COMS 4115: Programming Languages and Translators

SBML: Shen Bi Ma Liang

Chinese Character: 神筆馬良

English Translation: Magic Pen Boy

Project Report

Bin Liu (bl2329@columbia.edu)

Yiding Cheng (yc2434@columbia.edu)

Hao Li (hl2489@columbia.edu)

Wenhan Zhang (wz2174@columbia.edu)

Dec. 19, 2008

CONTENTS

CHAPTER 1 4

1 INTRODUCTION 4

1.1 MOTIVATION 4

1.2 IMPLEMENTATION PLAN 4

1.3 EXAMPLE OF SBML 4

CHAPTER 2 6

2 LANGUAGE TUTORIAL 6

2.1 EXAMPLE 1 6

2.2 EXAMPLE 2 7

2.3 EXAMPLE 3 8

CHAPTER 3 10

3 LANGUAGE MANUAL 10

3.1 INTRODUCTION 10

3.2 LEXICAL CONVENTIONS 10

3.2.1 Keywords 10

3.2.2 Identifiers (Names) 10

3.2.3 Comments 11

3.2.4 Constants 11

3.2.5 String 11

3.3 DATA TYPES 11

3.4 OPERATORS AND SPECIAL CHARACTERS 12

3.4.1 Operators 12

3.4.1.1 Arithmetic operator 12

3.4.1.2 Comparison operator 12

3.4.1.3 Logic operator 12

3.4.1.4 Special Characters 13

3.5 FUNCTIONS 13

3.5.1 User Defined Functions 13

3.5.2 Function definition syntax 13

3.5.3 Built-in Functions 13

3.6 SCOPE RULES 14

3.7 EXPRESSIONS 14

3.7.1 Primary Expressions 14

3.7.2 Digit(s) 14

3.7.3 Letter 14

3.7.4 Identifier 14

3.7.5 Binary_operation *14*
3.7.6 Argument_list *15*
3.7.7 Attribute_list *15*
3.7.8 Set and get value *15*
3.8 STATEMENTS 15
3.9 SAMPLE CODE (SAMPLE CODE) 16

CHAPTER 4 17

4 PROJECT PLAN 17
4.1 HOW OUR GROUP DISCUSSION MAKES 17
4.2 PROJECT PROCESS 17
4.2.1 Learning basic knowledge and planning *18*
4.2.2 Specification and Development *18*
4.2.3 Testing *18*
4.3 PROJECT TIMELINE 19
4.4 ROLES AND RESPONSIBILITIES OF EACH TEAM MEMBER: 19
4.5 PROGRAMMING STYLE GUIDE USED BY THE TEAM 19
4.6 PROJECT DEVELOPMENT ENVIRONMENT 20

CHAPTER 5 21

5 ARCHITECTURAL DESIGN 21
5.1 BLOCK DIAGRAM SHOWING THE MAJOR COMPONENTS OF SBML 21
5.2 DESCRIBE THE INTERFACES BETWEEN THE COMPONENTS 21

CHAPTER 6 22

6 TEST PLAN 22
6.1 TEST VARIABLE SCOPE 22
6.2 TEST SHAPE DECLARATION 23

CHAPTER 7 24

7 LESSONS LEARNED 24

APPENDIX A: 26

SCANNER.MLL 26

APPENDIX B: 28

PARSER.MLY 28

APPENDIX C 33

AST.MLI 33

APPENDIX D: 35

RECT.MLI;CIRCLE.MLI;LINE.MLI;IMAGE.MLI;LABEL.MLI 35

APPENDIX E: 37

INTERPRETER.ML 37

Chapter 1

1 Introduction

1.1Motivation

The name, SBML, inspired by a famous Chinese fairy tale, Shen Bi Ma Liang (Chinese characters: 神筆馬良), which is the story of a boy who has a magic pen, and whatever he drew with that pen, it going to turn alive. Unfortunately, the language can not effectively turn drawings alive, but it can make human thoughts alive on webpage.

SBML is a computer language to generate webpage with versatile objects only based on the program designed by web designer. The basic idea of this language is to firstly understand the web design from the user, and then implement the demanded work via generating corresponding Javascript code, which can be run directly by Internet Explorer.

SBML actually saved web designers from spending their time to learn various web development techniques, what they only need is the language designed by us which is easily understandable, then they don't have to anything else to design the web. By using SBML, user could create fantastic web pages with various complex shapes. Also, to make the drawings alive, users are allowed to attach events to every shape they create.

1.2 Implementation Plan

In this project, our main goal is to implement a language which can help people draw what they like into web pages. In implementation, we use Object Caml as our main programming language, and its tools including Ocamllex and Ocamlyacc, which are about to be used to design the language scanner and parser, respectively.

After parsing, we translate the result into Vector Markup Language (VML), which is an XML language, used to produce vector graphics.

1.3 Example of SBML

```
int main()
{
  int i;
```

```
  int j;
  Line c =
  Line{.startx:10;.starty:20;.endx:100;.endy:100;.linecolor:"blue
  ";
  .stroke:2};
  Circle a =
  Circle{.centerx:50;.centery:60;.radius:10;.circlecolor:"red"};
  for(i=0;i<10;i=i+1)
  {c.endx = c.endx + 30;
    c.endy = c.endy +15;
   Draw c;
   }
  for(j=0;j<3;j=j+1)
  {
  a.radius = a.radius + 20;
  Draw a;
  }
}
```

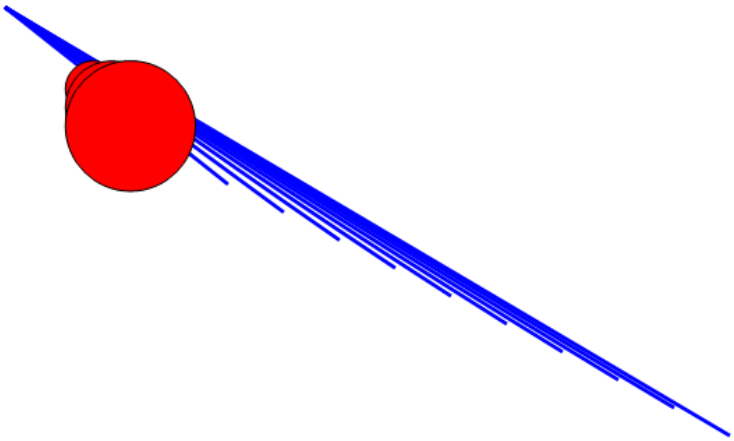


Figure 1.1: the output of the example syntax

Chapter 2

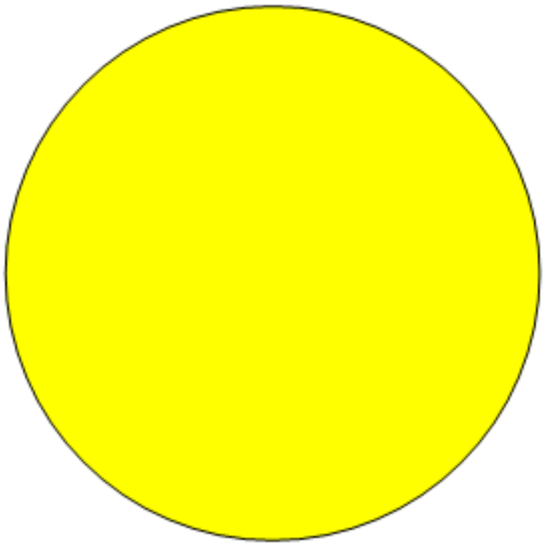
2 Language Tutorial

2.1Example 1

SBML's special data types can draw some shapes into a webpage:

```
/* to draw a yellow circle with (20,30) as the center and radius
of 200 */
int main()
{
    Circle a =
    Circle{.centerx:20;.centery:30;.radius:200;.color:"yellow"};
    Draw a;
}
```

The output is:



