
    print(Perimeter(e1));
    print_newline();
    print("the perimeter of the polygon is ");
    print(Perimeter(poly1));
    print_newline();
}

!!print.alg
def int main()
{
    print(2);
    print_newline();
    print(2.1);
    print_newline();
    print("string");
}

!!draw1.alg
def int main()
{
    ellipse e1;
    ellipse e2;
    line l1;
    line l2;
    l1=[[1;1],[2;2]];
    l2=[[1;-1],[2;-2]];
    e1={[-14.14;0],10,10};
    e2={[14.14;0],10,10};
    Draw(l1);
    Draw(l2);
    Draw(e1);
    Draw(e2);
}

!!ellipse_ellipse.alg
def int main()
{
```

```
    ellipse e1;
    ellipse e2;
    int i;
    i=25;
    e1={[0;0],10,10};
    e2={[i;0],5,5};
    Draw(e1);
    while(i>=0)
    {
        i=i-.5;
        Draw(e2);
        display(e2);
        print(e1|-e2);
        print_newline();
        Move(e2,-5,0);
    }
}

!!ellipse_line.alg
def int main()
{
    ellipse e1;
    line l1;
    int i;
    i=50;
    e1={[0;0],10,10};
    l1=[[50;0],[50;1]];
    Draw(e1);
    while(i>=0)
    {
        Draw(l1);
        if((l1|-e1):="tangent")
        {
            print("tangent");
            print_newline();
            display(l1);
            done;
        }
        Move(l1,-5,0);
    }
}
```

```
!!hexagon.alg
def int main()
{
    point p1;
    point p2;
    point p3;
    point p4;
    point p5;
    point p6;
    polygon poly1;
    polygon poly2;
    polygon poly3;
    polygon poly4;
    p1=[10;0];
    p2=[0;10];
    p3=[10;20];
    p4=[20;20];
    p5=[30;10];
    p6=[20;0];
    poly1=[p1,p2,p3];
    poly2=[p1,p2,p3,p4];
    poly3=[p1,p2,p3,p4,p5];
    poly4=[p1,p2,p3,p4,p5,p6];
    Draw(poly1);
    Draw(poly2);
    Draw(poly3);
    Draw(poly4);
    print("Areas ");
    print(Area(poly1));
    print_newline();
    print(Area(poly2));
    print_newline();
    print(Area(poly3));
    print_newline();
    print(Area(poly4));
    print_newline();
    print("Perimeters ");
    print(Perimeter(poly1));
    print_newline();
    print(Perimeter(poly2));
    print_newline();
    print(Perimeter(poly3));
```

```
    print_newline();
    print(Perimeter(poly4));
    print_newline();
}

!!inter_area.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    point p1;
    point p2;
    point p3;
    polygon poly;
    l1=[[10;10],[20;20]];
    l2=[[0;0],[20;1]];
    l3=[[15;20],[20;0]];
    Draw(l1);
    Draw(l2);
    Draw(l3);
    (!display the information of the three lines!)
    display(l1);
    display(l2);
    display(l3);
    p1=l1^l2; !!p1 is the intersect point of l1 and l2
    p2=l2^l3; !!p2 is the intersect point of l2 and l3
    p3=l3^l1;   !!p3 is the intersect point of l1 and l3

    poly=[p1,p2,p3]; !! poly is the intersect area of the three lines
    print("the area of the intersection area is ");
    print(Area(poly)); !!print the area of the intersection part of the three
lines
    print_newline();
}

!!same_height_triangle.alg
def int main()
{
    polygon triangle;
    int i;
    int j;
```

```
    int k;
    k=-10;
    j=20;
    i=2;
    triangle=[[k;0],[0;j],[i;0]];
    for (i=2;i<50;i=i*2)
    {
        triangle=[[k;0],[0;j],[i;0]];
        print(Area(triangle));
        print_newline();
        Draw(triangle);
    }
}

!!similar_triangle.alg
def int main()
{
    polygon triangle;
    polygon triangle1;
    int i;
    int j;
    i=2;
    j=1;
    triangle=[[0;0],[i;0],[0;j]];
    triangle1=triangle;
    while(i<50)
    {
        Draw(triangle1);
        print(Area(triangle1));
        print_newline();
        i=i*2;
        j=j*2;
        triangle1=[[0;0],[i;0],[0;j]];

    }
    if(triangle1~triangle)
        {print("similar");
        print_newline();}
}

