SFPL

Simple Floor Plan Language
09/25/2005

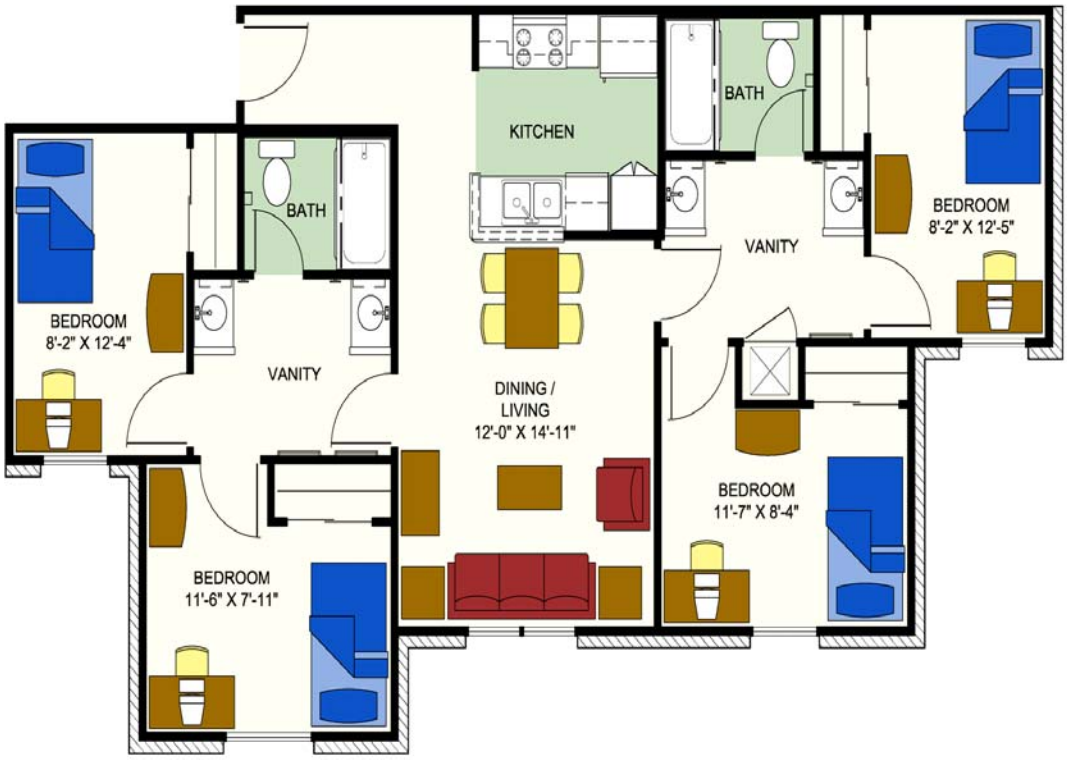
By: Huang-Hsu Chen (hc2237)
Xiao Song Lu(xl2144)
Natasha Nezhdanova(nin2001)
Ling Zhu(lz2153)

Chapter 1

Introduction

1.1 Introduction

Today, many people would like their homes custom-designed. They may want to make a floor plan for their newly bought properties or they may want to make changes on their living places such as to separate a big bedroom into two smaller ones. However, designing a house floor plan from scratch requires professional knowledge and can be costly and time-consuming. The language SFPL, which stands for Simple Floor Plan Language, provides an easy and efficient way for people to design their dream home. The language allows users to write a program that lays out basic dimensions and adds details such as doors, windows and other elements. The program will generate and output two-dimensional floor plan diagram(s). By changing variables in the program, users can obtain different diagrams to compare with different designs. In addition, there are flow-control mechanisms in the language to facilitate designing relatively large projects. In brief, the language is designed clear and easy to use, which will make designing a floor plan an easy job for ordinary people. The graph below shows a typical professional floor plan.



1.2 Overview of the Language

SFPL is basically a high-level object-oriented language. Object-oriented programming languages enable programmers to create modules that do not need to be changed when a new type of object is introduced. Although not as comprehensive as big object-oriented languages such as Java, SFPL provide built-in support for objects like walls, windows, staircases and doors. These built-in objects will make the language more intuitive and easy to use.

SFPL is designed to be a translated language, with its target language being Java. The user is able to design a floor plan in the form of a program in a text file. The translator will then output a Java source file that can be edited and compiled into Java byte-code. The SFPL translator is equipped with error detection so that SFPL syntactical errors are not passed on to the Java source code. This also makes error-checking and debugging become a relatively smooth process. Finally, the graphic output aspect of the language generally

relies on the *Java.awt* package since it provides sufficient utilities to generate a static 2D diagram.

1.3 Goals

The language is designed with the following goals in mind:

- **User Friendliness** - One of the primary goals of SFPL is user friendliness that is the syntax should be easy to learn, read, write, and manipulate. The syntax is meant to be simple and intuitive so that allows users who have little programming experience to learn it quickly. Code definitions in SFPL are structured to be modular that increases the readability of the language.
- **High Level** - Users do not have to worry about internal representation of data, and algorithm implementation. The modularity capabilities allow for extension to complex problem-solving routines.
- **Flexible** –To give users more flexibility and to support designing large scale building projects, SFPL allows users to define new objects through inheritance. In addition users are allowed to define Functions and Packages to reuse sectors of codes. Moreover, the language provides flow control mechanism such as conditional statements and iterative loops.
- **Portable** - Since SFPL is a translated language with its target language as Java and its translator implemented in Java, all SFPL code can be translated, executed, and evaluated on any machine that has a Java Runtime Environment with compiler. Java is a highly portable language, which therefore makes the translator and the SFPL code that it translates highly portable as well.

1.4 Some Language Features

- **Primitive Data Types:** There are two basic numerical data types, namely integer and floating-point. They are to be used for computing areas, locations, and sizes. In addition, the Boolean types are needed for control statements.

- **Basic objects:** These are objects like lines, rectangles, triangles and circles. They are the building blocks for the floor plan diagrams. Users can define other objects through inheritance.

- **Other built-in objects:**

1. ***Wall, Window, Door, Stair…:*** The names of the objects speak for themselves. There will be methods associated with each object to determine their sizes, locations, shapes. They are subclasses of the basic objects.
2. ***Name:*** This object can be used to mark the function of each room on the final floor diagram, e.g., *kitchen*, *bedroom*, or *bathroom*.
3. ***Dimension:*** Dimensions are essential elements of any floor-plan. This object can be used to specifically mark the dimensions of windows, doors, and walls on the floor-plan diagram.
4. ***Design:*** This object is to be used to specify the properties of the final diagram, such as the background color, and the specific colors and/or symbols to designate the walls, doors, windows, and stairs on the floor plan.

- **Operators:** Operators are necessary for comparisons, loops, and size calculations. So SFPL will have all the essential mathematical operators, i.e., at least, the +, -, <, >, and = operators.

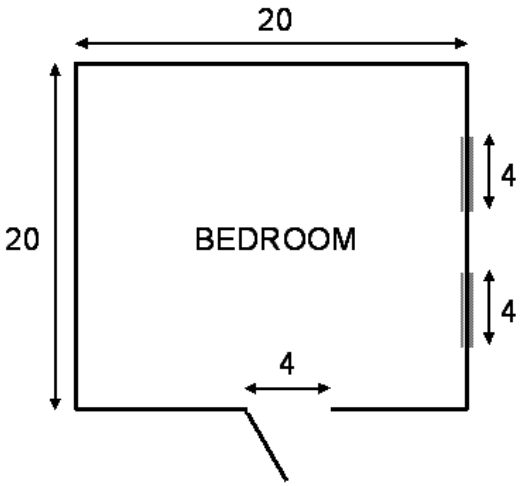
- **Control Statements**: To control the program flow, we are implementing the if/else, and while clauses for control statements. For instance, a while-loop can be used to distribute windows evenly along a particular wall of a room, as well as to check when the end (corner) of the room is reached. The if-statement may be used to choose the number of windows or doors (if any fit at all) to put at a particular location, for example, along a particular wall.

- **Array/Vector Built-In Data Structures** Since at the beginning of a design process, the quantity of some (or all) elements may be uncertain, it would be preferable to use dynamic allocation for storing those elements.

1.5 Sample Code

Sample code to construct the room shown below in SFPL:

```
wall WALLS[4]  // instantiate 4 walls
wall[0].put(10,10,30,10) // position wall[0]
wall[1].put(30,10,30,30)
wall[2].put(30,30,10,30)
wall[3].put(10,30,10,10)

window WIN[2]  // instantiate a window
WIN[0].put(30,14,30,18)
WIN[1].put(30,22,30,26)

door DOOR  // instantiate a door
DOOR.put(20,30,24,30)
```



The code to specify the name and dimensions of the room is not shown

1.6 Optional Features

It is possible to extend SFPL to provide methods for changing the materials out of which different elements of a design are made, and to calculate prices for different options. Such functionality would be very useful because it would allow the user to compare various possibilities for the choice of materials and choose the one that is most desirable financially. It would also be useful to introduce a method that could show the list of materials to be used for the maximum cost reduction.

The potential users of SFPL would be very grateful to its creators if the SFPL compiler could check for, and help them correct unpleasant design mistakes, e.g., windows overlapping doors. It would be nice if the compiler generated an error message and pointed out to the user what the problem is and how to fix it.

The above functionalities are desirable but may be difficult to implement. Thus whether to implement these optional functionalities will depend on the time constraints of the whole project.

Chapter 2

Tutorial

2.1 Program Flow

An SFPL program should start with function definitions and variable declarations. Variables can also be declared inside function definitions. However, functions cannot be defined within functions.

SFPL uses static scoping, so local variables are visible only within the function where they are defined.

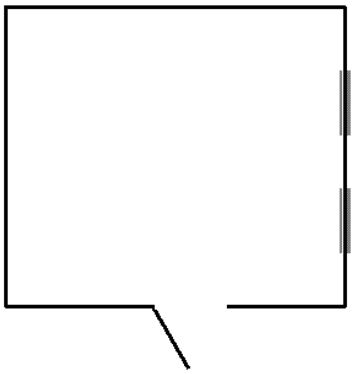
Built-in functions available to the user: line(), rectangle(), ellipse(), label(), and print(). SFPL's standard library provides the following functions: getX(), getY(), print(), wall(), window(), door(), stair(), and dimension(). Details on these functions can be found in the Reference Manual.

Standard library functions are compiled and linked with every SFPL program by default. So there is no need to include or import any external library files. Standard library functions can be overloaded, so the user can redefine any of the functions above except for the built-in functions.

2.2 Sample Program

The code below draws the room on the right:

```
point:p1,p2,p3, p4,p5,p6,p7,p8,p9,p10;
p1 = 100@100;
p2 = (100+100)@100;
p3 = 100@(100+100);
p4 = p1 + p1;

wall( p1,p2,blue );
wall( p2,p4,blue );
wall( p1,p3,blue );
wall( p3,p4,blue );
```



```
window((getX(p2)+3)@120,(getX(p2)-3)@(120+15),green);
window((getX(p2)+3)@165,(getX(p2)-3)@180, green);
door((getX(p3) + 40)@200,(getX(p3) + 60)@200,5, red, 1, 0,true );
```

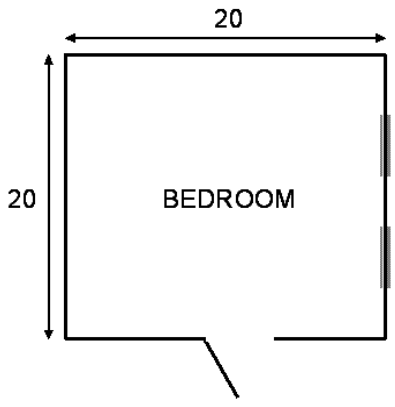
Details for choosing the last two parameters for the door function are provided in the Reference Manual.

Add dimensions:

```
dimension( 100@220,200@220, "20", true, grey );
dimension(220@100, 220@200,"20",false, grey);
```

Add label:

```
label(130@150, "BEDROOM", grey);
```



Hints: It might be helpful to choose a few key points for your design, and then set the door, window, wall functions, etc., by adding vectors to those points.

That is, each point can be treated as a vector starting at the origin and terminating at the coordinates specified by the point. So, if we choose p1 as the upper left corner of the room, we can position the label by specifying its starting point as p1+ (1/3(width of the room))@(1/2(length of the room)).

Chapter 3

Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

There are six classes of tokes: identifiers, keywords, constants, string literals, operators, and other separators. Blanks, horizontal and vertical tabs, newlines, and comments as described below (collectively, “white space”) are ignored except as they separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

If the input stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

3.1.2 Comments

The characters /* introduce a comment, which terminates with the characters */. The characters // introduce a single-line comment.

3.1.3 Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore _ counts as a letter. Upper and lower case letters are treated differently. There is no limit on the length of identifiers.

3.1.4 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

blue	boolean	line	orange
brown	color	point	
green	ellipse	print	
grey	else	rectangle	
black	false	return	
purple	function	string	
red	if	true	
white	int	void	
yellow	label	while	

3.1.5 Integer Constants

An integer constant is a sequence of digits. It is always taken to be decimal.

3.1.6 String Literals

A string literal is a sequence of characters surrounded by double quotes, as in “this is a string”. String literals do not contain newline or double-quote characters.

3.1.7 Point Literals

A point literal is a sequence of digits separated by the @ symbol. It represents the coordinates of a point. I.e., 2@3 represents the coordinates of p(2,3), where the first integer is the x-coordinate, and the second integer is the y-coordinate.

3.1.8 Color Literals

A color literal is the # symbol followed by 6 characters to specify the color on the RGB scale or by one of the following words: red, orange, yellow, green, blue, purple, brown, black, grey, or white. For example, #c6e2ff is the slate-gray color, and #blue is the blue color.

3.2 Operators

3.2.1 Multiplicative Operators

The multiplicative operators * and / group left-to-right. The operands of * and / must be of type int.

The binary * operator denotes multiplication.

The binary / operator yields the quotient of the division of the first operand by the second; if the second operand is 0, the result is undefined.

3.2.2 Additive Operators

The additive operators + and – group left-to-right. The operands of + and – must be of type int.

The result of the + operator is the sum of the operands.
The results of the – operator is the difference of the operands.

3.2.3 Relational Operators

The relational operators group left-to-right. The operand of the <, >, <=, and >= operators must be of type int.

The operators < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true. The type of the result is int.

3.2.4 Equality Operators

The == (equal to) and the != (not equal to) operators are analogous to the relational operators except for their lower precedence. (Thus a<b == c<d is 1 whenever a<b and c<d have the same truth-value.)

3.2.5 Logical Operators

The && operator groups left-to-right. It returns true if both its operands are not equal to zero or false, false otherwise.

The || operator groups left-to-right. It returns true if either of its operands is not equal to zero or false, and false otherwise.The operands of the && and || operators must be of type int or boolean.

3.2.6 Assignment Operator

Assignment operator = groups right-to-left.

The left operand must be of type int, boolean, string, color, or point.