2.2 Example 2

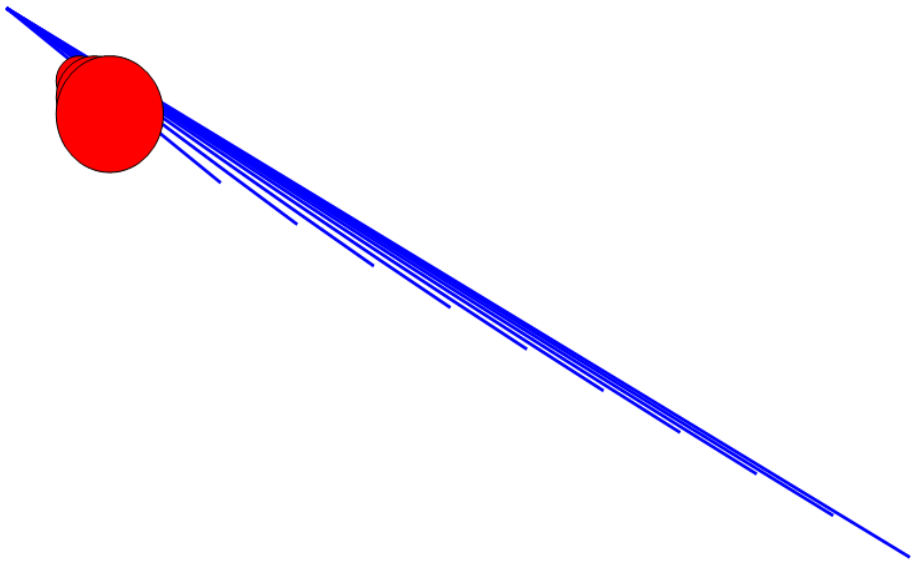
SBML's "for" expression can draw a series of complicated shapes in a webpage:

```
int main()
{
   int i;
   int j;
   Line c=Line{.startx:10;.starty:20;.endx:100;.endy:100;
            .color:"blue";.stroke:2};
   Circle a = Circle{.centerx:50;.centery:60;.radius:10;
            .color:"red"};

   for(i=0;i<10;i=i+1)
   {
   c.endx = c.endx + 50;
   c.endy = c.endy +25;
   Draw c;
   }


   for(j=0;j<3;j=j+1)
   {
   a.radius = a.radius + 20;
   Draw a;
   }
}
```

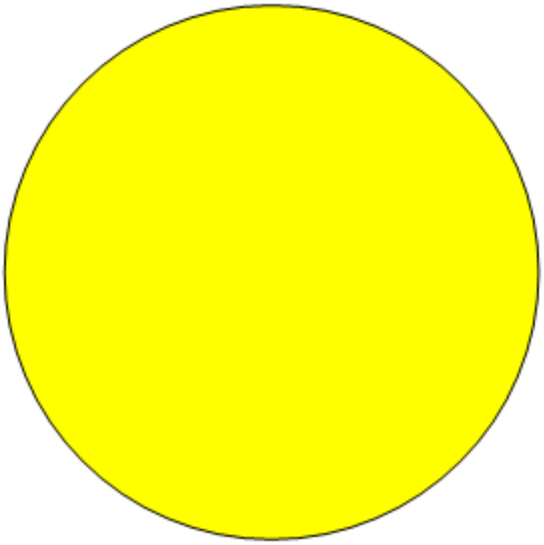
The result is



2.3Example 3

SBML's "if…else" expression:

```
int main()
{
Circle b = Circle
{.centerx:20;.centery:30;.radius:200;.circlecolor:"yellow"};
Line a = Line
{.startx:2;.endx:3;.starty:3;.endy:3;.color:"af";.name:"abc"};
    int i=1;
    if (i=1) Draw b;
    else Draw a;
}
```



Chapter 3

3 Language Manual

3.1 Introduction

SBML is a programming language intended to draw graphs on the webpage, with an additional functionality for writing and manipulating text on graphs and pictures as well. Syntax and coding style are similar to C/C++. SBML uses some built-in types each of which has properties, just like classes in C++ or Java but without build-in function. Those intuitive properties provide the user a straightforward way to manipulate the features of the text he/she wants to print.

Every SBML program must include a main function (only one function named 'main' is allowed), it could have several return types, and no arguments are allowed.

3.2 Lexical Conventions

3.2.1 Keywords

The following identifiers are reserved for use as keywords in SBML language:

RETURN IF ELSE FOR WHILE INT

RECT CIRCLE LINE LABEL STRING IMAGE

3.2.2 Identifiers (Names)

Identifiers are sequences of letters and digits. Identifiers are case sensitive. Letter should be at the beginning of the identifier.

Underline can also contain in the identifier.

3.2.3 Comments

SBML supports likely both C and C++ style comments. Whatever multi-line comments start with the characters "/*" and terminate with the characters "*/".

3.2.4 Constants

SBML only have integer constant.

An integer constant is a sequence of digits. Integers in SBML are all taken as decimal. The keyword to declare an integer is int.

3.2.5 String

In SBML, a string is a sequence of characters surrounded by double quotes “ " ”.

3.3 Data Types

In SBML, there are eight built-in data types:

Rect, Circle, Label, Line, Image, String, Int

Types	Description
Label	A built-in type which expresses a data type of a sequence of characters (or a class used to draw a string, with 5 parameters: content, font, size, color, coordinate)
Rect	A built-in type used to draw a rectangle; each instance will take 6 parameters: left, top, width, height, color, child(the name of a instance of the type: Label)
Line	A built-in type used to draw a line; each instance will take 6 parameters: startx, starty, endx, endy, color, stroke
Image	A built-in type used to load a picture on the page; each instance will take 5 parameters: left, top, width, height, url.
Circle	A built-in type used to draw a Circle; each instance will take 4 parameters: centerx(x-coordinate of the center of the circle), centery(y-coordinate of the center of the circle), radius, fillcolor
String	A basic type that contains a sequence of characters
Int	A basic type that contains a 32 bit signed integer, with a range

	of -2147483648 to 2147483647

3.4 Operators and special characters

3.4.1 Operators

3.4.1.1 Arithmetic operator

Operator “+”

Usage example: operation_code + operation_code // add the two operation codes

Operate “-“

Usage example: operation_code - operation_code // the fore operation code subtracts the latter

Operator “*”

Usage example: operation_code * operation_code // the fore operation code multiplies the latter

Operator “/”

Usage example: operation_code / operation_code // the fore operation code divides the latter

All arithmetic operators are binary operators, among them, multiplication and divide have the same precedence which is higher than plus and minus, as the same as in basic math.

3.4.1.2 Comparison operator

In SBML, “<”, “>”, “==”, “<=”, “>=”, “!=” are defined “less than”, “greater than”, “equal to”, “equal or lesser than”, “equal or greater than”, “unequal to” separately as comparison operators, which are used between two operation codes as the same as the arithmetic operators.

3.4.1.3 Logic operator

In SBML, “||”, “&&” are defined “or”, “and” separately as logic operators, which are used between two operation codes as the same as the operators mentioned above.

3.4.1.4 Special Characters

In SBML, there are some special characters with functional significance:

Special Characters	Usage	Example
“ (” & “) ”	Expression grouping; also used as a function parameter list delimiter	Foo(int bar, int tar)
“ { ” & “ } ”	Delimit list and function body	for (){ }
‘ " ’	String literal	string str = "hello world";
“ , ”	Argument list separator	Line{.startx:10; starty:20; endx:30; endy:40};
“ : ”	Assign value to a member type	Circle{.centerx:10;centery: 10;};
“ ; ”	Line break	A = 6;
“ . ”	Get the attribute belonged to an identifier	Rect.width = 5

3.5 Functions

3.5.1 User Defined Functions

SBML allows users to create their own functions to realize the specific operations desired by their on-going design projects. Also the user-defined functions could be stored in a shared library in order to be accessed and utilized by multiple users or for multiple projects.

3.5.2 Function definition syntax

function_declaration: type identifier'('(declaration ','declaration)* | empty)')' '{' statements (return_statement semicolon)? '}'

3.5.3 Built-in Functions

Usage example: Draw p; /* usage: to directly dump shape to the page */

3.6 Scope Rules

In SBML, the scope rules are very similar to those in C or C++. A variable or function is not available until it is declared. Functions cannot be nested and therefore cannot be overridden after they are declared. They are declared in the global scope and available until the program completes execution. A variable is available until the end of the block, in which it was declared, is reached. In the case of variable declared in the global scope, the variable only goes out of scope on program termination. Nested blocks can access variables defined in parent blocks. If a new variable is declared in a nested block with the same name as a variable from a parent block, said variable was overridden. The nested block will then only have access to the new variable from point of declaration. When the block is closed, the original variable will return to scope.

3.7 Expressions

3.7.1 Primary Expressions

Primary expressions in SBML involving right_value (identifier, digits, string, binary_operation), declaration, function_call.

3.7.2 Digit(s)

The expression digit is defined to include ‘0’-‘9’, and digits is the closure of digit, which denoted as digit digit *

3.7.3 Letter

The expression letter includes the lower case letters ‘a’-‘z’ and the upper case letters ‘A’-‘Z’ in English alphabet.

3.7.4 Identifier

The expression Identifier is the concatenation of the expression letter and the closure of digit or letter, which is denoted by letter(digit | letter)*.