!!draw1.alg
def int main()
```

```
{
    ellipse e1;
    ellipse e2;
    line l1;
    line l2;
    l1=[[1;1],[2;2]];
    l2=[[1;-1],[2;-2]];
    e1={[-14.14;0],10,10};
    e2={[14.14;0],10,10};
    Draw(l1);
    Draw(l2);
    Draw(e1);
    Draw(e2);
}
```

****

Legal Functional Test Set_2 Originally authored by Shiyao Zhu (sz2395)

****

```
!!func_decl.alg
!!move11 function to move a point by vector(1,1)
def void move11(point p)
{
    Move(p,1,1);
}

!!test global function declaration and reference
def int main()
{
    point p;
    p=[0;0];
    display(p);
    move11(p);
    display(p);
}

!!global_var_decl.alg
!!test global variable

point p;

!!move11 function to move a point by vector(1,1)
```

```
def void move11()
{
    Move(p,1,1);
}

!!test global function declaration and reference
def int main()
{
    p=[0;0];
    display(p);
    move11();
    display(p);
}

!!area_compare.alg
def int main()
{
    polygon poly1;
    polygon poly2;
    ellipse e1;
    ellipse e2;
    poly1=[[0;0],[0;1],[1;1],[1;0]];
    poly2=[[0;0],[1;1],[1;0]];
    e1={[0;0],1,1};
    e2={[1;1],2,2};
    if(poly1~~>poly2)
    {print("poly1 has area larger than or equal to poly2");
    print_newline();}
    else
    {print("poly1 has area less than poly2");}
    if(e1~~<e2)
    {print("e1 has area less than or equal e2");
    print_newline();}
    else
    {print("e1 has area larger than  e2");}


}

!!congruent.alg
def int main()
{
```

```
    polygon poly1;
    polygon poly2;
    polygon poly3;
    polygon poly4;
    poly1=[[0;0],[0;1],[1;1],[1;0]];
    poly2=[[0;0],[1;1],[1;0]];
    poly3=[[0;0],[0;1],[1;1]];
    poly4=[[0;0],[0;2],[2;2],[2;0]];
    if(poly1~=poly2)
    {print("poly1 is congruent to poly2");
    print_newline();}
    if(poly1~=poly4)
    {print("poly1 is congruent to poly4");
    print_newline();
    }
    if(poly2~=poly3)
    {print("poly2 is congruent to poly3");
    print_newline();
    }
}
```

```
!!line_intersect.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    l1=[[0;0],[1.1;1.1]];
    l2=[[0;1],[1;2]];
    l3=[[0;0],[0;1]];
    {print("l1 and l2 ");
    print_newline();
    display((l1^l2));
    }
    if(l2//l3)
    {
    print("l3 and l2");
    print_newline();
    display((l2^l3));
    }
}
```

```
!!line_para.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    l1=[[0;0],[1.1;1.1]];
    l2=[[0;1],[1;2]];
    l3=[[0;0],[0;1]];
    if(l1//l2)
    {print("l1 is parallel with l2");
    print_newline();
    }
    if(l2//l3)
    {
    print("l3 is parallel with l2");
    print_newline();
    }
}

!!perimeter_compare.alg
def int main()
{
    polygon poly1;
    polygon poly2;
    ellipse e1;
    ellipse e2;
    poly1=[[0;0],[0;1],[1;1],[1;0]];
    poly2=[[0;0],[1;1],[1;0]];
    e1={[0;0],1,1};
    e2={[1;1],2,2};
    if(poly1>>=poly2)
    {print("poly1 has perimeter larger than or equal to poly2");
    print_newline();}
    else
    {print("poly1 has perimeter less than poly2");}
    if(e1<<=e2)
    {print("e1 has perimeter less than or equal e2");
    print_newline();}
    else
    {print("e1 has perimeter larger than  e2");}
```

```
}


!!relation.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    ellipse e1;
    ellipse e2;
    ellipse e3;
    l1=[[0;0],[1.1;1.1]];
    l2=[[-1;0],[0;5]];
    l3=[[0;0],[0;1]];
    e1={[1;1],1,1};
    e2={[1;1],1,2};
    e3={[3;1],1,1};
    print("l1 and e1 ");!!secant
    print(l1|-e1);
    print_newline();
    print("l3 and e1 ");!!tangent
    print(l3|-e1);
    print_newline();
    print("l3 and e2 ");!!tangent
    print(l3|-e2);
    print_newline();
    print("l2 and e2 ");!!separate
    print(l2|-e2);
    print_newline();
    print("e3 and e1 ");!!circumscribe
    print(e1|-e3);
    print_newline();
}