The right operand must be of type int, boolean, or an integer constant if the left operand is of type int or boolean. It must be of type string or a string literal, if the left operand is of type string. It must be of type color or a color literal, if the left operand is of type color. Finally, it must be of type point or a point literal, if the left operand is of type point.

3.2.7 Comma Operator

A pair of expression separated by a comma is evaluated left-to-right.

3.2.8 Coordinate operator

Coordinate operator @ groups left-to-right.

Left and right operands must be arithmetic expressions or they must be of type int. Combinations are possible, e.g., int@(a*b + c), where a,b, and c are all of type int.

3.2.9 Operator Precedence and Associativity

Below, the operators are listed from the highest to the lowest precedence:

Operators	Associativity
( )	left to right
* /	left to right

@	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
=	right to left
,	left to right

3.3 Declarations

Declarations specify the interpretation given to each identifier; they do not reserve storage associated with the identifier. Declarations that reserve storage are called *definitions*.

3.3.1 Variable declarations:

variable-declaration:
type-specifier : *declarator-list* ;

type-specifier:
void
int
boolean
color
point

string

declarator-list:
declarator
declarator-list, *declarator*
declarator = *initializer*
declarator-list, *declarator* = *initializer*

initializer:
integer constant | *string literal* | *point literal* | *color literal*

3.3.2 Function definitions:

Functions must be defined when they are declared. A function definition must be an external declaration, i.e., it cannot be inside any other function. Function definitions must be made at the beginning of a translation unit.

function-definition:
function *type-specifier identifier*(*parameter-list)*
{
variable-declaration-list
statement-list
}

parameter-list:
type-specifier : *identifier*
parameter-list, *type-specifier* : *identifier*

variable-declaration-list:
variable-declaration

variable-declaration-list variable-declaration

statement-list:
statement
statement-list statement

3.4 Statements

Except as described, statements are executed in sequence. They fall into the following groups:

statement:
assignment-statement
compound-statement
if-statement
while-statement
function-call-statement
return statement
break-statement
continue-statement

3.4.1 Assignment statements

assignment-statement:
identifier = *initializer*;

3.4.2 Compound Statements (Blocks)

compound-statement:
{ *statement-list* }

statement-list:
statement
statement-list statement

3.4.3 If statements

if-statement:
if (*expression*) *statement* else *statement*

3.4.4 While Statements

while-statement:
while (*expression*) *statement*

3.4.5 Function Call Statement

function-call-statement:
identifier(expression-list) ;

expression-list:
expression
expression-list, expression

3.4.6 return Statements

return-statement:
return ;
return *expression* ;

3.4.7 Break Statement

break ;

3.4.8 Continue Statement

continue ;

3.5 Expressions

Expressions include arithmetic, relational, and logical expressions.

expression:
logical-expression
logical-expression || *logical-expression*

logical-expression:
relational-factor
relational-factor && *relational-factor*

relational-factor:

arithmetic-expression

arithmetic-expression relational-operator arithmetic-expression

relational-operator: one of

>= <= > < == !=

arithmetic-expression:

point-expression

point-expression + *point-expression*

point-expression - *point-expression*

point-expression:

arithmetic-expression

arithmetic-expression @ arithmetic-expression

arithmetic-term:

atom

atom * *atom*

atom | *atom*

atom:

identifier

function-call-statement

int

string

boolean

(*expression*)

3.6 Scope

The SFPL language uses dynamic scoping. When there a variable is used that is not declared locally, it is bounded to the nearest variable declaration.

3.7 Built-in Functions

SFPL's built-in functions provide basic functionality for drawing simple shapes. They cannot be overridden by the user.

line(point: p1, point: p2, color: c)

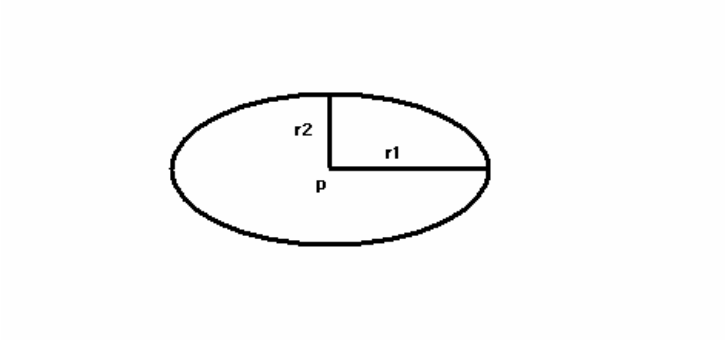
Draws a line of color c from p1 to p2.

rectangle(point: p, int: w, int: l, color: c1, color: c2)

Draws a rectangle of width w and length l with upper-left corner at p. c1 is the contour color, c2 is the filling color.

ellipse(point: p, int: r1, int: r2, color: c1, color: c2)

Draws an ellipse centered at p, with radii r1 and r2. c1 is the contour color, c2 is the filling color.



label(point: p, string: l, color: c)

Puts label “l” of color c on the diagram, e.g. BATHROOM, KITCHEN, etc. p is the upper-left corner of the label.

print (x, y, ...)

Takes an arbitrary number of variables regardless of their type and outputs their values. This function is used mainly for testing.

getX (point: p)

Returns the X coordinate of point p.

getY (point: p)

Returns the Y coordinate of point p.

3.8 Standard Library

SFPL’s standard library is a part of the language. All standard library functions are compiled with every SFPL program by default. There is no need to include or import those functions in your program.

wall(point: p1, point: p2, color: c)

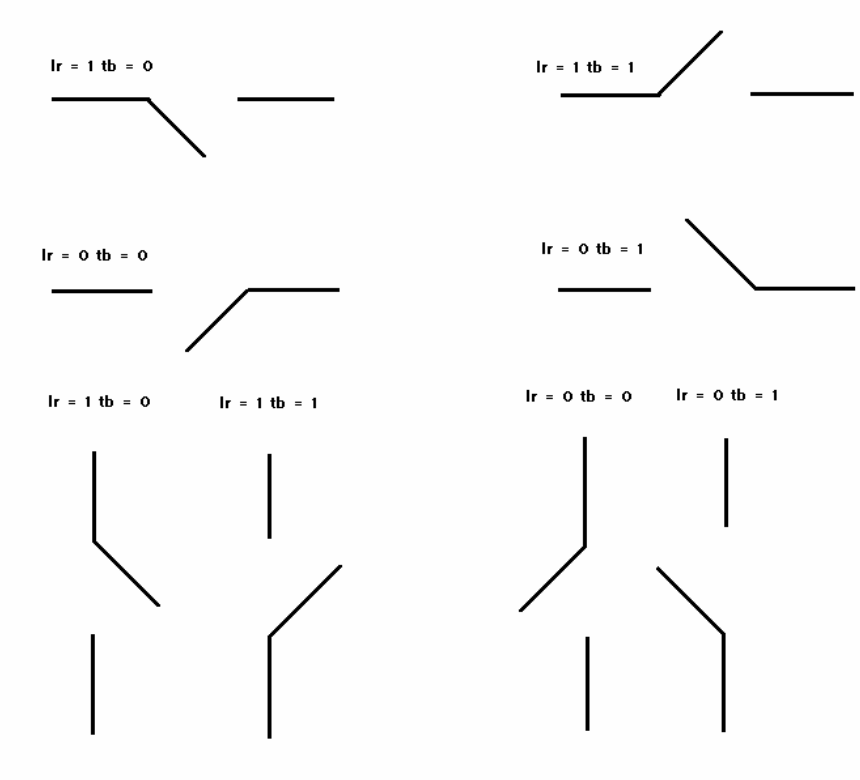
Draws a wall of color c from p1 to p2.

window(point: p1, point: p2, color: c)

Draws a window of color c from p1 to p2.

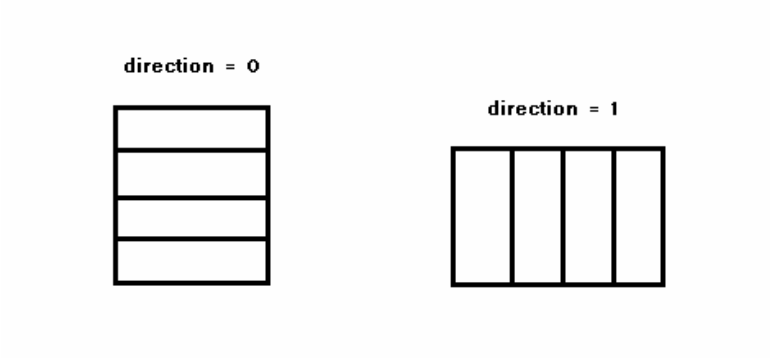
door(point: p1, point: p2, int: length, color c, int lr, int bt, boolean:h)

Draws a door of color c from p1 to p2. The last two parameters of type int specify the direction the door opens and the position of the door's handle. lr=0 for left, lr=1 for right; bt=0 for bottom, bt=1 for top, h defines the directions (horizontal or vertical)of the door. The possible configurations are shown below:



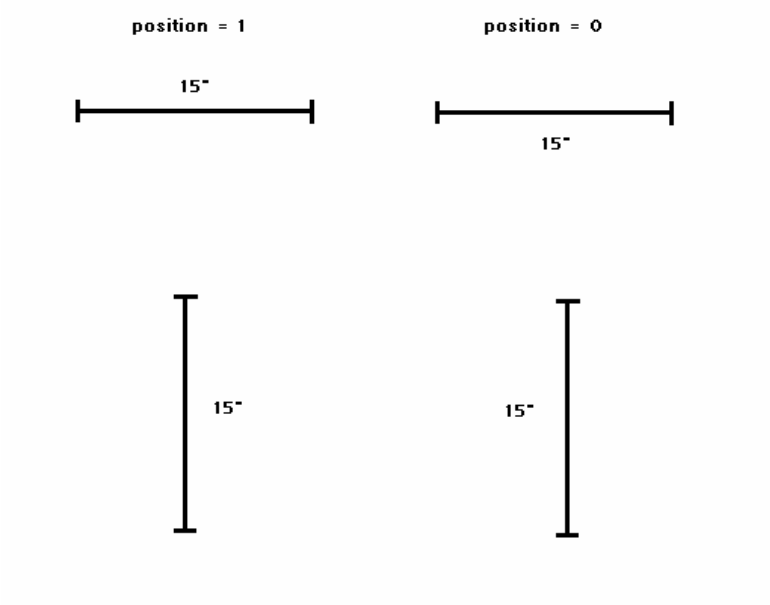
stair(point p, int w, int l, color c1, color c2, Boolean direction)

Draws a staircase of width w and length l with upper-left corner at p. c1 is the contour color, c2 is the filling color; direction= true for steps to be drawn horizontally, direction=false for steps to be drawn vertically, as shown below.



dimension(point p1, point p2, string s, int position, color c)

Draws a labeled arrow of color c from p1 to p2 to show a dimension of an object. s is the label. If position=0, the label will be drawn to the left of the arrow if the arrow is vertical, or below the arrow if it is horizontal; if position=1, the label will be drawn to the right of a vertical arrow or above a horizontal arrow.



3.9 Sample SFPL program

```
    //function definitions

    function int f1(int:a, point:b) {…}
    function void f2(color:c, point:d) {…}

    //beginning of program

    int: A = 5;
point: B = 2@3, D = 5@1;
color: C = #554466;

f1(A, B);
f2(C, D);
    //end
```

Chapter 4

Project Plan

To avoid major fallbacks, our group decided to plan ahead and set up incremental milestones. Throughout the semester we had weekly status meetings on Saturdays to update on our individual responsibilities. We also planned weekly meetings with our supervisor who helped us solve problems as they came up.

4.1 Timeline

Planning	
9.20	Development of language goals and desired functionality
9.24	Deciding upon language structure and basic requirements
9.27	White Paper complete
Specifications	
9.30	Syntax development
10.08	Creating the grammar
10.15	Removing inconsistencies between the grammar and the LRM
10.20	LRM complete
Implementation and Testing	
10.23	Developing interfaces
10.29	Building AST tree
11.15	Creating data types

11.20	Coding the interpreter
11.22	Completing the symbol table
12.01	Coding the built-in functions
12.15	Expanding and adjusting documentation
12.15	Testing
12.18	Finishing documentation

4.2 Team Responsibilities

Front End	Ling Zhu, Xiao Song Lu
Back End	Huang-Hsu Chen, Ling Zhu
Documentation	Natalia Nezhdanova, Xiao Song Lu
Testing	All team members

4.3 Programming Guide and Software Development Environment

The back end code was written entirely in Java, and the front end in ANTLR. All code was managed through CVS. Prior to developing the back end, the interface between the front end and the back end was agreed upon. However, Ling participated in developing both the front-end and the back end, which helped to ensure that there were no inconsistencies. In general, there was often an overlap in responsibilities when it came to both the front end parser and back end, which meant that more than one person, could be working on a specific piece of the source code. Explicit commenting that was time stamped along with the initials of the developer working on it insured that everyone was on the same page for that specific piece of code.

Operating System	Windows XP
Java Runtime Environment	1.4.2
Development Platform	Eclipse 3.1
ANTLR	2.7.5
Version control system	CVS

Because the back end was completely done in Java, we utilized its object oriented philosophy by creating a good set of APIs. By creating well-defined classes and methods, the developers who worked on the ANTLR tree walker would not have to know the specifics on how the back end objects worked. Instead, they relied on the API and trusted the back end developer in implementing the appropriate methods.

The same could be said for Huang-Hsu, the back end developer. He could be assured that the main object he was being fed from the tree walker would meet his specifications exactly. In other words, he did not have to worry about the syntactic sugar of the language; the walker developers transformed whatever form the SFPL source code was in into a well-structured unit that the back end could understand.

4.4 Project Log

9.20 Project topic decided.

9.24 White paper draft done

9.27 White paper draft submitted

9.30 Start Syntax design

10.08 LRM draft done

10.12 Lexer draft done

10.15 LRM finalized

10.20 LRM submitted

10.23 Parser, Lexer finalized, interface developed

10.29 AST tree built

11.15 Creating data types

11.22 Symbol table completed, coding the interpreter

12.01 Coding the built-in functions

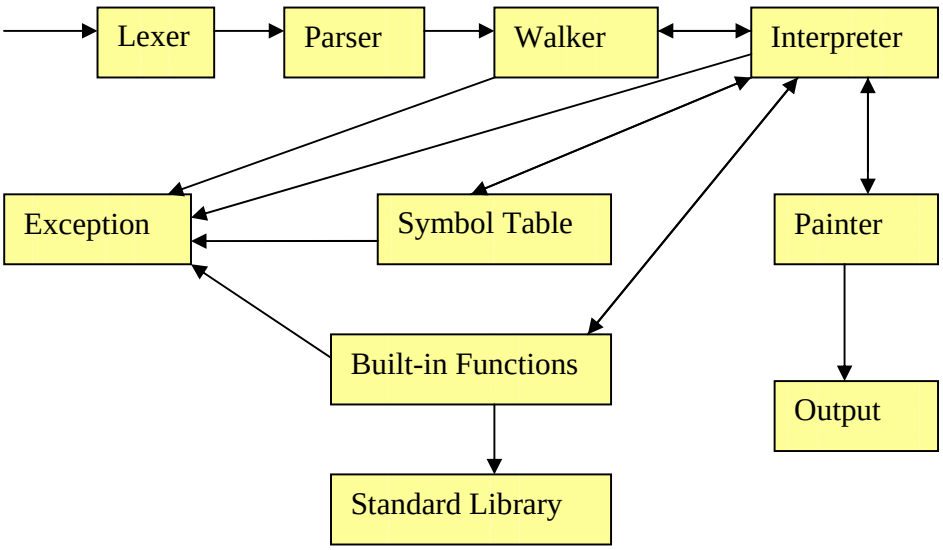
12.10 User-defined functions, standard library and built-in functions implemented.