3.7.5 Binary_operation

The expression binary_operation has the form of the concatenation of a digits or identifier with a operator followed by another digits or identifier, denoted by (digits | identifier)(arithmetic operator | comparison_operator | logical_operator)(digit | identifier)

3.7.6 Argument_list

Argument is separated by “,” argument_list : empty | identifier (',' identifier)*;

3.7.7 Attribute_list

attribute : type ':' right_value; attribute_list : (attribute)* ;

3.7.8 Set and get value

set_value : identifier'.'type;

3.8 Statements

In SBML, statements are executed sequentially, the following are the types of statements:

```
   Statement :declaration|function_call|assignment|binary_operation
|build_in_function|if_statement|for_statement;statments:(statement
semicolon)*;
   program : ( statements | function_declaration )*;
   l.If statement:

   if_statement : 'if' '(' binary_option ')' '{' statements '}' ‘else’
‘{’ Statements ‘}’
   l.Iteration statement:

   for_statement : 'for' identifer '=' digits 'to' digits '{' statements
'}'
   l.Declaration statement:

   type identifier;
   l.Assignment statement:

   ( identifier.attribute | type identifier | identifier ) = (string
| digits | identifier | binary_operation)
   l.Return statement:

   return_statement : 'return' identifier | digits | string |
binary_operations;
   l.Function call statement:
```

```
function_call : identifier '(' argument_list ')';
```

3.9 Sample Code (sample code)

This sample code is going to generate a rectangle with a label “hello world” in the page.

```
Start Page page;
string str = "hello world";
Born Color color = Color{R:0f;G:57;B:89;};
Born Label label = Label{Content:str;Color:color;};
Born Rect rect = Rect{Label:label;Color:color;Width:100;Height:100;};
Draw rect > page; Mem page;
```

Chapter 4

4 Project Plan

4.1 How our Group Discussion makes

To ensure our project could be finished on time, we established a unique way to maximize our productivity. As a group formed of four people, our first goal is to achieve best allocation of human resource. At the staging phase of the entire project, we held our meetings twice a week, to let all of us to fully express the picture of this project in our minds, and we share our ideas and thoughts, with the discussion of the possibility of realization for each detail.

After decided SBML, our four people team separated into two, and each of the sub-team meet regularly, and we held a weekly meeting of the whole group to share our thoughts and merge the progression of each 2 person team.

The progress of the born of SBML had gone through several steps:

Step 1: Two sub-teams work individually to design the scanner and the parser, with the syntax and the functionalities we would like to realize in our later implementations, and after the design, we discuss over our two versions, merge them to build the blue print of the final scanning and parsing rules for our SBML language.

Step 2: After the syntax had been fixed, the group was separated again into two sub-teams with two members for each. Team A in charge for writing the interpreter to realize all previously imaginary functionalities, and Team B is in charge of debugging and testing, also the draft of the report and the presentation. The work for both sub-teams are strictly synchronized, we firstly established a CVS server to let the programming to be consistent, secondly, we use Google Documents to maintain our documents and track the changes, thirdly, traditionally, we use email lists and everyone reply to all to express himself concerning the project. Besides, the work for both sub-teams and all team members are guided by the timeline we confirmed at the beginning of the project, which is shown in section 4.x.

4.2 Project Process

There are four stages in the project process: planning, specification, development, and testing.

4.2.1 Learning basic knowledge and planning

Learning basic knowledge of PLT and Ocaml language: The first stage was from the first lecture to the release of the proposal. Each member learns knowledge of PLT and Ocaml language to get the general idea of the parser and the main character of Ocaml language. This stage is important for us to make sure what we can do and should do in the project. The idea of SBML was also created during this time. But it has been modified a lot when we see problems or have new ideas.

Planning: After two heat discussions, the entire group members agree on the proposal of SBML and we begin to discuss the detail of our SBML language. Many good ideas exist but some of them are undoable and are rejected by the group members. Planning stage was from the proposal till the release of the Language Reference Manual. Team members give ideas on implementation, details of functionalities and dispatch of the work. Everybody takes one section of the LRM so that he is more familiar with this part. Then we exchanged the writing to other members for error checking. It also helped each member has a clue of the picture of the whole project. After the LRM is finished, everybody is familiar with the whole project and specialized in several aspects of the project.

4.2.2 Specification and Development

Development Stage: This part is the most important one. It's from LRM to the final exam. The major aspects of the project include scanner, parser, AST, interpreter, code generation and documentation. We first divided the team into two groups (one working on scanner/AST, the other is on parser) so that everybody is not working independently. And after the scanner/AST and scanner were finished, two members shifted to work on interpreter and the other two member starts to work on test cases and documentation.

After the final exam till the submission day, all group members meet more frequently to keep an eye on others' progress. Test case group tell the interpreter group the bug they have found and the interpreter group keep updating their change towards the syntax. Demo codes and documentations were finally finished.

4.2.3 Testing

Our testing concludes three kinds of testing: accuracy testing and failure testing.

Accuracy testing: We use the correct syntax which SBML supports to write a lot of test cases to test whether SBML can give the desired results.

Failure testing: We use the forbidden syntax which SBML do not support to write lots of wrong test case to test whether SBML can give the exception.

We will show our test case in Chaper 6.

4.3 Project Timeline

September 3rd: Project team formation

September 18th: Team meets and decides to take SBML as the project

September 22nd: Proposal completion

September 24th: Proposal due

September 27th: Meet and discuss the detail of SBML functionality

October 1st: Discussion and finalizing grammar

October 8th: LRM division

October 22nd: LRM due. Separate the team into two group:

Scanner/Ast group and parser group.

November 3rd - November 15th: Two groups communicate regularly to make sure they

are in the same direction.

November 17th: finish AST and scanner. Discuss the problems of parser.

November 25th: finish parser and discuss about interpreter. Interpreter starts.

December 8th: Final exam

December 15th: Interpreter finishes and starts testing and final report documentation.

December 17th: Finish final report. Project finished

December 18th: Prepare for presentation

December 19th: Final project due and presentation

4.4 Roles and responsibilities of each team member:

Bin Liu:	Problem shooter, Scanner/ast, interpreter, documentation
Yiding Cheng:	Leader, scanner/ast, interpreter,VML
Hao Li:	Parser, test case, documentation
Wenhan Zhang:	Parser, test case, documentation

4.5 Programming style guide used by the team

The Ocaml coding style is according to the MicroC example provided by

Professor Edwards.

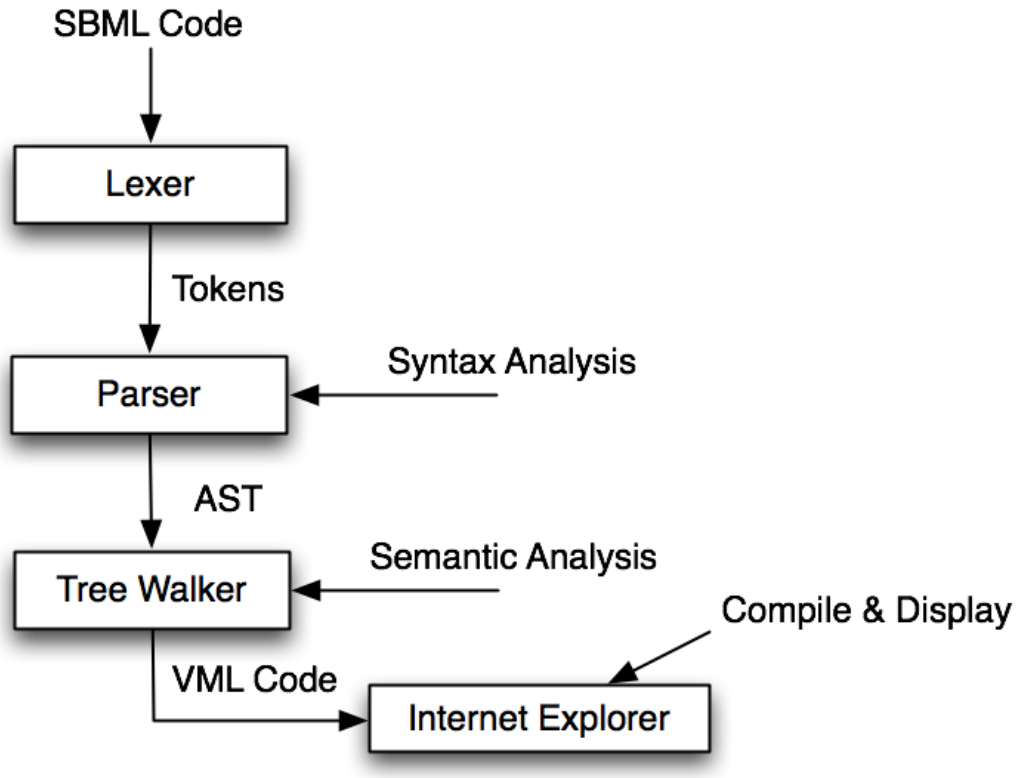
4.6 Project Development Environment

Unfortunately, there are not good IDE support of OCaml project. Therefore, we primarily use VIM under Linux environment to write OCaml code and use Makefile to compile the whole project. To resolve the dependency, we use CVS to synchronize the code written by each developers. It works fine for small project like OCaml.

Chapter 5

5 Architectural Design

5.1 Block diagram showing the major components of SBML



5.2 Describe the interfaces between the components

AST have all the function definition, to get all the tokens from the scanner, and parse them to separate code intervals. Then Interpreter translates every code interval. After the program finishes, take a look at the global table and send all kinds of maps to VML.