!!similar.alg
def int main()
{
    polygon poly1;
    polygon poly2;
    polygon poly3;
```

```
    polygon poly4;
    poly1=[[0;0],[0;1],[1;1],[1;0]];
    poly2=[[0;0],[1;1],[1;0]];
    poly3=[[0;0],[0;1],[1;1]];
    poly4=[[0;0],[0;2],[2;2],[2;0]];
    if(poly1~poly2)
    {print("poly1 is similar to poly2");
    print_newline();}
    if(poly1~poly4)
    {print("poly1 is similar to poly4");
    print_newline();
    }
    if(poly2~poly3)
    {print("poly2 is similar to poly3");
    print_newline();
    }
}

!!string.alg
def int main()
{
    string str1;
    string str2;
    string str3;
    str1="abc";
    str2="bcd";
    str3=str1;
    if(str1:=str2)
    {print("str1 is the same with str2 ");print_newline();}
    if(str1:=str3)
    {print("str1 is the same with str3 ");print_newline();}
    if(str3:=str2)
    {print("str3 is the same with str2 ");print_newline();}
}

!!input_for.alg
def int main()
{
    int i;
    line l1;
    line l2;
    i=0;
```

```
    for(i=0;i<5;i=i+1)
    {
        l1=[[1;1],[2;2]];
        l2=[[1;2],[2;i]];
        if(l1//l2)
        {
        print(i);
        print_newline();
        print("l1 parallel l2");
        print_newline();
        display(l2);
        }
    }

}

!!input_if.alg
def int main()
{
    line l1;
    ellipse e1;
    l1=[[1;0],[1;1]];
    e1={[0;0],1,1};
    if((l1|-e1):="tangent")
        print("the line and ellipse are tangent");
}

!!input_while.alg
def int main()
{
    int i;
    line l1;
    line l2;
    i=0;
    while(i<1000)
    {
        i=i+1;
        l1=[[1;1],[2;2]];
        l2=[[2;2],[3;i]];
        if(l1//l2)
        {
        print(i);
```

```
        print_newline();
        display(l2);
        print("l1 parallel l2");
        done;
        }
    }
}
```

****

Legal Functional Test Set_3 Originally authored by Qiang Deng (qd2114)

****

```
!!arithmetic.alg
def int main()
{
    int i;
    int j;
    float f1;
    float f2;
    i=1;
    j=-2;
    f1=1.5;
    f2=-1.5;
    print(i+j);
    print_newline();
    print(i-.j);
    print_newline();
    print(i*j);
    print_newline();
    print(i/j);
    print_newline();
    print(i%j);
    print_newline();
    print(f1+f2);
    print_newline();
    print(f1-.f2);
    print_newline();
    print(f1*f2);
    print_newline();
    print(f1/f2);
    print_newline();
}
```

```
!!comment.alg
def int main()
{
    (!print("testcomment");!)
    !!print("testcommentinline");
}


!!func_decl.alg
def void move11(point p)
{
    Move(p,1,1);
}

def int main()
{
    point p;
    p=[0;0];
    display(p);
    move11(p);
    display(p);
}


!!input_builtin.alg
def int main()
{
    line l1;
    ellipse e1;
    line l2;
    polygon poly1;
    point p1;
    float f;
    p1=[1,2];
    p2=[1,3];
    l2=[[1,2],[3,3]];
    l1=[[1,1],[1,2]];
    e1=[[1,1],2,1];
    poly1=[[1,1],[2,2],[3,3],[5,6],[7,8]];
    display(l1);
    Move(l1,1,2);
```

```
    display(l1);
    display(e1);
    Move(e1,1,2);
    display(e1);
    display(p1);
    Move(p1,1,2);
    display(p1);
    display(poly1);
    Move(poly1,1,2);
    display(poly1);
    Area(poly1);
    Area(e1);
    Perimeter(poly1);
    Perimeter(e1);
    f=getAngle(l1,l2);
    f=getDis();
}
```

!!input_display.alg

```
def int main()
{
    line l1;
    ellipse e1;
    polygon poly1;
    point p1;
    p1=[1,2];
    l1=[[1,1],[1,2]];
    e1=[[1,1],2,1];
    poly1=[[1,1],[2,2],[3,3],[5,6],[7,8]];
    display(l1);
    display(e1);
    display(p1);
    display(poly1);
    Area(poly1);
    Area(e1);
    Perimeter(poly1);
    Perimeter(e1);
}
```

!!input_logic.alg