12.15 Testing finished

Chapter 5

Architectural Design

The SFPL translator consists of the following components: a lexer that reads program files and converts them into tokens. It then outputs the tokens to a parser. The parser analyzes the syntactic structure of the program and outputs an abstract syntax tree based on it. Finally, there is a walker that travels the abstract syntax tree and calls corresponding functions from the backend. The backend consists of the following: an interpreter that looks up symbol tables and diverts operations to type systems, supporting libraries for built-in functions and standard library functions, and a simple error and exception processing mechanism.



The lexer, parser and tree walker are implemented by Antlr. The lexer and parser are implemented in Antlr file grammar.g, and the tree walker is implemented in Antlr file walker.g. Source file walker.g contains short actions calling the methods in the class Intepreter.

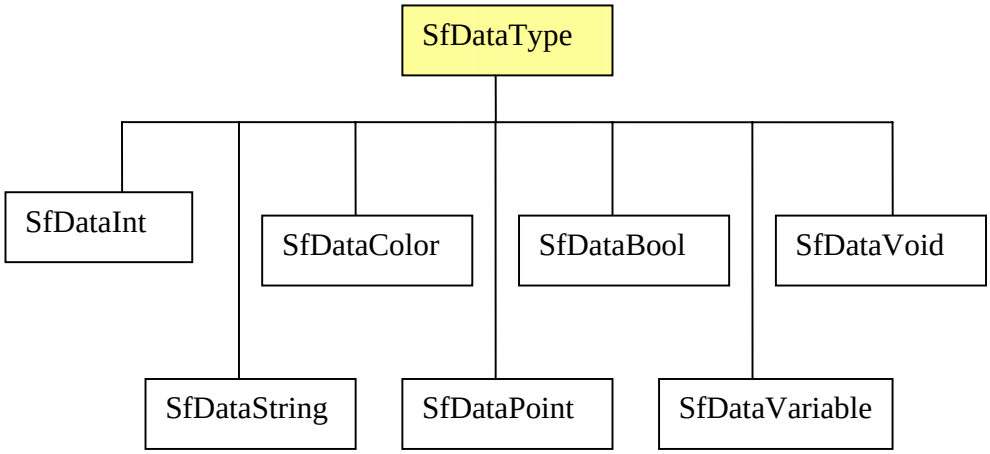
The main() method is in the class Main. It first copies the standard library and the input file to testFile.txt. Then it reads the content of the latter as the input. After that the main() method initializes the painter for the graphic output, the lexer, the parser and the walker. Thus, all the functions in the standard library are automatically included for every SFPL source file.

When reading the function definition, the interpreter puts the function name, return type, the parameters and their types, and the root of the subtree of the function body into the current symbol table. For the variable definition, their names and types are also put in the current symbol table. When assigning a new value to a variable, it goes to the current symbol table to search the variable. If not found it goes up to search the parent of the current symbol table. When a function call occurs, it takes the parameters from the function call statements and tries to match the function definition in the current symbol table. If matched, a child of the current symbol table is created.

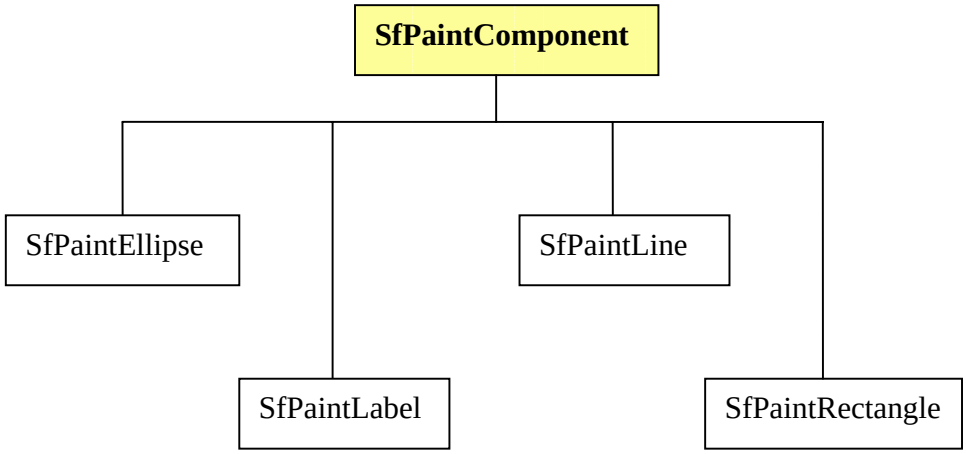
The back end consist two data type systems. One is corresponding to the operation of walker and interpreter, and the other is in charge of displaying our output window and drawing all the objects on the output window.

All the data and variables are encapsulated in SfData* classes, which are derived from a single class SfDataType. This class does the common jobs fot all types and handles the exceptions. If we call an undefined method in the class, it will go to the base class which is SfDataType. The relation among all the SfData* classes is shown in the figure below, where the SfDataType is the base class and all the derived classes are handling specified data type. The SfDataVoid is only handling the return type. All other SfDatatype, such as SfDataBool, SfDataInt, SfDataColor, SfDataPoint and SfDataString, respectively, are simply wrapper classes of Java elemental types which handle the data

in our language. Another special class member is SfFunction which define the structure of user-define functions, it has a function name, and a self symbol table which store the variable define inside, and also a reference to its parent symbol table to retrieve the argument list and the variables defined outside this function.



The second part of the type system in back-end is responsible for the output window. Its structure is shown in the figure below. As we can see, SfPaintComponent is the base class of all the wrapper data classes. The SfBuiltin function will generate plenty of SfPaint* objects. Different SfPaint* classes are constructing different kinds of graphics, colors, ranges and labels on the output window to show the final result of our language.



Chapter 6

Testing Plan

6.1 Unit testing

For the front end, Lexer and Parser (AST tree) were tested separately. For the back end, each java class has been test individually during the coding. Each member was responsible for testing the code once he or she has written it. In addition, some classes like the symbol table have methods to output their contents during testing.

6.2 Basic data types and operations

Each basic data type has been tested with operations such as assignment, logic or arithmetic operations (if applicable) on them. The results were printed in a file or on the screen. Some user-defined functions and control-flow statements were also tested with those types. Moreover, some advanced tests such as nested loops, recursion, nested if-else, and scoping have been run to ensure that those specific features of the SFPL language were captured. Below are several sample tests:

6.2.1 Factorial function

```
function int f (int:a){
if (a == 1) { return 1;}
else{ return a*f(a-1);}
}
int: x;
x = f(5);
print (x);
```

The output is 120. The aim of this test was to ensure the function can recursively call itself.

6.2.2 Dynamic scoping

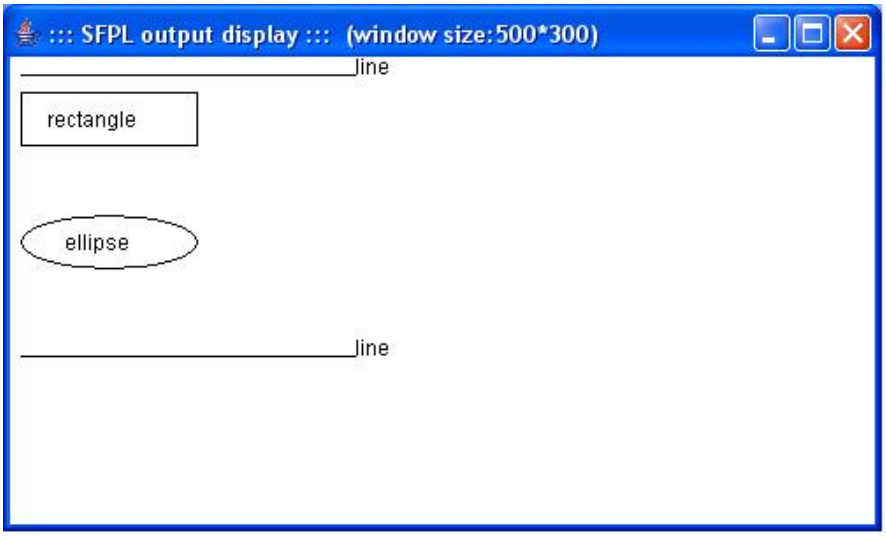
```
function int f (int:a){
 return x+a;}
function int g (int:b) {
int: x;
x = 1;
return f(b);
}
int: x;
x = 0;
print(g(5));
print (f(5));
```

The outputs are 6 and 5. This test shows that the program uses dynamic scoping as it is supposed to.

6.3 Built-in functions and standard library

6.3.1 Built-in function test

```
color: c1,c2;
int: i,j;
point: p1,p2,p3;
c1 = black;
c2 = white;
line (10@40, 200@40, c1);
label (200@40, "line", c1);
rectangle (10@50, 100, 30, c1,c2);
label (25@70, "rectangle",c1);
ellipse ( 10@120, 100,30,c1,c2);
label (35@140, "ellipse", c1);
p1 = 200@10;
p2 = getY(p1)@getX(p1);
p3 = (getX(p2)+190)@getY(p2);
line (p2,p3,c1);
label (p3, "line", c1);
```

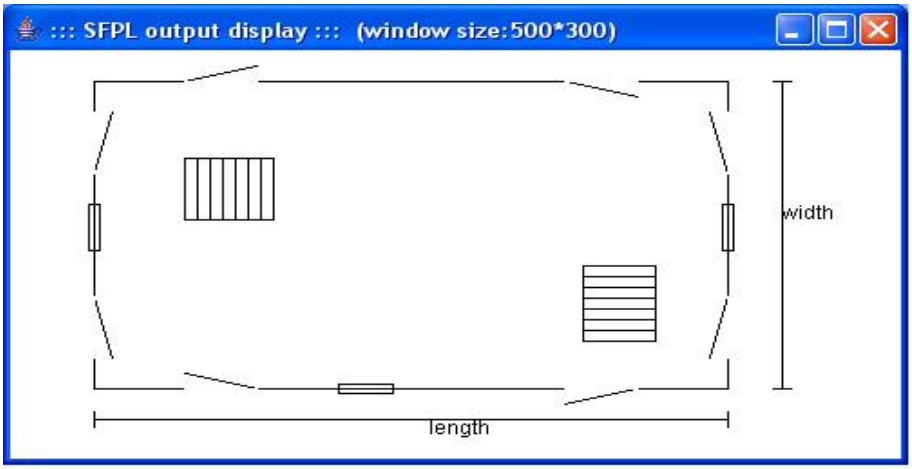


The aim of the above test was to make sure that all the built-in functions' outputs are correct.

6.3.2 Standard library test

```
point: p1,p2,p3,p4;
color:c1,c2;
c2 = #ffffff;
c1 = #000000;
p1 = 50@50;
p2 = 50@250;
p3 = 400@50;
p4 = 400@250;
wall(p1,p2,c1);
wall(p2,p4,c1);
wall(p3,p1,c1);
wall(p3,p4,c1);
door (100@50,140@50, 10, c1, 1, 1, true);
door (310@50, 350@50, 10, c1, 1, 0, true);
```

```
door (310@250, 350@250, 10, c1, 0, 0, true);
door (100@250,140@250, 10, c1, 0, 1, true);
door (50@70,50@110, 10, c1, 1, 1, false);
door (50@190,50@230, 10, c1, 1, 0, false);
door (400@70,400@110, 10, c1, 0, 1, false);
door (400@190,400@230, 10, c1, 0, 0, false);
stair (100@100,7,40,c1,c2,false);
stair (320@170,40,7,c1,c2,true);
window (397@130, 403@160,c1);
window (47@130, 53@160,c1);
window (185@253, 215@247,c1);
dimension (50@270, 400@270, "length" , true, c1);
dimension (430@50, 430@250, "width" , false, c1);
```



The program prints a room with 2 types of stairs, 8 types of doors, two types of dimensions, walls, and windows. The aim of the above test was to show that all the standard library functions produce correct outputs as documented in the Reference Manual.