Chapter 6

6 Test plan

6.1 Test variable scope

```
int a;
string b;
int a()
{
    return 15;
}

int gcd(int x, int y)
{
  while (x != y) {
    if (x > y) x =x-y;
    else y =y- x;
  }
  return x;
}

int main()
{
    a=10;
    Print(a);
    int a;
    a=a+5;
    Print(a);
    string b="ab";
    Print(b);
    Print(a());
    Print(gcd(25,35));
}
int c;
```

6.2 Test Shape declaration

```
int main()
{
  int i=0;
  Image a = Image{.left:5;.top:5;.width:5;.height:5;
         .url:"abc.com";.name:"iamImage"};
  Circle b= Circle{.centerx:1;.centery:2;.radius:2;
        .color:"abc";.name:"abc"};
  Label c = Label{.left:1;.top:2;.width:3;.height:65;
        .content:"iamlabel";.name:"abc"};
  Line d = Line{.startx:2;.endx:3;.starty:3;.endy:3;
           .color:"af";.name:"abc"};
  Rect e = Rect{.left:11;.top:12;.color:"red";.width:14;
  .height:13;.name:"test"};
  string f="abc";
  for(i=0;i<10;i=i+3)
       {Draw a;
       Print (a);
       Draw b;
       Print(b);
       Draw c;
       Print(c);
       Draw d;
       Print(d);
       Draw e;
       Print(e);
       Draw f;
       Print(f);
       }
}
```

Chapter 7

7 Lessons Learned

Hao Li:

We have learned a lot during the project. First of all, it is about project management. Every team should set up a reasonable time line for the project and all the members should strictly follow the time line since that everybody finishes his work and waiting for the delay work of the last one is a waste of time. What is more important, the team should be divided into sub-teams to handle different parts of the project. It can clearly improve the efficiency of the project process. For example, in our project, we divide the group into two groups, one is responsible for the scanner and AST and the other group parser. It really works a lot compared with four members do the work separately and meet to find out the best one. Secondly, communications inside the group is of great importance. Developing a compiler is a team work and corporation with others. Get to know what the whole project is going on is quite important since every part of the project has relationship with the others so we should update our part frequently in order to make sure the project is in the right track. Thirdly, we use VML which is support by Microsoft to draw multiple shapes on the net. The project makes familiar with VML. Finally, we have learned from the project that testing procedure is as important as development. Accuracy testing and failure testing help us to find out the bugs in the project.

Wenhan Zhang:

To study a new language is interesting, and to study how to design my own language is really fantastic. Before I took the course, I never thought I could handle this, you know, a new language, even in such a powerful group. Now, after these lectures, I know any large software system is much easier to understand and implement if I can take care with the fundamental abstractions and interfaces. Breaking the compiler into this many pieces allows for reuse of the components and makes anything easier.

Then, I want to talk about the Ocaml language which we use to implement our language. It is really a suitable for develop the compilers because of its succinct character. For example , our interpreter has 700 lines of code. If we use java, we will have to use at least two thousand lines of code to implement it. It’s amazing.

Yiding Cheng:

It’s not easy to coordinate the whole project especially when people in the same group are implementing separate modules. VML is fun, I will use Javascript if we have

time.

Bin Liu:

It’s hard to follow the schedule. Ocaml is powerful, but it can be annoying since mastering the datastructure and handling the type match is not easy to go along with.

I should have kept the code more succinct if I know “module” better. It’s really fun to implement a compiler.

Appendix A:

Scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }
| "/*"     { comment lexbuf }
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ':'     { COLON }
| '"'     { QUOT  }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"      { AND }
| "||"      { OR  }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"    { BOOL }
| "string" { STRING }
| "Color"  { COLOR }
```

```
| "Label"  { LABEL }
| "Rect"   { RECT }
| "Circle" { CIRCLE }
| "Image"  { IMAGE }
| "Line"   { LINE }
| "Draw"   { DRAW }
| "Print" { PRINT}
| '.'['a'-'z']+  as lxm  {MEM_TYPE(lxm) }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| '"'[^'"']*'"' as lxm { STRING_VALUE(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

Appendix B:

Parser.mly

```
%{ open Ast %}
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA POINT COLON QUOT
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT
%token DRAW PRINT
%token RECT CIRCLE LINE LABEL STRING IMAGE COLOR BOOL
%token <string> STRING_VALUE
%token <int> LITERAL
%token <string> ID
%token <Ast.expr_type> BLTIN_TYPE
%token <string> MEM_TYPE
%token EOF

%nonassoc NOELSE
%nonassoc ELSE

%left ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left START DRAW MEM

%start program
%type <Ast.program> program
%%

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

fdecl:
   INT ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
```

```
                { {
            return_type = "int";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
  | STRING ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "string";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
    | IMAGE ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "image";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
    | LABEL ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "label";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
  | LINE ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "line";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
    | RECT ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "rect";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }
  | CIRCLE ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
                { {
            return_type = "circle";
            fname   = $2;
                      formals = $4;
                      body    = List.rev $7 } }


formals_opt:
```

```
    /* nothing */       { [] }
  | formal_list         { List.rev $1 }
formal_list:
       formal_def               {  [$1]    }
       |formal_list COMMA formal_def      { $3 ::$1}

formal_def:
       INT ID        {( "int", $2)}
       | STRING ID       {( "string", $2)}
       | RECT ID     {("rect", $2)}
       | IMAGE ID    {("image", $2)}
       | LINE ID     {("line", $2)}
       | LABEL ID    {( "label", $2)}
       | CIRCLE ID       {( "circle", $2)}


value_list:
    LBRACE RBRACE { [] }
    | LBRACE   attri_list RBRACE {   List.rev   $2  }

attri_list:
       member_def    {  [$1]  }
       |attri_list SEMI member_def     { $3 ::$1}
member_def:
       MEM_TYPE COLON    LITERAL    {( $1, string_of_int($3))}
       | MEM_TYPE COLON STRING_VALUE   {( $1, $3)}

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

vdecl:
  INT ID SEMI {("int", $2)}
  | STRING ID SEMI    {("string", $2)}
  | RECT ID SEMI  {("rect", $2)}
  | CIRCLE ID SEMI    {("circle", $2)}
  | LINE ID SEMI  {("line", $2)}
  | LABEL ID SEMI {("label", $2)}
  | IMAGE ID SEMI {("image", $2)}


stmt:
    expr SEMI { Expr($1) }
  | LBRACE stmt_list RBRACE { Block(List.rev $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_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | RETURN expr SEMI { Return($2) }
  | INT ID SEMI   {Define_int($2,0)}
  | STRING ID SEMI    {Define_string ($2,"")}
  | RECT ID SEMI  {Define_rect ($2,[("","")])}
  | CIRCLE ID SEMI    {Define_circle ($2,[("","")])}
  | LINE ID SEMI  {Define_line ($2,[("","")])}
  | LABEL ID SEMI {Define_label ($2,[("","")])}
  | IMAGE ID SEMI {Define_image ($2,[("","")])}
  | INT ID ASSIGN LITERAL SEMI          {      Define_int($2,$4)     }
  | STRING ID ASSIGN  STRING_VALUE SEMI      {      Define_string ($2,
$4)}
    |   RECT   ID  ASSIGN RECT   value_list SEMI      {
              Define_rect ($2, $5 )}
    |   CIRCLE ID  ASSIGN CIRCLE value_list SEMI       {
              Define_circle($2, $5)}
    |   LINE   ID  ASSIGN LINE   value_list SEMI         {
              Define_line($2, $5)}
    |   LABEL  ID  ASSIGN LABEL  value_list SEMI      {
              Define_label($2, $5)}
    |   IMAGE  ID  ASSIGN IMAGE  value_list SEMI      {
              Define_image($2, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr        { $1 }

expr:
    LITERAL         { Literal($1)    }
  | ID                    { Id($1) }
  | ID MEM_TYPE ASSIGN expr   { Member_Assign($1, $2, $4)   }
  | ID MEM_TYPE               { Member($1,$2)}
  | STRING_VALUE       { Stringvalue($1)}
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
```

```
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | ID ASSIGN expr       { Assign($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | ID LPAREN actuals_opt RPAREN    { Call($1, $3) }
  | DRAW ID    {Draw($2)}
  | PRINT LPAREN expr RPAREN{Print($3)}
actuals_opt:
    /* nothing */ { [] }
  | actuals_list { List.rev $1 }
actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

Appendix c

ast.mli

```
type expr_type = RECT | CIRCLE | LABEL | LINE | IMAGE | STRING |INT

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

type expr =
    Literal of int
  | Id of string
  | Stringvalue of string
  | Binop of expr * op * expr
  | Assign of string * expr
  | Member of string * string
  | Member_Assign of string * string * expr
  | Call of string * expr list
  | Draw of string
  | Print of expr
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Declare of expr_type * string
  | Define_int of string * int
  | Define_string of string * string
  | Define_circle of string * (string * string) list
  | Define_line of string * (string * string) list
  | Define_image of string * (string*  string) list
  | Define_rect of string * (string * string ) list
  | Define_label of string * (string * string) list
  | Return of expr

type func_decl = {
```

```
    return_type : string;
    fname : string;
    formals : (string * string) list;
    body : stmt list;
  }


type program = (string * string) list * func_decl list
```

Appendix D:

rect.mli;circle.mli;line.mli;image.mli;label.mli

rect.mli：