```
def int main()
{
    boolean a;
    line l1;
    line l2;
    line l3;
    line l4;
    (!l1 is parallel with l2, l1 is not parallel with l4!)
    l1=[[1;1],[2;2]];
    l2=[[1;2],[2;3]];
    l3=[[1;1],[2;2]];
    l4=[[1;2],[2;2]];
    (!this should be false!)
    if((l1//l2)and(l2//l4)) {print("yes1");print_newline();}
    (!this should be true!)
    if((l1//l2)or(l2//l4)){print("yes2");print_newline();}
    (!this should be false!)
    a=not(l1//l2);
    if(a){print("yes3");print_newline();}
}
```

```
!!input_testparser.alg
def int main()
{
    int a;
    int b;
    int c;
    ellipse e1;
    ellipse e2;
    string str;
    e1=[[1,2],[3,4]];
    e2=[[2,2],[3,4]];
    a=1;
    b=1;
    str="a b";
    c=a-.b;
    c=a+b;
    e1~e2;
    e2~=e1;
    e2~>e1;
}
```

```
!!input_algo.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    point p1;
    point p2;
    point p3;
    polygon p1;
    l1=[[1,1],[2,2]];
    l2=[[0,0],[2,0]];
    l3=[[2,2],[2,0]];
    p1=l1^l2;
    p2=l2^l3;
    p3=l3^l1;
    l1=[p1,p2];
}
```

```
!!input_area.alg
def int main()
{
    line l1;
    line l2;
    line l3;
    point p1;
    point p2;
    point p3;
    polygon poly;
    ellipse e;
    l1=[[1,1],[2,2]];
    l2=[[0,0],[2,0]];
    l3=[[2,2],[2,0]];
    display(l1);
    display(l2);
    display(l3);
    p1=l1^l2;
    p2=l2^l3;
    p3=l3^l1;
    display(l1);
```

```
    display(l2);
    display(l3);
    l1=[p1,p3];
    poly=[p1,p2,p3];
    print(Area(poly));
    e=[p1,2,3];
    display(l1);
    display(poly);
    display(e);
    print_newline();
    print(Area(e));
}


!!logic.alg
def int main()
{
    boolean a;
    line l1;
    line l2;
    line l3;
    line l4;
    (!l1 is parallel with l2, l1 is not parallel with l4!)
    l1=[[1;1],[2;2]];
    l2=[[1;2],[2;3]];
    l3=[[1;1],[2;2]];
    l4=[[1;2],[2;2]];
    (!this should be false!)
    if((l1//l2)and(l2//l4)) {print("yes1");print_newline();}
    (!this should be true!)
    if((l1//l2)or(l2//l4)){print("yes2");print_newline();}
    (!this should be false!)
    a=not(l1//l2);
    if(a){print("yes3");print_newline();}
}
```

****

Semantic Check Test Set_1 Originally authored by Ainur Rysbekova (ar3005)

****

!!Sorry there is something wrong with output_1.alg and output_2.alg…

!!output_3.alg

```
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
int b;
int foo (int a,int b)
{int a;
polygon * b;
return a;
}
int main ()
{
return 0;
}
```

!!output_4.alg

```
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
int b;
int foo (int a,int b)
{int a;
polygon * b;
return a;
}
int main ()
{
return 0;
}
```

!!output_5.alg

```
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
int b;
```

```
int foo (int a,int b)
{int a;
polygon * b;
return a;
}
int main ()
{
return 0;
}
```

!!output_6.alg

```
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
int b;
int foo (int a,int b)
{int a;
polygon * b;
return a;
}
int main ()
{
return 0;
}
```

!!output_7.alg

```
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
int b;
int foo (int a,int b)
{int a;
polygon * b;
return a;
}
int main ()
{
return 0;
```

```
}
```

```
!!output_8.alg
#include "main.h"
#include "stdio.h"
#include"string.h"
int a;
ellipse * foo (ellipse * e3)
{
return Move(*e3,2,1);
}
int main ()
{ellipse * e1;
ellipse * e2;
e2=foo((e1));
return 0;
}
```

****

Semantic Check Test Set_2 Originally authored by Qingye Jiang (qj2116)

****

```
!!test_binop_wrong_type.alg
(!In this code, there is a wrong binop expression with wrong types!)
(!The result is exception Failure("Illegal type used in a binop expression")!)