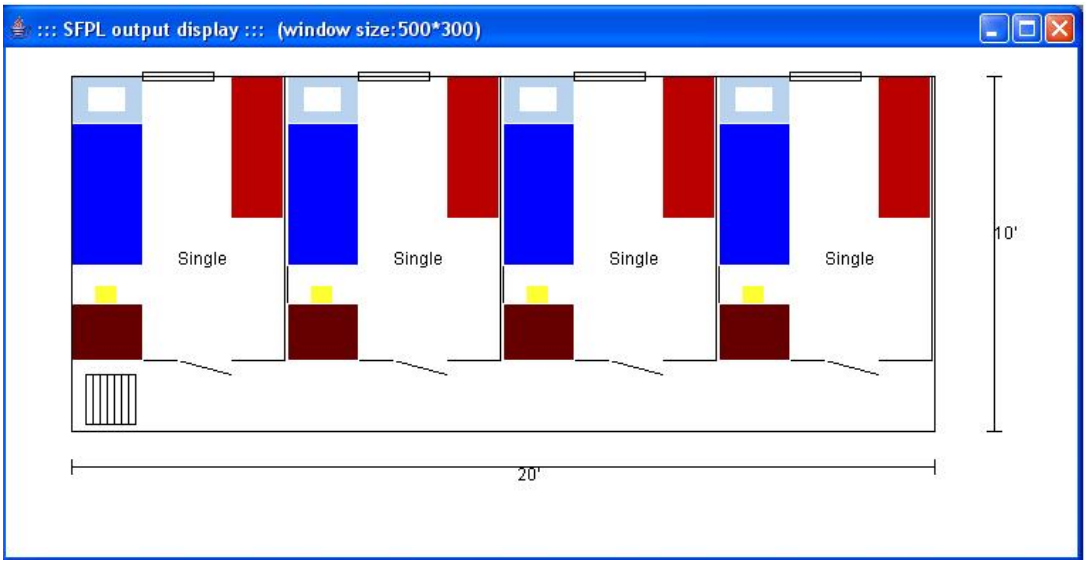
6.4 A Small Demo

This test is just to show that everything works together. It creates several user-defined functions including: bed, cabinet, desk, single (room) and uses a loop mechanism to print four rooms on a floor.

```
function void wall1 (point: p, int: l) //horizontal
{ point:p1;
  p1 = (getX(p)+l)@getY(p);
  line(p,p1, black);
  return;
}
function void wall2 (point: p, int: l) //vertical
{point:p1;
 p1 = getX(p)@(getY(p) + l);

 line(p,p1, black );
  return;
}
function void desk (point:p, int:l, int:w)
{point:p1;
 rectangle(p,l,w,white,brown);
 p1 = (getX(p) + l/3)@ (getY(p) - w/3);
 rectangle(p1,l/3,w/3,white,yellow);
return;
}
function void bed (point: p, int: w, int: l)
{ point:p1,p2;
  color:c;
  c = #B9D3EE;
```

```
  p1 = (getX(p))@ (getY(p) + l/3);
  rectangle (p1,w,l*2/3,white,blue);
  rectangle (p1,w,l,white,blue);
  p2 = (w/4+ getX(p))@(l/12 +  getY(p));
  rectangle(p,w,l/3,white,c);
  rectangle(p2,w/2,l/6,white,white);
  return;
}

function void cabinet (point: p, int: w, int:l)
{ rectangle(p,w,l,white,#bb0000);
  return;
}
function void single (point:p, int: w, int: l)
{ point: p1,p2;

  wall1(p,w);
  wall2(p,l);
  wall1(getX(p)@(getY(p)+l),w);
  wall2((getX(p)+w)@getY(p),l);
  bed(p,w/3,l/2);
  p1 = (getX(p) + w*3/4)@getY(p);
  cabinet(p1, w/4, l/2);
  p1 = getX(p)@(getY(p) + l*4/5);
  desk(p1,w/3,l/5);
  p1 = (getX(p)+w/3)@(getY(p)+3);
  p2 = (getX(p)+w*2/3)@(getY(p)-3);
  window(p1, p2, black);
  p1 = (getX(p)+w/2)@(getY(p)+l);
  p2 = (getX(p)+w*3/4)@(getY(p)+l);
  door(p1,p2,10,black,1,0,true);
  p1 = (getX(p)+w/2)@(getY(p)+l*2/3);
  label (p1, "Single", black);
```

```
  return;
}
point:p,p1,p2,p3,p4;
int : a;
a = 4;
p=50@50;
p1 =p;
while (true)
{single(p,150,200);
 a = a -1;
 if (a==0) {break;}
 p = (getX(p)+152)@50;
}
p2 = getX(p1)@300;
p3 = (getX(p)+152)@50;
p4 = (getX(p)+152)@300;
wall(p1,p2,black);
wall(p2,p4,black);
wall(p1,p3,black);
wall(p4,p3,black);
stair(60@260,5,35,black,white,false);
dimension(700@50,700@300, "10'",false,black);
dimension(50@325,658@325,"20'",true,black);
```



6.5 Other tests

We created many sample programs to test the error handling of our back end, such as programs calling functions with wrong types of parameters. In addition, we used regression methods to make sure that the newly modified code still complies with its specified requirements, and that unmodified code has not been affected by the maintenance activity.

Chapter 7

Lessons Learned

Xiao Song Lu

This was my first large software team project, and I learned a great deal in working as a team member. Firstly, I learned the importance of planning. The key in planning this project is how to plan in the mid-term month. During mid-term month, team members have their own exams to prepare for and consequently, it is very difficult to arrange meetings and do team work together. It is better to divide the jobs early and plan carefully according each member's own schedule. Let each member focus on their own separate job. After mid-term the team can come back and use the rest of the month to complete the whole picture. Communication is also crucial during the midterm period. Our group process was delayed during the mid-term period as we did not realize this problem early. Secondly, I realize a strong leader is crucial for a team project as there needs to be someone to stop endless discussion and make the final say. During the design stage, our group wasted quite a lot of time in discussion due to lack of a team leader. Thus, choosing a team leader is crucial for the success of the team project.Thirdly, the team project provided an opportunity for me to learn from other fellow students. As an inexperienced programmer, I learned many coding skills from my teammates. Finally, I learned that ANTLR is a powerful but poorly documented tool. It is very hard to find right level of examples online. Sometimes, it is wise to just bring the problem to the team advisor.

Ling Zhu

The most interesting part of the process to me was working together to achieve common goals. While I was assigned certain parts of the project to work on, I wanted to keep track of what was going on with the other pieces. This required explanation and periodic briefings from the other team members. When coding with other people, it's fun to see how programming styles and methods vary. One of the best parts about working in a group is that I find that I don't run into so many of the small problems that plague solo projects. If I was unsure about something, I wouldn't hesitate to ask one of my teammates. Having several people also comes in handy when debugging. A fresh set of eyes on your code can bring to light issues which were hidden to you. Finally, although the task seemed daunting at first, and I did not know where to begin, breaking the project down into bite sized pieces helped to clarify each of our roles.

Huang-Hsu Chen

My job is about the back-end which consists of interpreter, datatype systems, output datatype systems and graphical output. With the implementation of back-end, I realized the design of the language is very important and dominates this project. Because during the design stage, we didn't have a big picture of our whole project, our design was not easy for implementation. So there were a lot of problems left to be solved in the back-end. However, we overcame the problems and achieved the goal of the project. The main lesson I learned is how the different data type systems communicate with the interpreter and graphical outputs. And another important things I learned is how to handle the error exceptions during the interpret stage. After all, this project was really interesting, and we spent many nights working at the Clic lab at school. We were excited when our program finally worked.

Natalia Nezhdanova

I was responsible for the documentation of the SFPL project. I've never worked on any documentation before, so it was a new experience for me. Of course, documenting the project required communication with the team members who have actually implemented

it. I enjoyed their company throughout the semester. As the one who couldn't contribute much to the project due to some serious health problems, I wasn't sure if the project as a whole would eventually be completed successfully. But my teammates did a great job, and I'm very grateful to them for being so reliable.

Appendix A

Source Code

A.1 Front End

grammar.g

```
// @author      Ling Zhu (lz2153@columbia.edu)
// @author      Xiao Song Lu (xl2144@columbia.edu)

class SfplAntlrLexer extends Lexer;

 options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = SfplAntlrVocab;
}

{
    int m_errno = 0;
    public void reportError(String strErrorMsg )
    {
        super.reportError(strErrorMsg);
        m_errno++;
    }
```

```
    public void reportError(RecognitionException e)
    {
        super.reportError(e);
        m_errno++;
    }
}


protected
ALPHA
        : 'a'..'z' | 'A'..'Z'
        ;

protected
DIGIT
        : '0'..'9'
        ;

WS
        : (' ' | '\t')+                 { $setType(Token.SKIP); }
    ;


ID      options {testLiterals = true;}
        :       (ALPHA | '_') (ALPHA | '_' | DIGIT)*
        ;


INTEGER
        :       (DIGIT)+
        ;

STRING
        :       ""! (~("" | '\n'))* ""!
        ;

COLOR
```

```
        :       '#'(DIGIT|ALPHA)*
        ;

NEWLINE
        :       (
                        ('\r' '\n') => '\r' '\n' {newline();}
                |       '\r'    {newline();}
                |       '\n'    {newline();}
                )       {$setType(Token.SKIP);}
    ;

COMMENT
        :       (
                        "/*"
                                (
                                        options {greedy = false;}:
                                        (
                                                NEWLINE
                                                |~('\r' | '\n')
                                        )
                                )*
                        "*/"
                        | "//" (~('\r' | '\n'))* NEWLINE
                )       {$setType(Token.SKIP);}
        ;

MULT    : '*';
DIV     : '/';
PLUS    : '+';
MINUS   : '-';
PT              : '@';

GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
```

```
NEQ     : "!=";
AND             : "&&";
OR              : "||";

LPAREN  : '(';
RPAREN  : ')';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
LBRK    : '[';
RBRK    : ']';
ASSIGN  : '=';
COMMA   : ',';
COLON: ':';


/* Parser draft -- NOT finished yet */
class SfplAntlrParser extends Parser;

options
{
   k = 2;
   buildAST = true;
   exportVocab = SfplAntlrVocab;
}

tokens
{
        PROG;
        VAR;
        VAR_DEF_LIST;
        VAR_LIST;
        STATEMENT;
        FUNC_CALL;
        EXPR_LIST;
}
```

```
{
   int m_errno = 0;
   public void reportError(String strErrorMsg )
   {
      super.reportError(strErrorMsg);
      m_errno++;
   }
   public void reportError(RecognitionException e)
   {
      super.reportError(e);
      m_errno++;
   }
}

program
        :       (func_def)* (var_def_list)* (statement)* EOF!
                {#program = #([PROG,"PROG"], program);}
        ;

var_def_list
        :       var (comma_id)* SEMI!
                {#var_def_list = #([VAR_DEF_LIST,"VAR_DEF_LIST"],var_def_list);}
        ;

comma_id
        :       COMMA! ID
                {#comma_id = #([VAR,"VAR"],ID);}
        ;

func_def
        :       "function"^ return_type ID LPAREN! var_list RPAREN! func_body
        ;

var_list
        :       var (COMMA! var)*
                {#var_list = #([VAR_LIST, "VAR_LIST"], var_list);}
        |       /* can be empty */
```

```
                {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

var
        :       basic_type COLON! ID
                {#var = #([VAR, "VAR"],var);}
        ;

return_type
        :       basic_type
        |       "void"
        ;

basic_type
        :       "int"
        |       "string"
        |       "boolean"
        |       "point"
        |       "color"
        ;


func_body
        :       LBRACE! (var_def_list)* (statement)* RBRACE!
                {#func_body = #([STATEMENT,"FUNC_BODY"], func_body);}
        ;

statement
        :       if_stmt
        |       while_stmt
        |       break_stmt
        |       continue_stmt
        |       assign_stmt
        |       return_stmt
        |       func_call_stmt
        |       LBRACE! (statement)* RBRACE!
                {#statement = #([STATEMENT,"STATEMENT"], statement); }
```

```
        ;

if_stmt
    :   "if"^ LPAREN! expression RPAREN! statement
        (options {greedy = true;}: "else"! statement )?
    ;

while_stmt
        :       "while"^ LPAREN! expression RPAREN! statement
        ;

break_stmt
        :       "break" SEMI! /* no need as root */
        ;

continue_stmt
        :       "continue" SEMI! /* no need as root */
        ;

return_stmt
        :       "return"^ (expression)? SEMI!
        ;

assign_stmt
        :       ID ASSIGN^ expression SEMI!
        ;

func_call_stmt
        :       func_call SEMI!
        ;

func_call
        :       ID LPAREN! expr_list RPAREN!
                {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
        ;

expr_list
```

```
        :       expression (COMMA! expression)*
                {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list);}
                | /* can be empty */
                {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list);}
        ;

expression
        :       logic_expr (OR^ logic_expr)*
        ;

logic_expr
        :       relation_expr (AND^ relation_expr)*
        ;

relation_expr
        :       arith_expr ((GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arith_expr)?
        ;

arith_expr
        :       point_expr ((PLUS^ | MINUS^) point_expr)*
        ;

point_expr
        :       arith_term (PT^ arith_term)*
        ;

arith_term
        :       factor ((MULT^ | DIV^) factor)*
        ;

factor
        :       ID
        |       func_call
        |       INTEGER
        |       STRING
        |       COLOR
        |       "true"
```

```
|       "false"
|  "green"
|  "grey"
|  "black"
|  "blue"
|  "orange"
|  "white"
|  "brown"
|  "yellow"
|  "red"
|  "purple"
|       LPAREN! expression RPAREN!
;
```