```
type rect =
	{	mutable rect_left : int;
		mutable rect_top : int;
		mutable rect_color : string;
		mutable rect_width :int;
		mutable rect_height :int;
		mutable rect_name : string;
	}
```

circle.mli：

```
type circle =
	{	mutable circle_centerx : int;
		mutable circle_centery : int;
		mutable circle_radius : int;
		mutable circle_color : string;
		mutable circle_name : string;
	}
```

line.mli：

```
type line =
	{	mutable line_startx : int;
		mutable line_starty : int;
		mutable line_endx : int;
		mutable line_endy : int;
		mutable line_stroke : int;
		mutable line_color : string;
		mutable line_name : string;
	}
```

image.mli:

```
type image = {
	mutable image_left : int;
```

```
        mutable image_top : int;
        mutable image_width : int;
        mutable image_height : int;
        mutable image_url : string;
        mutable image_name : string;
}
```

label.mli:

```
type label =
        {       mutable label_left : int;
                mutable label_top : int;
                mutable label_width : int;
                mutable label_height : int;
                mutable label_content : string;
                mutable label_name : string;
        }
```

Appendix E:

interpreter.ml

```
open Image;;
open Rect;;
open Circle;;
open Line;;
open Label;;
open Ast;;
open Printf;;

let file = "/var/www/index.html"
let head = "<html>
<body>
<style>v\\: * {behavior:url(#default#VML); display:inline-block}
</style>
<xml:namespace ns=\"urn:schemas-microsoft-com:vml\" prefix=\"v\" />"



(************************************************variable
definitions**********************************)

(* make name map *)
module NameMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end);;

let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644 file in
       fprintf oc "%s\n" head;;

let draw_shape id env =
    let
(_intMap,_stringMap,_lineMap,_labelMap,_circleMap,_rectMap,_imageMap,_
typeMap) = env in
    if (NameMap.mem id _rectMap)
```

```
       then
       let _ = NameMap.map (fun x->
       let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644
file in
       fprintf oc "<v:rect style=\"height: %d pt;width:%d pt\"
style=\"position:absolute;left:%d pt;top:%d pt;\" fillcolor=%s
strokecolor=\"black\" strokeweight=\"1pt\" /> \n" x.rect_height
x.rect_width x.rect_left x.rect_top x.rect_color
)_rectMap in print_endline "Drew a rect"

   else if (NameMap.mem id _labelMap)
       then let _ = NameMap.map (fun x->
       let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644
file in
       fprintf oc "<v:rect style=\"position:absolute;left: %d pt;top: %d
pt; width: %d;\" strokeweight=\"0pt\">
  <v:textbox>
    <FONT SIZE=%d>%s</FONT>
  </v:textbox>
</v:rect> \n" x.label_left x.label_top x.label_width x.label_height
x.label_content
) _labelMap in


print_endline("Drew a label")

   else if (NameMap.mem id _circleMap)
       then
       let _ = NameMap.map (fun x->
       let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644
file in
       fprintf oc "<v:oval style=\"position:absolute;left: %d pt;top:%d
pt; width: %d pt;height:%d pt\" fillcolor=%s>
</v:oval> \n" x.circle_centerx x.circle_centery x.circle_radius
x.circle_radius x.circle_color
) _circleMap in
print_endline("Drew a circle")

   else if (NameMap.mem id _lineMap)
       then let _ = NameMap.map (fun x->
       let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644
file in
       fprintf oc "<v:line from=\"%d pt,%d pt\" to=\"%d pt,%d pt\"
strokecolor=%s strokeweight=\"%dpt\" />\n" x.line_startx x.line_starty
```

```
x.line_endx x.line_endy x.line_color x.line_stroke
) _lineMap in
print_endline("Drew a line")

    else if (NameMap.mem id _imageMap)
        then let _ = NameMap.map (fun x->
        let oc = open_out_gen [Open_trunc;Open_append;Open_creat] 0644
file in
        fprintf oc "<img src=\"%s\" style=\"position:absolute;left: %d
pt;top: %d pt;\"     width = \"%d\" height = \"%d\" />\n" x.image_url
x.image_left x.image_top           x.image_width x.image_height
) _imageMap in
print_endline("Drew an image")
    else
        print_endline("wrong");;




(* define exception, to bring back the return value *)
exception ReturnException of (string * (string * string) list) *
    (int  NameMap.t * string  NameMap.t *  Line.line NameMap.t *
Label.label  NameMap.t *
    Circle.circle  NameMap.t *   Rect.rect NameMap.t* Image.image
NameMap.t * string NameMap.t);;

(* define all type name in string format *)
let int_name = "int" and string_name = "string" and label_name = "label"
    and  circle_name = "circle" and line_name = "line"
    and image_name = "image" and rect_name = "rect";;

(* image list and map, for verify member name *)
let image_member_list =
[".left";".width";".top";".height";".url";".name"];;
let create_image =
        {
                image_left = 100;
                image_top = 0;
                image_width = 0;
                image_height = 0;
                image_url = "";
```

```
            image_name = "new_image";
      };;
let image_reverse_map = NameMap.empty;;
let image_reverse_map = NameMap.add ".left" int_name image_reverse_map;;
let image_reverse_map = NameMap.add ".top" int_name image_reverse_map;;
let image_reverse_map = NameMap.add ".width" int_name image_reverse_map;;
let image_reverse_map = NameMap.add ".height" int_name
image_reverse_map;;
let image_reverse_map = NameMap.add ".url" string_name
image_reverse_map;;
let image_reverse_map = NameMap.add ".name" string_name
image_reverse_map;;

(* rect member and map, for verify member name *)
let create_rect =
      {
            rect_left = 0;
            rect_top = 0;
            rect_color = "#000000";
            rect_width = 100;
            rect_height = 100;
            rect_name = "new_rect";
      };;

let rect_member_list =
[".left";".top";".width";".height";".color";".name"];;

let rect_reverse_map = NameMap.empty;;
let rect_reverse_map = NameMap.add ".left" int_name rect_reverse_map;;
let rect_reverse_map = NameMap.add ".top" int_name rect_reverse_map;;
let rect_reverse_map = NameMap.add ".width" int_name rect_reverse_map;;
let rect_reverse_map = NameMap.add ".height" int_name rect_reverse_map;;
let rect_reverse_map = NameMap.add ".color" string_name
rect_reverse_map;;
let rect_reverse_map = NameMap.add ".name" string_name rect_reverse_map;;

(* circle member and map, for verify member name *)
let create_circle =
      {
            circle_centerx = 100;
            circle_centery = 100;
            circle_radius = 50;
            circle_color = "#000000";
            circle_name = "new_circle";
```

```
	};;
let circle_member_list =
[".centerx";".centery";".radius";".color";".name"];;

let circle_reverse_map = NameMap.empty;;

let circle_reverse_map = NameMap.add ".centerx" int_name
circle_reverse_map;;

let circle_reverse_map = NameMap.add ".centery" int_name
circle_reverse_map;;

let circle_reverse_map = NameMap.add ".radius" int_name
circle_reverse_map;;

let circle_reverse_map = NameMap.add ".color" string_name
circle_reverse_map;;

let circle_reverse_map = NameMap.add ".name" string_name
circle_reverse_map;;



(*line map and list, for verify member name *)
let line_member_list = [".startx";
".endx";".starty";".endy";".stroke";".color";".name"];;
let create_line =
	{
		line_startx = 0;
		line_endx = 100;
		line_starty = 0;
		line_endy = 100;
		line_stroke = 1;
		line_color = "#000000";
		line_name = "new_line";

	};;



let line_reverse_map = NameMap.empty;;

let line_reverse_map = NameMap.add ".startx" int_name line_reverse_map;;
```

```
let line_reverse_map = NameMap.add ".endx" int_name line_reverse_map;;

let line_reverse_map = NameMap.add ".starty" int_name line_reverse_map;;

let line_reverse_map = NameMap.add ".endy" int_name line_reverse_map;;

let line_reverse_map = NameMap.add ".stroke" int_name line_reverse_map;;

let line_reverse_map = NameMap.add ".color" string_name
line_reverse_map;;

let line_reverse_map = NameMap.add ".name" string_name line_reverse_map;;