int a;
int b;

def int main(ellipse w, polygon p1)
{
     boolean f;
     f = a ~> w;
     return 0;
}
```

```
!!test_binop_wrong_type_2.alg
(!In this code, we test another case of wrongly use of binop operation!)
(!The result is exception Failure("Illegal type used in a binop expression")!)
```

```
ellipse a;
ellipse b;
polygon p1;

def int main()
{
     p1 =  (a == b);
     return 0;
}
```

!!test_built_in_function.alg

```
(!In this code, we will test the built-in function Area with wrong parameter
list!)
(!The result is exception Failure("The numeber of parameters given when calling
function Area is wrong!")!)

int a;
int b;
 def int main()
{
    ellipse e1;
   float result;
   result =  Area(a,e1);
   return result;
}
```

!!test_built_in_function2.alg

```
(!In this code, we test another built-in function Move!)
(!The result is exception Failure("Wrong usage of function Move")!)
int a;
ellipse e1;
polygon p1;
def int main()
{

   Move(e1,p1);
  return 0;
}
```

!!test_dup_function.alg

```
(!This code includes an error of two functions with the same name !)
(!The result is exception Failure("Function whose name is foo has been defined
more than once") !)

int a;
int b;

def int foo(ellipse e1, ellipse e2)
{
     return a;
}

def polygon foo()
{
   polygon temp;
   return temp;
}

def int main()
{
   return 0;
}
```

```
!!test_dup_global_var.alg
(! This code redefined the global variables !)
(! The result is exception Failure("Global Variable has been redefined!")!)
int a;
ellipse a;
float b;

def int main()
{
   int i;
   return 0;
}
```

```
!!test_dup_local_var.alg
(! There is duplicated local variables in this code!)
(! The result is exception Failure("Duplicate parameter a in function foo")!)
int a;
```

```
int b;

def int foo(int a, int b)
{
   int a;
   polygon a;
   return a;
}

def int main()
{
  return 0;
}
```

!!test_dup_para_var.alg

```
(! This code has duplicated parameters in function's definition!)
(! The result is exception Failure("Duplicate parameter in function foo")!)
int a;
int b;

def void foo(int a, int b, ellipse a)
{
     print(a);
}

def int main()
{
   int i;
   return 0;
}
```

!!test_for_loop.alg

```
(!In this code, in the for loop, there are not enough expression for this loop!)
(!The result is exception Parsing.Parse_error!)
int a;
def int main()
{
   int i;
   for(i = 0;i<1)
   {
```

```
        print(i);
    }
  return 0;
}
```

!!test_function_wrong_return.alg

```
(!This codes include a function whose return type is wrong!)
(!The result is exception Failure("Unmatched return type with function's
defination!")
!)
int a;

def ellipse foo(int b, ellipse e1)
{
     b = 2;
     return b;
}

def int main()
{
     ellipse e2;
     ellipse e3;

     e3 = foo(2, e2);
    return 0;
}
```

!!test_no_main_func.alg

```
(!In this code, there is no main function!)
(!THe result is exception Failure("No Main Function exist!")!)

int a;
int b;
def int foo(int a,int b)
{
   ellipse e1;
   ellipse e2;
   if(e1 ~> e2)
   {
```

```
     a=5;
   }
   return (a+b);
}
```

!!test_while_loop.alg

```
(! This code tests the while loop, and in the while(expr), we will make expr
invalid!)
(! The result is exception Parsing.Parse_error!)
int a;
def int main()
{
    int i;
    while(i;)
    {
      i = i+1;
   }
  return i;
}
```

!!test_wrong_statement.alg

```
(! This code has a wrong statement in the body of function foo!)
(! The result is exception Failure("There is a statement in function foo which
is not a function call or assignment, which is invalid!")!)

ellipse e1;

def void foo(ellipse e1, polygon p1)
{
   line l1;
   line l2;
   e1 = {[3;2],2,2};
   l1 // l2 ;  (!This is an invalid statement!)
}

def int main()
{
   return 0;
}
```

```
(!In this code, a function is defined but is wrongly used afterwards!)
(!The result is exception Failure("The number of parameters given when calling
function foo is wrong!")!)
int a;
def ellipse foo(ellipse e3)
{
  return Move(e3,2,1);
}
```

!!test_wrong_usage_of_function.alg

```
def int main()
{
    ellipse e1;
    ellipse e2;

    e2 = foo(e1,2);
    return 0;
}
```