walker.g

```
// @author      Ling Zhu (lz2153@columbia.edu)
// @author      Xiao Song Lu (xl2144@columbia.edu)

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

class SfplAntlrWalker extends TreeParser;
options
{
   importVocab = SfplAntlrVocab;
}

{
   static SfDataType null_data = new SfDataType();
   Intepreter ipt = new Intepreter();
   // for plist and exprlist
   Vector v_vector = new Vector();
   Vector expr_vector = new Vector();
}

expr returns [ SfDataType r ]
{
   SfDataType a,b,rt,rtype;
   SfDataType[] sx = null;
   a = null;
   b = null;
   rt = null;
   rtype = null;
   Vector v;
   r = null_data;
```

```
    SfSymbolTable symb = null;
    String varType = null;
}

        : #(OR a=expr right_or:.)
            {
                if ( a instanceof SfDataBool )
                    r = ( ((SfDataBool)a).value ? a : expr(#right_or) );
                else
                    r = a.or( expr(#right_or) );
            }
        | #(AND a=expr right_and:.)
            {
                if ( a instanceof SfDataBool )
                    r = ( ((SfDataBool)a).value ? expr(#right_and) : a );
                else
                    r = a.and( expr(#right_and) );
            }

         | #(PROG  (stmt:. { if ( ipt.canProceed() )
                            {
                                    r = expr(#stmt);
                            }
                           } )*)

         | #(STATEMENT (smt:. { if ( ipt.canProceed() )
                                {
                                        r = expr(#smt);
                                }
                                 } )*)

        |    #(VAR_DEF_LIST (vardef:. { r = expr(#vardef);})*)

        |    #(variable:VAR a=expr (b=expr)?) {
                                                                                int childnum =
variable.getNumberOfChildren();
```

```
                                                                    if (childnum ==
1) /* use a */
        ipt.setVariable(a);
                                                                    if (childnum ==
2)  /* use b */
        ipt.setVariable(b);
                                                                    }
        |       #(FUNC_CALL a=expr sx=exprlist) { r = ipt.funcInvoke( this, a, sx);}
    |   #(ASSIGN a=expr b=expr) { r = ipt.assign(a, b); }
    |   #(PLUS a=expr b=expr)   { r = a.plus( b ); }
        |       #(MINUS a=expr b=expr)   { r = a.minus( b ); }
    |   #(DIV  a=expr b=expr)   { r = a.divide( b ); }
    |   #(MULT a=expr b=expr)   { r = a.time( b ); }
    |   #(GE a=expr b=expr)       { r = a.ge( b ); }
    |   #(LE a=expr b=expr)       { r = a.le( b ); }
    |   #(GT a=expr b=expr)       { r = a.gt( b ); }
    |   #(LT a=expr b=expr)       { r = a.lt( b ); }
    |   #(EQ a=expr b=expr)       { r = a.eq( b ); }
        |   #(NEQ a=expr b=expr)      { r = a.ne( b ); }
                |   #(PT  a=expr b=expr)      {if ((a instanceof SfDataInt)&& (b instanceof
SfDataInt))
```

```
                                        r = new SfDataPoint( ((SfDataInt)a).intValue(a),
((SfDataInt)b).intValue(b));
                                    }

        |       #(w:"while" a=expr loopbody:.) {

        while(ipt.loopCanProceed( expr(w.getFirstChild())))
                                                                                    {

        if(ipt.getControl() == Intepreter.fc_break)

        break;

        r = expr(#loopbody);

        if(ipt.getControl() == Intepreter.fc_continue)

        {

        ipt.reset();

        continue;
                                                                                        }
                                                                                    }

        ipt.reset();
                                                                            }

        |       #("if" a=expr thenp:. (elsep:.)?)
                        {
                                if ( !( a instanceof SfDataBool ) )
                                return a.error( "if: expression should be bool" );
                                if ( ((SfDataBool)a).value )
                                r = expr( #thenp );
                                else if ( null != elsep )
                                r = expr( #elsep );
```

```
                    }
|   "break"    { ipt.setBreak(); }

|   "continue" { ipt.setContinue(); }

|   #("function"
                        rt=expr {
                                        varType = ipt.getCurrVarType();
                                        rtype = ipt.getReturnType(varType);
                                }
                        fname:ID sx=vlist fbody:.)
                                                        {

    ipt.funcRegister(fname.getText(), rtype, sx, #fbody);
                                                        }

|   "point" { ipt.setCurrVarType("point"); }

|   "color" { ipt.setCurrVarType("color"); }

|   "int"      { ipt.setCurrVarType("int"); }

|   "string" { ipt.setCurrVarType("string"); }

|   "boolean" { ipt.setCurrVarType("boolean"); }

|   "void" { ipt.setCurrVarType("void"); } //no problem

|   i:INTEGER { r = new SfDataInt(Integer.parseInt(i.getText())); }

|   s:STRING { r = new SfDataString(s.getText()); }

|   "true"    { r = new SfDataBool(true); }

|   "false"   {r = new SfDataBool(false);}
```

```
    |   c:COLOR         { r = new SfDataColor(c.getText()); }

  //change "#colorname" to "colorname"
                |   "grey" { r = new SfDataColor( "#666666");}
                |   "green" { r = new SfDataColor( "#006600");}
                |   "black" { r = new SfDataColor( "#000000");}
                |   "blue"  { r = new SfDataColor( "#0000FF");}
                |   "orange" { r = new SfDataColor( "#FF6600");}
                |   "white"  { r = new SfDataColor( "#FFFFFF");}
                |   "brown"  { r = new SfDataColor( "#660000");}
                |   "yellow"  { r = new SfDataColor( "#FFFF33");}
                |   "red"    { r = new SfDataColor( "#FF0000");}
                |   "purple"  { r = new SfDataColor( "#990099");}

    |   id:ID { r = ipt.getVariable(id.getText()); }

                |           #(re:"return" ( a=expr
                )?
        )                   {
                                                                int num =
re.getNumberOfChildren();
                                                                if(num == 0)
                                                                        r = new
SfDataVoid();
                                                                if(num == 1)
                                                                        r = a;
                                                                ipt.setBreak();
                                                        }
    ;

pexpr returns [SfDataType r] // called by vlist
{
        r = null;
}

                :           "point" { ipt.setCurrVarType("point"); }
```

```
    |   "color" { ipt.setCurrVarType("color"); }

    |   "int"     { ipt.setCurrVarType("int"); }

    |   "string" { ipt.setCurrVarType("string"); }

    |   "boolean" { ipt.setCurrVarType("boolean"); }

    |   id:ID     { r = new SfDataVariable(id.getText());}
    ;

vlist returns [ SfDataType[] pv ] // variable list in function definition part
{
   pv = null;
   SfDataType a, b;
   SfDataType[] r;
   a = null;
   b = null;
   r = null;
}
                    :           #(VAR_LIST (pdef:. { r = vlist(#pdef);})*) {

                                pv = ipt.convertToVarlist(v_vector);

                                v_vector.clear();

            }

    |   #(VAR a=pexpr b=pexpr) {
                                                                SfDataType p =
ipt.newParameter(b);
                                                                v_vector.add(p);
                                                        }
                    ;

exprlist returns [ SfDataType[] ev ]
{
```

```
    SfDataType a;
    ev = null;
}
                    :           #(EXPR_LIST
                                        ( a=expr      { expr_vector.add(a); }
              )*
             )              { ev = ipt.convertToVarlist(expr_vector);
                                                    expr_vector.clear();}
                    ;
```

A.2 Back End

Intepreter.java

```
/**
 * Class Intepreter
 *
 * @author        Ling Zhu (lz2153@columbia.edu)
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 */

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



public class Intepreter
{
        SfSymbolTable currSymbolTable = null;
        String currFuncName;
        String varType;

        final static int fc_none = 0;
    final static int fc_break = 1;
    final static int fc_continue = 2;
    final static int fc_return = 3;

    private int control = fc_none;

        public Intepreter()
        {
```

```
        control = fc_none;
        setCurrFunc("main");
        this.registerBuiltin();

        // to initiate Vector
        SfPainter.vDraw = new Vector();
}


public void setCurrFunc(String funcname)
{
        SfSymbolTable parent = this.currSymbolTable;
        this.currSymbolTable = new SfSymbolTable(funcname, parent);
        this.currFuncName = funcname;
}


public SfSymbolTable getCurrSymbolTable()
{
        return currSymbolTable;
}

public int getControl()
{
        return this.control;
}

public void setBreak()
{
        this.control = Intepreter.fc_break;
}

public void setContinue()
{
        this.control = Intepreter.fc_continue;
}
```

```
	public void reset()
	{
		this.control = Intepreter.fc_none;
	}

	public void setCurrVarType(String vtype)
	{
		this.varType = vtype;
	}

	public String getCurrVarType()
	{
		return this.varType;
	}

	public SfDataType getVariable( String s )
	{
		SfDataType x = null;
		x = this.currSymbolTable.getValue(s);

		if(x == null) // new variable
			x = new SfDataVariable(s);

		return x;
}

	public SfDataType getReturnType(String type)
	{
		SfDataType p = null;

		if(type.equals("point"))
			p = new SfDataPoint();
		if(type.equals("color"))
			p = new SfDataColor();
		if(type.equals("int"))
			p = new SfDataInt();
		if(type.equals("string"))
```

```
                p = new SfDataString();
        if(type.equals("boolean"))
                p = new SfDataBool();
        if(type.equals("void"))
                p = new SfDataVoid();

        return p;
}

public SfDataType newParameter(SfDataType p)
{
        SfDataType param = null;
        String pname = p.getVarName();
        String ptype = this.getCurrVarType();

        if(p instanceof SfDataVariable)
        {
                if(ptype.equals("point"))
                {
                        param = new SfDataPoint();
                        param.setVarName(pname);
                }
                if(ptype.equals("color"))
                {
                        param = new SfDataColor();
                        param.setVarName(pname);
                }
                if(ptype.equals("int"))
                {
                        param = new SfDataInt();
                        param.setVarName(pname);
                }
                if(ptype.equals("string"))
                {
                        param = new SfDataString();
                        param.setVarName(pname);
                }
```

```
                if(ptype.equals("boolean"))
                {
                        param = new SfDataBool();
                        param.setVarName(pname);
                }
        }

        return param;
}

public SfDataType[] convertToVarlist(Vector v)
{
        SfDataType[] varlist = null;
        if(v.size() > 0)
                varlist = new SfDataType[v.size()];

        if(varlist != null)
        {
                for(int i=0; i<v.size(); i++)
                {
                        SfDataType p = (SfDataType)v.elementAt(i);
                        varlist[i] = p;
                }
        }

        return varlist;
}


public void setVariable(SfDataType v)
{
        String varName = v.getVarName();
        SfSymbolTable symb = this.getCurrSymbolTable();
        String varType = this.getCurrVarType();

        if(!(v instanceof SfDataVariable))
        {
```

```
                if(symb.containsVar(varName)) // already exists;
                        v.error("Variable " + varName + " already exists..");
        }
        // create different data types based on the variable type
        if(varType.equals("point"))
        {
                SfDataPoint newv = new SfDataPoint();
                newv.setVarName(v.getVarName());
                symb.insert(newv);
        }
        if(varType.equals("color"))
        {
                SfDataColor newv = new SfDataColor();
                newv.setVarName(v.getVarName());
                symb.insert(newv);
        }
        if(varType.equals("int"))
        {
                SfDataInt newv = new SfDataInt();
                newv.setVarName(v.getVarName());
                symb.insert(newv);
        }
        if(varType.equals("string"))
        {
                SfDataString newv = new SfDataString();
                newv.setVarName(v.getVarName());
                symb.insert(newv);
        }
        if(varType.equals("boolean"))
        {
                SfDataBool newv = new SfDataBool();
                newv.setVarName(v.getVarName());
                symb.insert(newv);
        }

        return;
}
```

```
	public SfDataType assign(SfDataType a, SfDataType b)
	{
		if((a instanceof SfDataVariable))
			return a.error("Variable " + a.getVarName() + " not decclared..");
		if((b instanceof SfDataVariable))
			return a.error("Variable " + b.getVarName() + " not decclared..");

		SfDataType x = a.assign(b);
		x.setVarName(a.getVarName());

		this.currSymbolTable.setValue(x.getVarName(), x);
		return x;
	}

	public boolean loopCanProceed(SfDataType a)
	{
		boolean result = false;
		if(a instanceof SfDataBool)
		{
			if(SfDataBool.boolValue(a) == true)
				result = true;
		}

		return result;
	}

	public void loopEnd()
	{
		control = fc_none;
	}


public void registerBuiltin() {
	SfBuiltinFunction.register( currSymbolTable );
}
```

```
        // modified by Huang-Hsu
        // to execute the Builtin functions
        public SfDataType execBuiltin(int id, SfDataType[] params)
        {
                return SfBuiltinFunction.execute(currSymbolTable, id, params );
        }

        public void funcRegister( String name, SfDataType return_type,
                                                        SfDataType[] args, AST body )
        {
                SfFunction func = new SfFunction(return_type, name, args, body,
this.currSymbolTable);
                this.currSymbolTable.put( name, func);
   }

        // modified by Huang-Hsu @ 1130
        public SfDataType funcInvoke(   SfplAntlrWalker walker,
                                                                SfDataType func,
                                                                SfDataType[] params )
throws antlr.RecognitionException
        {
                // func must be an existing function
      if ( !( func instanceof SfFunction ) )
         return func.error( "not a function" );

      // Builtin Functions
      // Names of formal args are not necessary for Builtin Functions
      if ( ((SfFunction)func).isInternal() )
         return execBuiltin( ((SfFunction)func).getId(), params );

      // -----------------------------------------------------
      // ------------- user defined functions ----------------

      // otherwise check numbers of actual and formal arguments
      SfDataType[] args = ((SfFunction)func).getArgs();
      if(args != null && params != null)
         if ( args.length != params.length )
```

```
                return func.error( "unmatched length of parameters" );

// create a new symbol table
SfSymbolTable parent = this.currSymbolTable;
currSymbolTable = new SfSymbolTable(((SfFunction)func).getVarName(), parent);
this.currFuncName = currSymbolTable.getFuncName();

if(args != null && params != null)
{
    // put args into symbol table
    for(int i=0; i<args.length; i++)
    this.currSymbolTable.insert(args[i]);
    // assign actual parameters to formal arguments
   for ( int i=0; i<args.length; i++ )
   {
      SfDataType p = args[i];
      this.assign(p, params[i]);
   }
}

// call the function body
SfDataType r = walker.expr( ((SfFunction)func).getBody() );

// check whether return_type matches
SfDataType return_type = ((SfFunction)func).getReturntype();
if(!return_type.typeName().equals(r.typeName()))
    return func.error("unmatched return type");

// reset control
this.reset();

// remove this symbol table and return
this.currSymbolTable = currSymbolTable.getParent();
this.currFuncName = this.currSymbolTable.getFuncName();

                return r;
    }
```

```
        public boolean canProceed()
        {
      return control == fc_none;
   }
}
```

Main.java

```
/**
 * Class Main
 *
 * @author        Ling Zhu (lz2153@columbia.edu)
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 */

import java.io.*;

import javax.swing.JFrame;

import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class Main {

	public static void main(String args[])
	{

		try
		{
			// read STL
			File fSTL = new File("STL.txt");

			FileInputStream fi = new FileInputStream(fSTL);
		DataInputStream inputSTL = new DataInputStream(fi);

		// read fileName from command line
		File f = new File(args[0]);
		FileInputStream fi2 = new FileInputStream(f);
		DataInputStream inputCmd = new DataInputStream(fi2);

		FileOutputStream fw = new FileOutputStream("testFile.txt",false);
		// write into testFile.txt
```

```
while( inputSTL.available()>0)
{
        fw.write( inputSTL.readByte() );
}
while( inputCmd.available()>0)
{
        fw.write( inputCmd.readByte() );
}
inputSTL.close();
inputCmd.close();
fw.close();
// -------------------------------------------------------------
FileInputStream fiInput = new FileInputStream("testFile.txt");
DataInputStream input = new DataInputStream(fiInput);
// end file I/O
// -------------------------------------------------------------
// SfPainter
SfPainter sp;


// Create the lexer and parser and feed them the input
SfplAntlrLexer lexer = new SfplAntlrLexer(input);
SfplAntlrParser parser = new SfplAntlrParser(lexer);
parser.program();


// Get the AST from the parser
CommonAST parseTree = (CommonAST)parser.getAST();

        // invoke walker
        SfplAntlrWalker walker = new SfplAntlrWalker();
        SfDataType r = walker.expr( parseTree );

        // invoke SfPainter's constructor to draw 2D output
        sp = new SfPainter("::: SFPL output display :::");
        sp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
                }
                catch(Exception e)
                {
                        System.err.println("Exception: "+e);
                }
        }
}
```