(* label map and list, for verify member name *)
let label_member_list =
[".left";".top";".width";".height";".name";".content"];;
let create_label =
      {
            label_left = 0;
            label_top = 0;
            label_width = 100;
            label_height = 50;
            label_content = "I am a label.";
            label_name = "new_label";
      };;
let label_reverse_map = NameMap.empty;;
let label_reverse_map = NameMap.add ".left" int_name label_reverse_map;;

let label_reverse_map = NameMap.add ".top" int_name label_reverse_map;;

let label_reverse_map = NameMap.add ".width" int_name label_reverse_map;;
let label_reverse_map = NameMap.add ".height" int_name
label_reverse_map;;
let label_reverse_map = NameMap.add ".content" string_name
label_reverse_map;;
let label_reverse_map = NameMap.add ".name" string_name
label_reverse_map;;

(* all variables map, to use as default value  *)
let imageMap = NameMap.empty
and intMap = NameMap.empty and stringMap = NameMap.empty
and rectMap = NameMap.empty and circleMap = NameMap.empty
```

```
and labelMap = NameMap.empty and lineMap = NameMap.empty and typeMap =
NameMap.empty;;

(* local table and global table default values *)
let global_table = (intMap, stringMap, lineMap, labelMap, circleMap,
rectMap,

imageMap, typeMap) and local_table = (intMap, stringMap, lineMap, labelMap,
circleMap, rectMap,
imageMap, typeMap);;

(*********************************************helper
functions*********************************************)
let find_int v env = let (_intMap, _, _, _, _, _, _, _) = env in
      NameMap.find v _intMap;;
let find_string v env = let (_,_stringMap, _, _, _, _, _, _) = env in
       NameMap.find v _stringMap;;
let if_int_constant v = if(v.[0] >= '0' && v.[0] <= '9') then true else
false;;

let _int_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,
       _rectMap, _imageMap, _typeMap) = table and _value =
              int_of_string(value) in let _intMap = NameMap.add

       var _value _intMap in (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,

       _rectMap, _imageMap, _typeMap);;


let _string_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = table  in let _stringMap =
NameMap.add

       var value _stringMap in (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,

       _rectMap, _imageMap, _typeMap);;
```

```
let _image_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = table in let new_value =
NameMap.find

       value _imageMap in let _imageMap = NameMap.add var new_value
_imageMap

       in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap);;



let _label_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap , _typeMap) = table in let new_value =
NameMap.find

       value _labelMap in let _labelMap = NameMap.add var new_value
_labelMap

       in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap);;



let _line_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = table in let new_value =
NameMap.find

       value _lineMap in let _lineMap = NameMap.add var new_value _lineMap

       in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap);;
```

```
let _circle_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = table in let new_value =
NameMap.find

       value _circleMap in let _circleMap = NameMap.add var new_value
_circleMap

       in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap);;


let _rect_assign var value table = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = table in let new_value =
NameMap.find

       value _rectMap in let _rectMap = NameMap.add var new_value _rectMap

       in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap);;

(* object assign functions *)
let int_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, int_name, ((_int_assign var value
locals), globals)
    else if NameMap.mem var g_typeMap then var, int_name, (locals,
(_int_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let string_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, string_name, ((_string_assign
var value locals), globals)
    else if NameMap.mem var g_typeMap then var, string_name, (locals,
```

```
(_string_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let label_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, label_name,((_label_assign var
value locals), globals)
    else if NameMap.mem var g_typeMap then var, label_name, (locals,
(_label_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let image_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, image_name, ((_image_assign var
value locals), globals)
    else if NameMap.mem var g_typeMap then var, image_name, (locals,
(_image_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let line_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, line_name, ((_line_assign var
value locals), globals)
    else if NameMap.mem var g_typeMap then var, line_name,(locals,
(_line_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let rect_assign var value env =
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, rect_name, ((_rect_assign var
value locals), globals)
    else if NameMap.mem var g_typeMap then var, rect_name, (locals,
(_rect_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

let circle_assign var value env =
```

```
    let locals,globals = env in
    let (_, _, _, _, _, _, _, l_typeMap) = locals and
        (_, _, _, _, _, _, _, g_typeMap) = globals in
    if NameMap.mem var l_typeMap then var, circle_name,((_circle_assign
var value locals), globals)
    else if NameMap.mem var g_typeMap then var, circle_name, (locals,
(_circle_assign var value globals))
    else raise (Failure("can't find variable. " ^ var));;

(* object member assignment *)
let member_assign var member _type v_right t_right table = let m_type =

       (match _type with

       "image" -> let __type = try NameMap.find member image_reverse_map
with

                            Not_found -> raise (Failure("Variable is not
defined." ^ _type)) in __type

       | "rect" -> let __type = try NameMap.find member rect_reverse_map
with

                            Not_found -> raise (Failure("Variable is not
defined." ^ _type)) in __type

       | "line" -> let __type = try NameMap.find member line_reverse_map
with

                            Not_found -> raise (Failure("Variable is not
defined." ^ _type)) in __type

       | "circle" -> let __type = try NameMap.find member circle_reverse_map
with

                            Not_found -> raise (Failure("Variable is not
defined." ^ _type)) in __type

       | "label" -> let __type = try NameMap.find member label_reverse_map
with

                            Not_found -> raise (Failure("Variable is not
defined." ^ _type)) in __type
```

```
        | _ -> "shouldn't happen here")
        and (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,_rectMap,
_imageMap, _typeMap) = table

        in

        if m_type <> t_right then raise (Failure("type not matched. " ^ var))

        else match _type with
               "image" -> let _image = NameMap.find var _imageMap in

                        (match member with

                        ".left" ->  _image.image_left <-
int_of_string(v_right)

                        |".top" -> _image.image_top <-
int_of_string(v_right)

                        |".width" -> _image.image_width <-
int_of_string(v_right)

                        |".height" -> _image.image_height <-
int_of_string(v_right)

                        |".url" -> _image.image_url <- v_right

                        |".name" -> _image.image_name <- v_right

                        | _ -> raise (Failure("member doesn't exist.")));

                        let _imageMap = NameMap.add var _image _imageMap in

                        (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,_rectMap, _imageMap, _typeMap)

              |"rect" -> let _rect = NameMap.find var _rectMap in

            (match member with

           ".left" -> _rect.rect_left <- int_of_string(v_right)

           |".top" -> _rect.rect_top <- int_of_string(v_right)
```

```
            |".color" -> _rect.rect_color <- v_right

            |".width" -> _rect.rect_width <- int_of_string(v_right)

            |".height" -> _rect.rect_height <- int_of_string(v_right)

            |".name" -> _rect.rect_name <- v_right
               |_ -> raise (Failure ("member doesn't exist.")));
                     let _rectMap = NameMap.add var _rect _rectMap in
                      (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,_rectMap, _imageMap, _typeMap)

        |"circle" -> let _circle = NameMap.find var _circleMap in

            (match member with

            ".centerx" -> _circle.circle_centerx <- int_of_string(v_right)

            |".centery" -> _circle.circle_centery <-
int_of_string(v_right)

            |".radius" -> _circle.circle_radius <- int_of_string(v_right)

            |".color" -> _circle.circle_color <- v_right

            |".name" -> _circle.circle_name<- v_right
            |_ -> raise (Failure ("member doesn't exist.")));

               let _circleMap = NameMap.add var _circle _circleMap in

                     (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,_rectMap, _imageMap, _typeMap)

        |"line" -> let _line = NameMap.find var _lineMap in

            (match member with

            ".startx" -> _line.line_startx <- int_of_string(v_right)

            |".starty" -> _line.line_starty <- int_of_string(v_right)

            |".endx" -> _line.line_endx <- int_of_string(v_right)

            |".endy" -> _line.line_endy <- int_of_string(v_right)
```

```
            |".stroke" -> _line.line_stroke <- int_of_string(v_right)

            |".color" -> _line.line_color <- v_right

            |".name" -> _line.line_name <- v_right
            |_ -> raise (Failure ("member doesn't exist.")));

                     let _lineMap = NameMap.add var _line _lineMap in

                      (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,_rectMap, _imageMap, _typeMap)

        |"label" -> let _label = NameMap.find var _labelMap in

            (match member with

            ".left" -> _label.label_left <- int_of_string(v_right)

            |".top" -> _label.label_top <- int_of_string(v_right)

            |".width" -> _label.label_width <- int_of_string(v_right)

            |".height" -> _label.label_height <- int_of_string(v_right)

            |".content" -> _label.label_content <- v_right

            |".name" -> _label.label_name <- v_right
            |_ -> raise (Failure ("member doesn't exist.")));

               let _labelMap = NameMap.add var _label _labelMap in

                     (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,_rectMap, _imageMap, _typeMap)

              | _ -> table;;

(* function for update the local value table *)
let declare _type id value value_list env =

       let (_intMap, _stringMap, _lineMap, _labelMap,

                    _circleMap, _rectMap, _imageMap, _typeMap), globals
= env in
```

```
      match _type with INT ->

            if (NameMap.mem id _typeMap)

            then raise (Failure ("Duplicated Definition. " ^ id))

            else let _typeMap = NameMap.add id int_name _typeMap in
       let _intMap = if(value = "") then NameMap.add id 0 _intMap

            else let value = int_of_string(value) in NameMap.add id value
_intMap

            in (_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap), globals
      | STRING ->

            if (NameMap.mem id _typeMap)

            then raise (Failure ("Duplicated Definition. " ^ id))

            else let _typeMap = NameMap.add id string_name _typeMap in
let

            _stringMap = if value = "" then NameMap.add id "" _stringMap

            else NameMap.add id value _stringMap
            in (_intMap, _stringMap, _lineMap, _labelMap, _circleMap,
_rectMap, _imageMap, _typeMap), globals
      | RECT ->

            if (NameMap.mem id _typeMap)
            then raise (Failure ("Duplicated Definition." ^ id))

            else let _typeMap = NameMap.add id rect_name _typeMap in

            let _rectMap = NameMap.add id create_rect _rectMap

            in if value_list = ["",""] then (_intMap, _stringMap,
_lineMap, _labelMap,_circleMap, _rectMap, _imageMap, _typeMap), globals
            else (List.fold_left (fun _env local ->

            let t_type =
```

```
                    try NameMap.find (fst local) rect_reverse_map with

                    Not_found -> raise (Failure ("invalid assignment. "
^ id))

             in member_assign id (fst local) rect_name (snd local) t_type
_env)

             ((_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap)) value_list), globals
      | CIRCLE ->

             if (NameMap.mem id _typeMap)

             then raise (Failure ("Duplicated Definition. " ^ id))

             else let _typeMap = NameMap.add id circle_name _typeMap in

             let _circleMap = NameMap.add id create_circle _circleMap

             in if value_list = ["",""] then (_intMap, _stringMap,
_lineMap, _labelMap,_circleMap, _rectMap, _imageMap, _typeMap), globals
             else (List.fold_left (fun _env local ->

             let t_type =
                    try NameMap.find (fst local) circle_reverse_map with

                    Not_found -> raise (Failure ("invalid assignment " ^
id))

             in member_assign id (fst local) circle_name (snd local)
t_type _env)

             (_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap) value_list), globals
      | LINE ->

             if (NameMap.mem id _typeMap)

             then raise (Failure ("Duplicated Definition. " ^ id))

             else let _typeMap = NameMap.add id line_name _typeMap in

             let _lineMap = NameMap.add id create_line _lineMap
```

```
            in if value_list = ["",""] then (_intMap, _stringMap,
_lineMap, _labelMap,_circleMap, _rectMap, _imageMap, _typeMap), globals
            else (List.fold_left (fun _env local ->

            let t_type =

                  try NameMap.find (fst local) line_reverse_map with

                  Not_found -> raise (Failure ("invalid assignment. "
^ id))

            in member_assign id (fst local) line_name (snd local) t_type
_env)

            (_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap) value_list), globals
      | LABEL ->

            if (NameMap.mem id _typeMap)

            then raise (Failure ("Duplicated Definition. " ^ id))

            else let _typeMap = NameMap.add id label_name _typeMap in

            let _labelMap = NameMap.add id create_label _labelMap

            in if value_list = ["",""] then (_intMap, _stringMap,
_lineMap, _labelMap, _circleMap, _rectMap, _imageMap, _typeMap), globals
            else (List.fold_left (fun _env local ->

            let t_type =

                  try NameMap.find (fst local) label_reverse_map with

                  Not_found -> raise (Failure ("invalid assignment " ^
id ))

            in member_assign id (fst local) label_name (snd local) t_type
_env)

            (_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap) value_list), globals
      | IMAGE ->
```

```
            if (NameMap.mem id _typeMap)

            then raise (Failure ("Duplicated Definition. " ^ id))

            else let _typeMap = NameMap.add id image_name _typeMap in

            let _imageMap = NameMap.add id create_image _imageMap

            in if value_list = ["",""] then (_intMap, _stringMap,
_lineMap, _labelMap,_circleMap, _rectMap, _imageMap, _typeMap), globals
            else (List.fold_left (fun _env local ->

            let t_type =

                  try NameMap.find (fst local) image_reverse_map with

                  Not_found -> raise (Failure ("invalid assignment. "
^ id))

            in member_assign id (fst local) image_name (snd local) t_type
_env)

            (_intMap, _stringMap, _lineMap, _labelMap,_circleMap,
_rectMap, _imageMap, _typeMap) value_list), globals

(* object member get functions *)

let get_value var member _type env = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

      _rectMap, _imageMap, _typeMap) = env
      in match _type with
            "image" -> let _image = NameMap.find var _imageMap in
                  (match member with

                  ".left" ->  string_of_int(_image.image_left),
int_name, env

                  |".top" -> string_of_int(_image.image_top), int_name,
env

                  |".width" -> string_of_int(_image.image_width),
int_name,
```

```
                    env
                    |".height" -> string_of_int(_image.image_height),
int_name,
                    env
                    |".url" -> _image.image_url, string_name, env
                    |".name" -> _image.image_name, string_name, env
                    | _ -> raise (Failure("member " ^ member ^ "  doesn't
exist for " ^ var)))
              |"rect" -> let _rect = NameMap.find var _rectMap in
           (match member with
           ".left" -> string_of_int(_rect.rect_left), int_name, env
           |".top" -> string_of_int(_rect.rect_top), int_name, env
           |".color" -> _rect.rect_color, string_name,env
           |".width" -> string_of_int(_rect.rect_width), int_name, env
           |".height" -> string_of_int(_rect.rect_height), int_name, env
           |".name" -> _rect.rect_name, string_name,env
           | _ -> raise (Failure("member " ^ member ^ "  doesn't exist for
" ^ var)))
       |"line" -> let _line = NameMap.find var _lineMap in
           (match member with
           ".startx" -> string_of_int(_line.line_startx),int_name,env
           |".starty" -> string_of_int(_line.line_starty), int_name, env
           |".endx" -> string_of_int(_line.line_endx), int_name, env
           |".endy" -> string_of_int(_line.line_endy), int_name, env
```

```
            |".stroke" -> string_of_int(_line.line_stroke), int_name, env

            |".color" -> _line.line_color, string_name, env
            |".name" -> _line.line_name, string_name,env
            | _ -> raise (Failure("member " ^ member ^ "  doesn't exist for
" ^ var)))

        |"circle" -> let _circle = NameMap.find var _circleMap in

            (match member with

           ".centerx" -> string_of_int (_circle.circle_centerx), int_name,
env

            |".centery" -> string_of_int (_circle.circle_centery),
int_name, env

            |".radius" -> string_of_int (_circle.circle_radius), int_name,
env

            |".color" -> _circle.circle_color, string_name, env
            |".name" -> _circle.circle_name, string_name, env
            | _ -> raise (Failure("member " ^ member ^ "  doesn't exist for
" ^ var)))

        |"label" -> let _label = NameMap.find var _labelMap in

            (match member with

            ".left" -> string_of_int (_label.label_left), int_name, env

            |".top" -> string_of_int (_label.label_top), int_name, env

            |".width" -> string_of_int (_label.label_width), int_name, env

            |".height" -> string_of_int (_label.label_height), int_name,
env

            |".content" -> _label.label_content, string_name, env
            |".name" -> _label.label_name,string_name,env
            | _ -> raise (Failure("member " ^ member ^ "  doesn't exist for
" ^ var)))
```

```
            | _ -> raise (Failure("invalid operation. variable can't be
int or string."));;

let convert_to_value var _type env = let (_intMap, _stringMap, _lineMap,
_labelMap, _circleMap,

       _rectMap, _imageMap, _typeMap) = env
       in match _type with
             "image" -> let _image = NameMap.find var _imageMap in
                   List.fold_left (fun value_list member ->
              let _type = NameMap.find member image_reverse_map and
(_value, _, _) =
                  get_value var member _type env in
              (_type, _value)::value_list) [] image_member_list
             |"rect" -> let _rect = NameMap.find var _rectMap in

          List.fold_left (fun value_list member ->
             let _type = NameMap.find member rect_reverse_map and (_value,
_, _) =
                  get_value var member _type env in
              (_type, _value)::value_list) [] rect_member_list
       |"line" -> let _line = NameMap.find var _lineMap in

              List.fold_left (fun value_list member ->
             let _type = NameMap.find member line_reverse_map and (_value,
_, _) =
                  get_value var member _type env in
              (_type, _value)::value_list) [] line_member_list
       |"circle" -> let _circle = NameMap.find var _circleMap in

          List.fold_left (fun value_list member ->
              let _type = NameMap.find member circle_reverse_map and
(_value, _, _) =
                  get_value var member _type env in
              (_type, _value)::value_list) [] circle_member_list
       |"label" -> let _label = NameMap.find var _labelMap in

           List.fold_left (fun value_list member ->
             let _type = NameMap.find member label_reverse_map and
(_value, _, _) =
                  get_value var member _type env in
              (_type, _value)::value_list) [] label_member_list
          | _ -> raise (Failure("invalid operation. variable can't be int
or string."));;
```

```
let print_result env = let (_intMap, _stringMap, _lineMap, _labelMap,
_circleMap,