SfBuiltinFunction.java

```
import java.util.Vector;
/**
 * SfBuiltinFunction
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v2
 * @since         2005.12.01
 */

// Builtin Function to implelement Standard Library
public class SfBuiltinFunction {

        // function ID to store in Symbol Table
        final static int f_print = 0;
        final static int f_what = 1;
        final static int f_line = 2;
        final static int f_rectangle = 3;
        final static int f_ellipse = 4;
        final static int f_label = 5;
        final static int f_helloworld = 6;
        final static int f_getX = 7;
        final static int f_getY = 8;


        // nothing in constructor
        SfBuiltinFunction()
        {

        }

        // to register all built-in functions
        public static void register( SfSymbolTable st )
        {
                //st.put( "print", new MxFunction( null, f_print ) );
                // put all built-in functions into symbol table
```

```
// and set name of all functions at the mean time
                // argument: id, SfFunction
                st.put( "print", new SfFunction( "print", f_print ) );
st.put( "what", new SfFunction( "what", f_what ) );
st.put( "line", new SfFunction( "line", f_line ) );
st.put( "rectangle", new SfFunction( "rectangle", f_rectangle ) );
st.put( "ellipse", new SfFunction( "ellipse", f_ellipse ) );
st.put( "label", new SfFunction( "label", f_label ) );
st.put( "getX", new SfFunction( "getX", f_getX ) );
st.put( "getY", new SfFunction( "getY", f_getY ) );
st.put( "helloworld", new SfFunction( "helloworld", f_helloworld ) );

    }


    // to execute the builtin functions
    public static SfDataType execute(SfSymbolTable st,
                                                        int id,
                                                        SfDataType[] params )
    {
                // debug
                //System.out.println("id:"+id);
                switch ( id )
                {
            case f_print:
            {
                        for ( int i=0; i<params.length; i++ )
                   params[i].print();
                return null;
            }
            case f_what:
            {
                if ( params.length > 0 )
                {
                   for ( int i=0; i<params.length; i++ )
                          params[i].what();
                }
```

```
    else
        st.what();
    return null;
}
case f_line:
{
    int x1 = ((SfDataPoint)params[0]).x;
    int y1 = ((SfDataPoint)params[0]).y;
    int x2 = ((SfDataPoint)params[1]).x;
    int y2 = ((SfDataPoint)params[1]).y;
    String color = ((SfDataColor)params[2]).value;
    SfPaintLine newLine = new SfPaintLine(x1,y1,x2,y2,color);
    // debug
    //newLine.print();
    SfPainter.vDraw.add(newLine);
    return null;
}
// rectangle( point p, int w, in l, color c1, color c2 )
case f_rectangle:
{
    int x1 = ((SfDataPoint)params[0]).x;
    int y1 = ((SfDataPoint)params[0]).y;
    int w = ((SfDataInt)params[1]).value;
    int l = ((SfDataInt)params[2]).value;
    String c1 = ((SfDataColor)params[3]).value;
    String c2 = ((SfDataColor)params[4]).value;
    SfPaintRectangle newRec = new SfPaintRectangle(x1,y1,w,l,c1,c2);
    // debug
    //newRec.print();
    SfPainter.vDraw.add(newRec);
    return null;
}
// ellipse( point p, int r1, in r2, color c1, color c2 )
case f_ellipse:
{
    int x1 = ((SfDataPoint)params[0]).x;
    int y1 = ((SfDataPoint)params[0]).y;
```

```
    int r1 = ((SfDataInt)params[1]).value;
    int r2 = ((SfDataInt)params[2]).value;
    String c1 = ((SfDataColor)params[3]).value;
    String c2 = ((SfDataColor)params[4]).value;
    SfPaintEllipse newEll = new SfPaintEllipse(x1,y1,r1,r2,c1,c2);
    // debug
    //newEll.print();
    SfPainter.vDraw.add(newEll);
    return null;
}
// label( point p, string l, color c )
case f_label:
{
    int x1 = ((SfDataPoint)params[0]).x;
    int y1 = ((SfDataPoint)params[0]).y;
    String l= ((SfDataString)params[1]).value;
    String c = ((SfDataColor)params[2]).value;
    SfPaintLabel newLabel = new SfPaintLabel(l,x1,y1,c);
    // debug
    //newLabel.print();
    SfPainter.vDraw.add(newLabel);
    return null;
}
case f_helloworld:
{
    helloworld();
}
case f_getX:
{
    SfDataInt x = new SfDataInt( ((SfDataPoint)params[0]).x );
    return x;
}
case f_getY:
{
    SfDataInt y = new SfDataInt( ((SfDataPoint)params[0]).y );
    return y;
}
```

```
        default:
            throw new SfException( "unknown builtin function" );
            }

}

// ---------------------------------------------------------
public SfDataType print(SfSymbolTable st, SfDataType[] params)
{
            return null;
}

//
public SfDataType what(SfSymbolTable st, SfDataType[] params)
{
            return null;
}

// to draw line by initiating a paintComponent
public SfDataType line(SfSymbolTable st, SfDataType[] params)
{
            //g.drawLine(0,0,20,20);
            return null;
}

// to draw rectangle by initiating a paintComponent
public SfDataType rectangle(SfSymbolTable st, SfDataType[] params)
{
            //g.drawRect(10,10,50,60);
            return null;
}

// to draw ellipse by initiating a paintComponent
public SfDataType ellipse(SfSymbolTable st, SfDataType[] params)
{
            return null;
}
```

```
	//to draw label by initiating a paintComponent
	public SfDataType label(SfSymbolTable st, SfDataType[] params)
	{
		return null;
	}


	// hellowrold can be used only once in the code, cause hanging problem
	public static void helloworld()
	{
		System.out.println("Hello World!!!!!!!!!!");
	}









	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

		SfBuiltinFunction test = new SfBuiltinFunction();
		//test.line(test.getGraphics());


	}

}
```

SfDataBool.java

```
import java.io.PrintWriter;
/**
 * Base class SfDataBool
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v2
 * @since         2005.11.07
 */

public class SfDataBool extends SfDataType{

    public boolean value;   //Made it public!!!

    // default constructor
    public SfDataBool(  ) {

    }


    public SfDataBool( boolean value ) {
        this.value = value;
    }

    public void set(boolean value)
    {
            this.value = value;
    }

    public String typeName() {
        return "bool";
    }

    public SfDataType copy() {
```

```
    return new SfDataBool( value );
}

// assistance for assign
public static boolean boolValue( SfDataType b ) {
        if ( b instanceof SfDataBool )
       return ((SfDataBool)b).value;
    b.error( "cast to Bool" );
    return false;           // default = false !!!!!!!!!!!! maybe be bug
}

// assign Bool = Bool
public SfDataType assign(SfDataType b)
{
        if(b instanceof SfDataBool)
        {
                this.set(boolValue(b));
                return this.copy();
        }
        return error(b , "Error @ assign ");
}
//-----------------------------------------------------

public void print( PrintWriter w ) {
    if ( this.varName != null )
       w.print( this.varName + " = " );
    w.println( value ? "true" : "false" );
}

// Int == Int
public SfDataType eq( SfDataType b ) {
    if ( b instanceof SfDataBool )
       return new SfDataBool( value == ((SfDataBool)b).value);
    return error(b , "Error @ bool == bool ");
}

// Int != Int
```

```
    public SfDataType ne( SfDataType b ) {
        if ( b instanceof SfDataBool )
            return new SfDataBool( value != ((SfDataBool)b).value );
        return error(b , "Error @ bool != bool ");
    }
    //-------------------------------------------------
    public SfDataType and( SfDataType b ) {
        if ( b instanceof SfDataBool )
            return new SfDataBool( value && ((SfDataBool)b).value );
        return error( b, "and" );
    }

    public SfDataType or( SfDataType b ) {
        if ( b instanceof SfDataBool )
            return new SfDataBool( value || ((SfDataBool)b).value );
        return error( b, "or" );
    }

            public static void main(String[] args) {
                    // TODO Auto-generated method stub
                    SfDataBool testBool = new SfDataBool (true);
                    testBool.print();
                    testBool.what();
            }

}
```

SfDataColor.java

```
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.regex.*;

/**
 * datatype: color
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.22
 */

public class SfDataColor extends SfDataType{

        String value;

   // default constructor
   public SfDataColor(  ) {

   }

        // constructor
        public SfDataColor(String i)
        {
                this.value = i;
        }

        public void set (String i)
        {
                this.value = i;
        }

        // return typename()
```

```
        public String typeName() {
    return "color";
}

        // copy
public SfDataType copy() {
    return new SfDataColor ( value );
}

// print ()
public void print( PrintWriter w ) {
    if ( this.varName != null )
        w.print( this.varName + " = " );
    w.println(  value  );
}

// assistance for assign
public static String colorValue( SfDataType b ) {
        if ( b instanceof SfDataColor )
        return ((SfDataColor)b).value;
    b.error( "cast to Color" );
    return null;
}

// assign Color = Color
public SfDataType assign(SfDataType b)
{
        if(b instanceof SfDataColor)
        {
                this.set(colorValue(b));
                return this.copy();
        }
        return error(b , "Error @ assign ");
}

// Color == Color
public SfDataType eq( SfDataType b ) {
```

```
        if ( b instanceof SfDataColor )
            return new SfDataBool( value == colorValue(b) );
        return error(b , "Error @ Color == Color ");
    }

    // Color != Color
    public SfDataType ne( SfDataType b ) {
        if ( b instanceof SfDataColor )
            return new SfDataBool( value != colorValue(b) );
        return error(b , "Error @ Color != Color ");
    }


    // color type don't allow any other operation on it
    // like + , - , < , > , >= , <=

    //  to check if the input string is accepted or not
            // color format : "#ffffff" or "#1f2f23"
            static public boolean matchColor(String input)
            {
                    Pattern p = Pattern.compile("#[a-f|A-F|0-9][a-f|A-F|0-9][a-f|A-F|0-9]" +
                                                                    "[a-f|A-F|0-9][a-f|A-F|0-
9][a-f|A-F|0-9]");
                    Matcher m = p.matcher(input);

                    return m.matches();
            }

            /**
             * @param args
             */
            public static void main(String[] args) {
                    // TODO Auto-generated method stub

                    System.out.println("#aaab is:"+SfDataColor.matchColor("#A23F32"));

                    String color = "#af2312";
```

```
                if(SfDataColor.matchColor(color))
                {
                        SfDataColor c = new SfDataColor(color);
                        c.print();
                        System.out.println("c type is:"+c.typeName());

                }
        }

}
```

SfDataInt.java

```
/**
 * Int datatype
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.07
 */


import java.io.PrintWriter;

public class SfDataInt extends SfDataType{

        // integer value
   int value;

   // default constructor
   public SfDataInt ( ) {

   }

   public SfDataInt ( int input ) {
      value = input;
   }

   public void set(int input ){
        value = input;
   }

   // type: int
   public String typeName() {
      return "int";
   }
```

```
// copyconstructor: int
public SfDataType copy() {
   return new SfDataInt ( value );
}

// for the comparison
public static int intValue( SfDataType b ) {
        if ( b instanceof SfDataInt )
       return ((SfDataInt)b).value;
   b.error( "cast to int" );
   return 0;
}

// for debug
public void print( PrintWriter w ) {
   if ( this.varName != null )
      w.print( this.varName + " = " );
   w.println( Integer.toString( value ) );
}


// modified by Ling
// Int = Int = Int
public SfDataType assign(SfDataType b)
{
        if(b instanceof SfDataInt)
        {
                this.set(intValue(b));
                return this.copy();
        }
        return error(b , "Error @ assign ");
}

// Int = Int + Int
public SfDataType plus( SfDataType b ) {
   if ( b instanceof SfDataInt )
      return new SfDataInt( value + intValue(b) );
```

```
    return error(b , "Error @ Int + Int ");
}


// Int = Int - Int
public SfDataType minus( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataInt( value - intValue(b) );
    return error(b , "Error @ Int - Int ");
}

// Int = Int * Int
public SfDataType time( SfDataType b ) {
   if ( b instanceof SfDataInt )
        return new SfDataInt( value * intValue(b) );
   return error(b , "Error @ Int * Int ");
}

// Int = Int / Int
public SfDataType divide( SfDataType b ) {
   if ( b instanceof SfDataInt )
        return new SfDataInt( value / intValue(b) );
   return error(b , "Error @ Int / Int ");
}

// Int > Int
public SfDataType gt( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value > intValue(b) );
    return error(b , "Error @ Int > Int ");
}

// Int >= Int
public SfDataType ge( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value >= intValue(b) );
    return error(b , "Error @ Int >= Int ");
```

```
}

// Int < Int
public SfDataType lt( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value < intValue(b) );
    return error(b , "Error @ Int < Int ");
}

// Int <= Int
public SfDataType le( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value <= intValue(b) );
    return error(b , "Error @ Int <= Int ");
}

// Int == Int
public SfDataType eq( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value == intValue(b) );
    return error(b , "Error @ Int == Int ");
}

// Int != Int
public SfDataType ne( SfDataType b ) {
    if ( b instanceof SfDataInt )
        return new SfDataBool( value != intValue(b) );
    return error(b , "Error @ Int != Int ");
}

        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }
```

```
}
```

SfDataPoint.java

```
import java.io.PrintWriter;

/**
 * Point
 *
 * @author      Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since       2005.11.22
 */
public class SfDataPoint extends SfDataType{
        int x;
        int y;

        // default constructor
        public SfDataPoint()
        {

        }

        public SfDataPoint(int ix, int iy)
        {
                this.x = ix;
                this.y = iy;
        }


   public void set( int ix, int iy)
        {
                this.x = ix;
                this.y = iy;
        }
```

```
        // return typename()
        public String typeName() {
    return "point";
  }

        // copy
  public SfDataType copy() {
    return new SfDataPoint ( this.x, this.y );
  }

//  ---------------------------------------------------------------------

  // assign Point
  public SfDataType assign(SfDataType b)
  {
        if(b instanceof SfDataPoint)
        {
                this.set( ((SfDataPoint)b ).x , ((SfDataPoint)b ).y) ;
                return this.copy();
        }
        return error(b , "Error @ assign ");
  }
  //---------------------------------------------------------------------

  // print point()
  public void print( PrintWriter w ) {
    if ( this.varName != null )
       w.print( this.varName + " = " );
    w.println(  this.x + "@" + this.y  );
  }

  // point + point
  public SfDataType plus(SfDataType back)
  {
        if ( back instanceof SfDataPoint )
                return new SfDataPoint(this.x+ ((SfDataPoint)back).x ,
                                                        this.y+ ((SfDataPoint)back).y );
```

```
        return error(back , "Error @ Point + Point ");
    }

    // point - point
    public SfDataType minus(SfDataType back)
    {
        if ( back instanceof SfDataPoint )
                return new SfDataPoint(this.x- ((SfDataPoint)back).x ,
                                                        this.y- ((SfDataPoint)back).y );
        return error(back , "Error @ Point - Point ");
    }


    // there are problem in this field
//    // point == point
//    public SfDataType eq( SfDataType b ) {
//        if ( b instanceof SfDataColor )
//            return new SfDataBool( value == intValue(b) );
//        return error(b , "Error @ point == point ");
//    }
//
//    // point != point
//    public SfDataType ne( SfDataType b ) {
//        if ( b instanceof SfDataColor )
//            return new SfDataBool( value != intValue(b) );
//        return error(b , "Error @ point != point ");
//    }

        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                SfDataPoint p1 = new SfDataPoint(3,4);
                p1.print();
                SfDataPoint p2 = new SfDataPoint(2,3);
                (p1.plus(p2)).print();
```

```
                (p1.minus(p2)).print();


        }

}
```

SfDataString.java

```
import java.io.PrintWriter;

/**
 * Base class datatype
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.07
 */

public class SfDataString extends SfDataType{
        String value;

        // default constructor
        public SfDataString()
        {

        }

        // constructor: 1. set value; 2.setVar
         public SfDataString( String input ) {
            this.value = input;
         }

         public void set( String input ) {
            this.value = input;
         }


         public String typeName() {
            return "string";
         }
```

```
// override the copy constructor
public SfDataType copy() {
   return new SfDataString( this.value );
}

// assistance for assign
public static String stringValue( SfDataType b ) {
        if ( b instanceof SfDataString )
       return ((SfDataString)b).value;
   b.error( "cast to String" );
   return null;
}

// assign String = String
public SfDataType assign(SfDataType b)
{
        if(b instanceof SfDataString)
        {
                this.set(stringValue(b));
                return this.copy();
        }
        return error(b , "Error @ assign ");
}

public void print( PrintWriter w ) {
   if ( this.varName != null )
      w.print( this.varName + " = " );
   w.print( this.value );
   w.println();
}

// String + String
public SfDataType plus(SfDataType back)
{
        if ( back instanceof SfDataString )
                return new SfDataString(this.value+ ((SfDataString)back).value );
        return error(back , "Error @ String + String ");
```

```
    }


    // String == String
    public SfDataType eq( SfDataType b ) {
        if ( b instanceof SfDataString )
            return new SfDataBool( value == stringValue(b) );
        return error(b , "Error @ String == String ");
    }

    // String != String
    public SfDataType ne( SfDataType b ) {
        if ( b instanceof SfDataString )
            return new SfDataBool( value != stringValue(b) );
        return error(b , "Error @ String != String ");
    }

/**
 * @param args
 */
public static void main(String[] args) {
        // TODO Auto-generated method stub

}

}
```

SfDataType.java

```
import java.io.PrintWriter;

/**
 * Base class datatype
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v3
 * @since         2005.12.10
 */

public class SfDataType {

   String varName;   // because we use hash table to run it



   public SfDataType()
   {
         varName = null;
   }

   //
   public SfDataType(String varName)
   {
         this.varName = varName;
   }

   public String getVarName()
   {
         return this.varName;
   }
```

```
// copy constructor
public SfDataType copy() {
   return new SfDataType();
}

public void setVarName( String name ) {
   this.varName = name;
}

// because
public String typeName() {
   return "unknown";
}

//---------------------------------------------------------------------------
// all type checking handle here
public SfDataType plus( SfDataType b ) {
   return error( b, "+" );
}

public SfDataType assign( SfDataType b ) {
   return error( b, "assign" );
}

public SfDataType minus( SfDataType b ) {
   return error( b, "-" );
}

public SfDataType time( SfDataType b ) {
   return error( b, "*" );
}

public SfDataType divide( SfDataType b ) {
   return error( b, "/" );
}

// >
```

```
public SfDataType gt( SfDataType b ) {
   return error( b, ">" );
}

// >=
public SfDataType ge( SfDataType b ) {
   return error( b, ">=" );
}

// <
public SfDataType lt( SfDataType b ) {
   return error( b, "<" );
}

// <=
public SfDataType le( SfDataType b ) {
   return error( b, "<=" );
}

// ==
public SfDataType eq( SfDataType b ) {
   return error( b, "==" );
}

// !=
public SfDataType ne( SfDataType b ) {
   return error( b, "!=" );
}

// and            (SfDataBool)
public SfDataType and( SfDataType b ) {
   return error( b, "and" );
}

// or   (SfDataBool)
public SfDataType or( SfDataType b ) {
   return error( b, "or" );
```

```
}



// ---------------------------------------------------------------------------
// to report error msg with the operation to the DataTypes
public SfDataType error( String msg )
{
        throw new SfException( "illegal operation: " + msg
                        + "( <" + this.getVarName() + "> "
                        + ( this.varName != null ? this.varName : "<?>" )
                        + " )" );
}

// to report error msg with the operation to the DataTypes
public SfDataType error( SfDataType b, String msg )
{
        if ( null == b )
                return error( msg );
        throw new SfException(
                        "illegal operation: " + msg
                        + "( <" + this.getVarName() + "> "
                        + ( this.varName != null ? this.varName : "<?>" )
                        + " and "
                        + "<" + this.getVarName() + "> "
                        + ( this.varName != null ? this.varName : "<?>" )
                        + " )" );
}


//  to print the variable name SfData
public void print( PrintWriter w )
{
    if ( this.varName != null )
        w.print( this.varName + " = " );
    w.println( "<undefined>" );
```

```
}

// to print the variable name of SfData
public void print()
{
   print( new PrintWriter( System.out, true ) );
}

// degug use to print the typeName + variableName
public void what( PrintWriter w )
{
   w.print( "<" + this.typeName() + ">  " );
   print( w );
}

// degug use to print the typeName + variableName
public void what()
{
   what( new PrintWriter( System.out, true ) );
}

        /**
         * @param args
         */
        public static void main(String[] args)
        {
                // TODO Auto-generated method stub
                System.out.println("test");
                //SfDataType testData = new SfDataType("a1");

                SfDataString s1 = new SfDataString("s1");
                //testData.what();


                s1.print();
                s1.what();
        }
```

```
}
```

SfDataVariable.java

```
/**
 * SfDataVariable
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v2
 * @since         2005.11.29
 */


import java.io.PrintWriter;

public class SfDataVariable extends SfDataType{

        // variable doesn't have value

        public SfDataVariable( String name )
        {
                super( name );
        }

        // error variable type
        public String typeName() {
           return "undefined-variable";
        }

        // error variable copy
        public SfDataType copy() {
           throw new SfException( " Variable " + this.getVarName() + " has not been defined" );
        }

        // error variable print
   public void print( PrintWriter w ) {
      w.println( this.getVarName() + " = <undefined>" );
   }
```

```
}
```

SfDataVoid.java

```
/**
 * SfDataVoid:   for the return type
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.12.10
 */


// for the return type of functions
public class SfDataVoid extends SfDataType{

        public SfDataVoid( )
        {

        }

        public String typeName() {
            return "void";
        }
}
```

SfException.java

```
/**
 * Exceptions of SFPL to handle
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.10.31
 */


public class SfException extends RuntimeException {

            // handle all exception
            public SfException( String msg ) {
        System.err.println( "Error: " + msg );
    }
}
```

SfFunction.java

```
import java.io.PrintWriter;

import antlr.collections.AST;

/*
 *
 * @author      Huang-Hsu Chen (hc2237@columbia.edu)
 * @version     v1
 * @since       2005.11.30
 * @project     PLT
 */

public class SfFunction extends SfDataType
{

   //String[] args;                    // we need a reference to the AST for the function entry
   //modified by Ling
         SfDataType return_type;
         SfDataType[] args;

   AST body;           // body = null --> built in
                                          // body != null--> user defined functions

   SfSymbolTable pst;   // the parent symbol table
   int id;           // for builtin functions invoke

   // consturctor for user-defined functions
   public SfFunction( SfDataType return_type, String name, SfDataType[] args,
                  AST body, SfSymbolTable pst) {
      super( name );
      // modified by Ling
      this.return_type = return_type;
```

```
    this.args = args;
    this.body = body;
    this.pst = pst;
}

// constructor for built-in functions
public SfFunction( String name, int id ) {
    super( name );
    this.id = id;
    this.args = null;
    pst = null;
    body = null;
}

// to print Sffunction
public void print( PrintWriter w ) {

        // two case:
        // true: Builtin-in functions
        if ( body == null )
    {
        w.println( this.varName + " = <builtin-function> id:" + this.id );
    }
    else
    {
        if ( varName != null )
            w.print( varName + " = " );
        w.print( "<user-function>(" );
        for ( int i=0; ; i++ )
        {
            w.print( args[i] );
            if ( i >= args.length - 1 )
                break;
            w.print( "," );
        }
        w.println( ")" );
    }
```

```
}

public SfFunction() {

    this.args = null;

    pst = null;
    body = null;
}

// ---------------------------------------------------------

public final boolean isInternal() {
    return body == null;
}

public final int getId() {
    return id;
}

// Added by Ling
public SfDataType[] getArgs()
{
        return this.args;
}

// Added by Ling
public AST getBody()
{
        return this.body;
}

// Added by Ling
public SfSymbolTable getParentSymbolTable()
{
        return this.pst;
}
```

```
    // Modified by Ling
    public SfDataType getReturntype()
    {
            return this.return_type;
    }

    public String typeName() {
        return "function";
    }

    public SfDataType copy() {
        return new SfFunction( return_type, varName, args, body, pst );
    }

    // ---------------------------------------------------------
}
```

STL.txt (Standard Library)

```
//Standard Library
//Huang-Hsu Chen        hc2237
//Xiao Song Lu

// wall
function void wall(point:p1, point:p2, color:c)
{
        line(p1, p2,c);
        return;
}

// window
function void window(point:p1, point:p2, color:c)
{
        point: p3;
        point: p4;

        // need four lines with the same color
        p3 = getX(p1)@getY(p2);
        p4 = getX(p2)@getY(p1);


        line(p1, p3, c);
        line(p1, p4, c);
        line(p2, p3, c);
        line(p2, p4, c);

        return;
}

// chage: 8 case -- > 4 case coz we can move p1 and p2 not necessary horizotal or nots
// 4 cases of door
// door
```

```
//redfined
function void door( point:p1, point: p2,int: length, color: c, int: lr, int: tb, boolean:h)
{

        point: p3;
        color: c1;
        c1 = #ffffff;

   if (lr==1 && tb==1 && h==true)          //h1
        {
                p3 = getX(p2) @ (getY(p2) - length);
                line (p1,p3,c);
                line (p1,p2,c1);

        }
   if (lr==1 && tb==0 && h==true)          //h2

   {   p3 = getX(p2) @ (getY(p2) + length);
                line (p1,p3,c);
                line (p1,p2,c1);
        }
        if (lr==0 && tb==1 && h==true)   //h3
        {
                p3 = getX(p1)@ (getY(p1) -length);
                line (p2,p3,c);
                line (p1,p2,c1);
        }
        if (lr==0 && tb==0 && h==true)   //h4
        {
                p3 = getX(p1) @ (getY(p1)+length);
                line (p2,p3,c);
                line (p1,p2,c1);
        }
        if (lr==1 && tb==1 && h==false)  //h1
        {
                p3 = (getX(p1)+length) @ getY(p1);
```

```
                line (p2,p3,c);
                line (p1,p2,c1);
        }
        if (lr==1 && tb==0 && h==false)  //h2
        {

                p3 = (getX(p2)+length) @ getY(p2);
                line (p1,p3,c);
                line (p1,p2,c1);
        }
        if (lr==0 && tb==1 && h==false)  //h3
        {
                p3 = (getX(p1)- length) @ getY(p1);
                line (p2,p3,c);
                line (p1,p2,c1);
        }
        if (lr==0 && tb==0 && h==false)  //h4
        {
                p3 = (getX(p2)- length) @ getY(p2);
                line (p1,p3,c);
                line (p1,p2,c1);

        }

        return;
}

// stair
// change: int direction --> boolean direction
function void stair ( point: p, int: w, int: l, color: c1, color: c2, boolean: direction )
{
        // stair fix into 4 stairs
        point: p1,p2,p3,p4;
        if(direction==true)          //horizontal
        {
                p1 = getX(p)@(getY(p)+l+l );
                p2 = getX(p1)@(getY(p1)+l+l);
```

```
                p3 = getX(p2)@(getY(p2)+l+l);
                rectangle( p,w,l,c1,c2);
                rectangle(p1,w,l,c1,c2);
                rectangle(p2,w,l,c1,c2);
                rectangle(p3,w,l,c1,c2);
                line(p,p3,c1);
                p4 = (getX(p3) + w)@getY(p3) ;
                line((getX(p) + w)@getY(p), p4, c1);

        }
        if(direction==false)        //vertical
        {
                p1 = (getX(p)+w+w)@getY(p);
                p2 = (getX(p1)+w+w)@getY(p1);
                p3 = (getX(p2)+w+w)@getY(p2);

                rectangle(p,w,l,c1,c2);
                rectangle(p1,w,l,c1,c2);
                rectangle(p2,w,l,c1,c2);
                rectangle(p3,w,l,c1,c2);
                line(p,p3,c1);
                p4 = getX(p3)@(getY(p3)+l) ;
                line(getX(p)@(getY(p)+l), p4, c1);

        }
        return;
}

//dimension
// change: int position --> boolean h
function void dimension( point: p1, point: p2, string: s, boolean: h, color: c )
{
        point: lt1,lt2,rt1,rt2, middle;
        int: len;
        lt1 = 0@0;
        lt2 = 0@0;
        rt1 = 0@0;
```

```
rt2 = 0@0;
len = 5; //fixed length of two end line

if(h==true)        //horizontal
{
          lt1 = (getX(p1)) @ (getY(p1)+len);
          lt2 = (getX(p1)) @ (getY(p1)-len);

          rt1 = (getX(p2)) @ (getY(p2)+len);
          rt2 = (getX(p2)) @ (getY(p2)-len);
         middle = ( ( (getX(p1)+getX(p2))/2 )+len*2 ) @( ( (getY(p1)+getY(p2))/2 ) +len*2 );

          // drawLine
          line(p1,p2,c);

          line(lt1,lt2,c);
          line(rt1,rt2,c);

          //
          label(middle, s, c);

}
if(h==false)       //vertical
{
          lt1 = (getX(p1)+len) @ getY(p1);
          lt2 = (getX(p1)-len) @ getY(p1);

          rt1 = (getX(p2)+len) @ getY(p2);
          rt2 = (getX(p2)-len) @ getY(p2);

          middle = (( (getX(p1)+getX(p2))/2 ) ) @ ( ( (getY(p1)+getY(p2))/2 ) - len*2 );
          // drawLine debug
          //print(lt1);
          //print(lt2);

          line(p1,p2,c);
          line(lt1,lt2,c);
```

```
                line(rt1,rt2,c);

                //
                label(middle, s, c);
        }
        return;
}
```

SfSymbolTable.java

```
/**
 * Class SfSymbolTable
 *
 * @author        Ling Zhu (lz2153@columbia.edu)
 */

import java.util.*;
import java.io.PrintWriter;

public class SfSymbolTable extends HashMap
{
        String funcName;
        SfSymbolTable parent;

        public SfSymbolTable(String funcName, SfSymbolTable parent)
        {
                this.funcName = funcName;
                this.parent = parent;
        }

        public final SfSymbolTable getParent()
        {
                return parent;
        }

        public String getFuncName()
        {
                return this.funcName;
        }

        public void setValue(String varName, SfDataType data) throws SfException
        {
                SfSymbolTable st = getRightSymbolTable(varName);

                if(st == null)
```

```
                throw(new SfException("Variable " + varName + " not declared."));
        else
                st.put(varName, data);
    }

    public SfDataType getValue(String varName)
    {
        SfDataType data = null;
        SfSymbolTable st = this.getRightSymbolTable(varName);

        if(st == null)
//              throw(new SfException("Variable not declared."));
                data = null;
        else
                data = (SfDataType)st.get(varName);

        return data;
    }

    public SfSymbolTable getRightSymbolTable(String varName)
    {
        SfSymbolTable st = this;

        while(st != null)
        {
                if(!st.containsVar(varName))
                        st = st.getParent();
                else
                        break;
        }

        return st;
    }

    public void insert(SfDataType var)
    {
        this.put(var.getVarName(), var);
```

```
        }

        public final boolean containsVar(String name)
        {
    return containsKey(name);
  }

  public void what( PrintWriter output )
  {

        for ( Iterator it = values().iterator() ; it.hasNext(); )
    {
       SfDataType d = ((SfDataType)(it.next()));
//        if ( !( d instanceof SfFunction && ((SfFunction)d).isInternal() ) )
//            d.what( output );
       d.what(output);
    }
  }

  public void what() {
    what( new PrintWriter( System.out, true ) );
  }

//        public static void main (String[] args)
//        {
//                SfSymbolTable symb = new SfSymbolTable("main", null);
//                SfDataInt i = new SfDataInt(4);
//                i.setVarName("i");
//                symb.insert(i);
//                SfDataInt x = (SfDataInt)symb.getValue("i");
//                System.out.println("x:" + x.getVarName());
//        }
}
```