       _rectMap, _imageMap, _typeMap) = env in
NameMap.mapi (fun x y-> print_endline x; print_endline y) _typeMap;;

let run (vars, funcs) =
    let func_decls = List.fold_left
        (fun funcs fdecl -> NameMap.add fdecl.fname fdecl funcs)
NameMap.empty funcs
    in
    let rec call fdecl actuals env =
    let rec expr env = function Literal(v) -> string_of_int v, int_name,
env

    | Stringvalue(v) -> v, string_name, env
       | Id(v) ->

      let (_,_,_,_,_,_,_,l_typeMap),
        (_,_,_,_,_,_,_,g_typeMap) = env in
         let _type,table = if NameMap.mem v l_typeMap then (NameMap.find
v l_typeMap), (fst env) else
       if NameMap.mem v g_typeMap then (NameMap.find v g_typeMap), (snd
env)
       else raise (Failure ("wrong"))
         in (if _type = int_name then string_of_int (find_int v table)

         else if _type = string_name then find_string v table else v), _type,

         env

       | Binop(v1, op, v2) ->

         let e1 = expr env v1 and e2 = expr env v2 in

         let _v1, t1, _ = e1 and _v2, t2, _ = e2 in

         if( t1 <> int_name || t2 <> int_name) then raise (Failure ("wrong"))

         else

         let boolean i = if i then 1 else 0 in
```

```
      let _v1 = int_of_string( _v1) and _v2 = int_of_string(_v2) in

      string_of_int (match op with

        Add -> _v1 + _v2

      | Sub -> _v1 - _v2

      | Mult -> _v1 * _v2

      | Div -> _v1 / _v2

      | Equal -> boolean (_v1 = _v2)

      | Neq -> boolean (_v1 != _v2)

      | Less -> boolean (_v1 < _v2)

      | Leq -> boolean (_v1 <= _v2)

      | Greater -> boolean (_v1 > _v2)

      | Geq -> boolean (_v1 >= _v2)), int_name, env
        | Assign(var, e) -> let (_,_,_,_,_,_,_,l_typeMap),
        (_,_,_,_,_,_,_,g_typeMap) = env in
         let _type = if NameMap.mem var l_typeMap then NameMap.find var
l_typeMap else
       if NameMap.mem var g_typeMap then NameMap.find var g_typeMap
       else raise (Failure ("variable not found." ^ var))
       and  (v_right, t_right, _) = expr env e in
                    if(_type <> t_right)
                    then raise (Failure ("type not matched"))
                    else (match _type with
                    "int" -> int_assign var v_right env
                    | "string" -> string_assign var v_right env
                    | "image" -> image_assign var v_right env
                    | "line" -> line_assign var v_right env
                    | "rect" -> rect_assign var v_right env
                    | "circle" -> circle_assign var v_right env
                    | "label" -> label_assign var v_right env
                    | _ -> "", "", env)
       | Member(var, member) ->
       let (_,_,_,_,_,_,_,l_typeMap), (_,_,_,_,_,_,_,g_typeMap) = env in
       if NameMap.mem var l_typeMap then let _type = NameMap.find var
```

```
l_typeMap in
          let name, __type, table = get_value var member _type (fst env)
in name, __type, (table,snd env)
       else if NameMap.mem var g_typeMap then let _type = NameMap.find var
g_typeMap in
          let (name, __type, table) = get_value var member _type (snd env)
in name, __type, (fst env, table)
       else raise (Failure ("Variable not found." ^ var))
      | Member_Assign(var, member, e) -> let (_, _, _, _, _,_, _, l_typeMap),
(_, _, _, _, _, _, _, g_typeMap) = env
          in if NameMap.mem var l_typeMap then let _type = NameMap.find
var l_typeMap and
              (v_right, t_right, _) = expr env e in
                          var, _type, ((member_assign var member _type
v_right t_right (fst env)), snd env)
          else if NameMap.mem var g_typeMap then let _type = NameMap.find
var g_typeMap and
              (v_right, t_right, _) = expr env e in
                          var, _type, (fst env, member_assign var member
_type v_right t_right (snd env))
          else raise (Failure ("Variable not found." ^ var))
    | Noexpr -> "", "", env
    | Print(e) ->
       let name,_type, env = expr env e in
       print_endline (name ^ "," ^_type); name,_type,env
   | Draw(id) -> let (_,_,_,_,_,_,_,l_typeMap), (_,_,_,_,_,_,_,g_typeMap)
= env in
       if NameMap.mem id l_typeMap then let _type = NameMap.find id
l_typeMap in
          (*draw_shape id (fst env); *) "", "", env
       else if NameMap.mem id g_typeMap then let _type = NameMap.find id
g_typeMap in
          (*draw_shape id (snd env);*) "", "", env
       else raise (Failure ("Variable not found." ^ id))
    | Call (f, actuals) ->
       let fdecl = try NameMap.find f func_decls
       with Not_found -> raise (Failure ("undefined function " ^ f))
    in
    let actuals, env = List.fold_left
       (fun (actuals,env) actual ->
          let v, _type, env = expr env actual in
             if _type = "int" || _type = "string"
             then ((v,[]) :: actuals), env
             else ("", (convert_to_value v _type (fst env))::actuals,
```

```
env
              ) ([], env) actuals
    in
    let (locals, globals) = env in
    try
    let _globals = call fdecl actuals (local_table, globals) in "","",
(locals, globals)
       with ReturnException(v, globals) ->
     (fst v), fdecl.return_type, (locals, globals)
    in
    let rec exec env = function
       Expr(e) -> let _ ,_ , env = expr env e in env
           | If(e, s1, s2) ->
       let v, _type, env = expr env e in

           if _type = int_name then

       exec env (if int_of_string(v) != 0 then s1 else s2)

           else raise (Failure ("int required"))

           | While(e, s) ->

           let rec loop env =

           let v, _type, env = expr env e in

               if _type = int_name then

       (if int_of_string(v) != 0 then loop (exec env s) else env)

          else raise (Failure ("int required"))

     in loop env

    | For(e1, e2, e3, s) ->

     let _, _type, env = expr env e1 in

     let rec loop env =

       let v, _type, env = expr env e2 in

          if _type = int_name then
```

```
        (if int_of_string(v) != 0 then

         let _, _, env = expr (exec env s) e3 in

         loop env

          else env)

       else env

     in loop env

    | Block(stmts) -> List.fold_left exec env stmts

    | Declare(t, id) -> declare t id "" [("","")] env

    | Define_int (id, value) -> let value = string_of_int(value) in declare
INT id value [("","")] env

    | Define_string (id, value) -> declare STRING id value [("","")] env
    | Define_rect (id, value) -> declare RECT id "" value  env
    | Define_circle (id, value) -> declare CIRCLE id "" value env
    | Define_image (id, value) -> declare IMAGE id "" value env
    | Define_line (id, value) -> declare LINE id "" value env
    | Define_label (id, value) -> declare LABEL id "" value env
    | Return(e) ->
       let v, _type, (locals, globals) = expr env e in
       if _type = "string" || _type = "int"
           then raise (ReturnException((v,[("","")]), globals))
       else
          raise (ReturnException(("", convert_to_value v _type (fst env)),
globals))

   in
   let env = try List.fold_left2
       (fun env formal actual -> match (fst formal)
          with "int" -> declare INT (snd formal) (fst actual) [("", "")]
env
           | "string" -> declare STRING (snd formal) (fst actual) [("",
"")]env
           | "circle" -> declare RECT (snd formal)  "" (snd actual) env
           | "image" -> declare IMAGE (snd formal)  "" (snd actual) env
           | "line" -> declare LINE (snd formal)  "" (snd actual) env
```

```
            | "label" -> declare LABEL (snd formal) "" (snd actual) env
            | "rect" -> declare RECT (snd formal) "" (snd actual) env
            | _ -> raise (Failure ("invalid varible type.")))
               env fdecl.formals actuals
    with Invalid_argument(_) ->
        raise (Failure ("wrong number of arguments to " ^ fdecl.fname))

    in snd (List.fold_left (fun env statement ->  exec env statement)
        env fdecl.body)
    in
    let (globals, _) = List.fold_left (fun env value -> match (fst value)
with
        "int" -> declare INT (snd value) "" [("","")] env
        | "string" -> declare STRING (snd value) "" [("","")] env
        |  "image" -> declare IMAGE (snd value) "" [("","")] env
        |  "circle" -> declare CIRCLE (snd value) "" [("","")] env
        |  "rect" -> declare RECT (snd value) "" [("","")] env
        |  "line" -> declare LINE (snd value) "" [("","")] env
        |  "label" -> declare LABEL (snd value) "" [("","")] env
        | _ -> raise (Failure ("invalid variable type.")))
         (global_table,global_table) vars
    in try call (NameMap.find "main" func_decls) [] (local_table,globals)
    with Not_found -> raise (Failure ("did not find the main() function"));;
```