Makefile

```
# SFPL's simple makefile
# Huang-Hsu Chen
#
# Before Compile: Set up Classpath for ANTLR
#           1.put the antlr-2.7.5.jar under certain directory
#           2.export CLASSPATH=$CLASSPATH:~/<directorytree>/antlr-
2.7.5.jar"
#
# How to Compile: type "make"

all:
      java antlr.Tool grammar.g
      java antlr.Tool walker.g
      javac -cp . *.java

run: all
      java -cp . Main rooms.txt

clean:
      -rm *.class
```

SfPaintComponent.java

```
import java.awt.Graphics;

/**
 * SfPaintComponent
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */

public class SfPaintComponent {

        public int id;       // to tell what kind of Component
        public String color = "#000000";  // outer color or font color
                                                                        // with default
color black

        // id of paintComponent
        final static int pComponent = 0;
        final static int pLabel = 1;
        final static int pLine = 2;
        final static int pEllipse = 3;
        final static int pRectangle = 4;


        // set id
        public SfPaintComponent(String color)
        {
                this.color = color;
        }

        public void draw(Graphics g)
        {
                System.out.println("It's SfPaintComponent, nothing to draw");
```

```
	}


	// to print different type
	public void print()
	{
		System.out.println("//----------------------------");
		switch(this.id)
		{
		case pComponent:
			System.out.println("SfPaintComponent");
		case pLabel:
			System.out.println("SfPaintLabel");
		case pLine:
			System.out.println("SfPaintLine");
		case pEllipse:
			System.out.println("SfPaintEllipse");
		case pRectangle:
			System.out.println("SfPaintRectangle");

		}
	}
}
```

SfPaintEllipse.java

```
import java.awt.Color;
import java.awt.Graphics;

/**
 * SfPaintEllipse
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */
public class SfPaintEllipse extends SfPaintComponent{

        // user drawOval(int width, int height))
        public int id;

        public int x;       // center point
        public int y;
        public int width; // b/2
        public int height;          // a/2
        public String innerColor;

        public SfPaintEllipse(int x, int y,
                                        int w, int h,String color , String innerColor)
        {

                super(color);
                this.id = pEllipse;
                this.x = x;
                this.y = y;
                this.width = w;
                this.height = h;
                this.innerColor = innerColor;
        }
```

```
	public void print()
	{
		System.out.println("Start point :" +this.x+"@"+this.y);
		System.out.println("width:"+ this.width+", height:"+this.height);
		System.out.println("outer color:" + this.color);
		System.out.println("inner color:" + this.innerColor);
		System.out.println("//-----------------------------");
		super.print();

	}

	// to draw the object on the screen
	public void draw(Graphics g)
	{


		// inner rectangle
		g.setColor(new Color (
				SfPainter.getColor(this.innerColor , 1),
				SfPainter.getColor(this.innerColor , 2) ,
				SfPainter.getColor(this.innerColor , 3) ) );

		g.fillOval(this.x,this.y,this.width, this.height);

//		 outer rectangle
		g.setColor(new Color (
				SfPainter.getColor(this.color , 1),
				SfPainter.getColor(this.color , 2) ,
				SfPainter.getColor(this.color , 3) ) );
		g.drawOval(this.x,this.y,this.width, this.height);

	}
}
```

SfPainter.java

```
/**
 * SfPainter
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */


import java.awt.*;
import java.util.Vector;

import javax.swing.JFrame;



public class SfPainter extends JFrame{

        static Vector vDraw;       // contain all paintComponent to be draw
        boolean isDebug = false;           // debug mode for debuging msg
        int DimensionX = 500;    //Dimension of window
        int DimensionY = 300;    //Dimension of window

        SfPainter(String title)
        {
                this.setTitle(title+"  (window size:"+this.DimensionX+"*" +this.DimensionY +")");
                this.setSize(new Dimension(DimensionX,DimensionY));
                this.setVisible(true);
                this.setBackground(Color.WHITE);

                // test
//              SfPaintLine l1 = new SfPaintLine(2,2,200,200,"#ff22ff");
//              SfPaintLine l2 = new SfPaintLine(1,50,300,200,"#1222ff");
//
```

```
//                  SfPaintRectangle r1 = new SfPaintRectangle(30,20,200, 300, "#000000",
"#1332fa");
//                  vDraw.add(l1);
//                  vDraw.add(l2);
//                  vDraw.add(r1);


                    this.show();
          }

          // to convert Hex string to int ( "FF" --> 255 )
          static public int getColor(String c, int no)
          {
                    c = c.trim();
                    String color = c.substring(1+(no-1)*2,3+ (no-1)*2);

                    //debug
                    //System.out.println("color"+color);
                    return Integer.parseInt( color, 16 /* radix */ );
          }

          public void paint(Graphics g)
          {
                    if(this.isDebug)
                              System.out.println("Start to paint");
                    // to draw all the components in the Vector vDraw
                    for(int i=0; i< this.vDraw.size(); i++)
                    {
                              SfPaintComponent c = (SfPaintComponent)this.vDraw.get(i);
                              c.draw(g);
                              // debug msg
                              if(this.isDebug)
                              {

                              }
                    }
          }
```

```
	/*
	// ----------------------------------------------------------
	// Test version paint
	public void paint(Graphics g)
	{
		// to paint all componet in the vector
		g.drawLine(30,30,100,200);
		g.drawString("test label",200,50);



		g.drawOval(20,20,200,200);
		g.setColor(Color.BLUE);
		g.fillOval(20,20,200,200);

	}
	// ----------------------------------------------------------
	*/


	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

		SfPainter sf = new SfPainter("::: test Painter frame :::");


		sf.repaint();


	}

}
```

SfPainterTest.java

```
/**
 * SfPainterTest
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.12.1
 */


import java.awt.*;
import antlr.collections.impl.Vector;
;

public class SfPainterTest extends Frame{

        public static Vector vDraw;        // contain all paintComponent to be draw
        boolean isDebug = false;           // debug mode for debuging msg

        SfPainterTest(String title)
        {
                this.setTitle(title);
                this.setSize(new Dimension(500,300));
                this.setVisible(true);

                this.vDraw = new Vector();
        }

        // to convert Hex string to int ( "FF" --> 255 )
        static public int getColor(String c, int no)
        {
                c = c.trim();
                String color = c.substring(1+(no-1)*2,3+ (no-1)*2);
                System.out.println("color"+color);
                return Integer.parseInt( color, 16 /* radix */ );
```

```
}

// ----------------------------------------------------------
// Test version paint
public void paint(Graphics g)
{
	// to paint all componet in the vector
	g.drawLine(30,30,100,200);
	g.drawString("test label",200,50);



	g.drawOval(20,20,200,200);
	g.setColor(Color.BLUE);
	String s = "FF";

	// convert a hex String to int
	// Note, there is no lead 0x, case insensitive
	int t = Integer.parseInt( s.trim(), 16 /* radix */ );
	System.out.println("t:"+t);


	System.out.println( SfPainterTest.getColor("#ffffff",3) );
	g.setColor(new Color(0,0,0));
	g.fillOval(20,20,200,200);

}
// ----------------------------------------------------------



/**
 * @param args
 */
public static void main(String[] args) {
	// TODO Auto-generated method stub
```

```
                SfPainterTest testSfP = new SfPainterTest("::: test Painter frame :::");

        }

}
```

SfPaintLabel.java

```
import java.awt.Color;
import java.awt.Graphics;

/**
 * SfPaintComponent
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */

public class SfPaintLabel extends SfPaintComponent{

        public String value;        // the string to be paint
        public int x;
        public int y;

        public int id;

        public SfPaintLabel(String v, int x, int y, String c)
        {
                // give color
                super(c);

                this.value = v;
                this.x =x ;
                this.y =y ;

                this.id = pLabel;
        }

        public void print()
        {
                super.print();
```

```
            System.out.println(this.value + ":" +this.x+"@"+this.y);
            System.out.println("outer:" +this.color);
            System.out.println("//----------------------------");
        }

        // to draw the object on the screen
        public void draw(Graphics g)
        {
            g.setColor(new Color (
                        SfPainter.getColor(this.color , 1),
                        SfPainter.getColor(this.color , 2) ,
                        SfPainter.getColor(this.color , 3) ) );
            g.drawString(this.value, this.x, this.y);
        }

}
```

SfPaintLine.java

```
import java.awt.*;

/**
 * SfPaintComponent
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */
public class SfPaintLine extends SfPaintComponent{

        public int sX;
        public int sY;
        public int eX;
        public int eY;

        public SfPaintLine(int sx, int sy, int ex, int ey , String color)
        {
                super (color);
                this.sX = sx;
                this.sY = sy;
                this.eX = ex;
                this.eY = ey;

                this.id = pLine;
        }

        public void print()
        {
                super.print();
                System.out.println("Start point :" +this.sX+"@"+this.sY);
                System.out.println("End   point :" +this.eX+"@"+this.eY);
                System.out.println("outer:" +this.color);
                System.out.println("//----------------------------");
```

```
        }

        public void draw(Graphics g)
        {
                g.setColor(new Color (
                                SfPainter.getColor(this.color , 1),
                                SfPainter.getColor(this.color , 2) ,
                                SfPainter.getColor(this.color , 3) ) );

                g.drawLine(this.sX, this.sY, this.eX, this.eY);
        }

}
```

SfPaintRectangle.java

```
import java.awt.Color;
import java.awt.Graphics;

/**
 * SfPaintComponent
 *
 * @author        Huang-Hsu Chen (hc2237@columbia.edu)
 * @version v1
 * @since         2005.11.29
 */
public class SfPaintRectangle extends SfPaintComponent{
        //drawRect(int x, int y, int width, int height)
        public int x;
        public int y;
        public int width;
        public int height;

        public String innerColor;

        public SfPaintRectangle(int x, int y, int w, int h, String color
                                                        , String innerColor)
        {
                super(color);

                this.x = x;
                this.y = y;
                this.width = w;
                this.height = h;

                this.id = pRectangle;

                this.innerColor = innerColor;
        }
```

```
	public void print()
	{
		super.print();
		System.out.println("Start point :" +this.x+"@"+this.y);
		System.out.println("width:"+ this.width+", height:"+this.height);
		System.out.println("outer color:" + this.color);
		System.out.println("inner color:" + this.innerColor);
		System.out.println("//-----------------------------");
	}

	// to draw the object on the screen
	public void draw(Graphics g)
	{


		// inner rectangle
		g.setColor(new Color (
				SfPainter.getColor(this.innerColor , 1),
				SfPainter.getColor(this.innerColor , 2) ,
				SfPainter.getColor(this.innerColor , 3) ) );

		g.fillRect(this.x,this.y,this.width, this.height);

//		 outer rectangle
		g.setColor(new Color (
				SfPainter.getColor(this.color , 1),
				SfPainter.getColor(this.color , 2) ,
				SfPainter.getColor(this.color , 3) ) );
		g.drawRect(this.x,this.y,this.width, this.height);

	}


}
```

A.3 Testing Cases

buildin.txt

```
//test all buildin functions
//written by Xiao Song Lu
color: c1,c2;
int: i,j;
point: p1,p2,p3;

c1 = black;
c2 = white;

line (10@40, 200@40, c1);
label (200@40, "line", c1);
rectangle (10@50, 100, 30, c1,c2);
label (25@70, "rectangle",c1);
ellipse ( 10@120, 100,30,c1,c2);
label (35@140, "ellipse", c1);


p1 = 200@10;
p2 = getY(p1)@getX(p1);
p3 = (getX(p2)+190)@getY(p2);
line (p2,p3,c1);
label (p3, "line", c1);
```

dynscop.txt

```
//test dynamic scoping
//writen by Xiao Song Lu
function int f (int:a){

 return x+a;}

function int g (int:b) {
int: x;
x = 1;

return f(b);
}

int: x;

x = 0;

print(g(5));
print (f(5));
```

ifloop.txt

```
//test if and  loop
//written by Xiao Song Lu


int: a,b,c,d;
d=0;

a = 10;

b=5;

c =4;

while (a > 0 )
{ a =a -1;
  while (b >0)
   {if(b <3) {break;}
    b = b -1;
    while (c > 0)
     {if (c <2) {break;}
      d = d +1;
      c =c -1;
     }
    c =4;

   }
    b=5;
  }


print (d);
```

nestedif.txt

```
//test nested if else
//written by Xiao Song Lu

function int f (int:a,int:b)
{

if (b > a)
{ if (b >= 15)
    { if (b >= 20)
        {return b ; }
      else {return 20;}
    }
  else  {return 15;}
}

else
  { if (a >= 16)
    { if (a >= 21)
        {return a; }
      else {return 21;}
    }
  else  {return 16;}
}
}


print(f (22,15));
print(f (17,15));
print(f (15,14));
print(f (10,21));
print(f (10,16));
print(f (10,14));
```

nestedloop.txt

```
//test nested loop
//written by Xiao Song Lu


int: a,b,c,d;
d=0;

a = 10;

b=5;

c =4;

while (a > 0 )
{ a =a -1;
  while (b >0)
   {if(b <3) {break;}
    b = b -1;
    while (c > 0)
     {if (c <2) {break;}
      d = d +1;
      c =c -1;
     }
    c =4;

   }
    b=5;
  }



print (d);
```

rooms.txt

```
//test && demo
//written by Xiao Song Lu

function void wall1 (point: p, int: l) //horizon
{ point:p1;
  p1 = (getX(p)+l)@getY(p);
  line(p,p1, black);
  return;
}

function void wall2 (point: p, int: l)//vertical
{point:p1;
 p1 = getX(p)@(getY(p) + l);

 line(p,p1, black );
  return;
}




function void desk (point:p, int:l, int:w)
{point:p1;


 rectangle(p,l,w,white,brown);
 p1 = (getX(p) + l/3)@ (getY(p) - w/3);
 rectangle(p1,l/3,w/3,white,yellow);
return;
}

function void bed (point: p, int: w, int: l)
{ point:p1,p2;
  color:c;
  c = #B9D3EE;
```

```
  p1 = (getX(p))@ (getY(p) + l/3);

  rectangle (p1,w,l*2/3,white,blue);

  rectangle (p1,w,l,white,blue);

  p2 = (w/4+ getX(p))@(l/12 +  getY(p));
  rectangle(p,w,l/3,white,c);
  rectangle(p2,w/2,l/6,white,white);
  return;
}

function void cabinet (point: p, int: w, int:l)
{ rectangle(p,w,l,white,#bb0000);
  return;
}



function void single (point:p, int: w, int: l)

{ point: p1,p2;

  wall1(p,w);
  wall2(p,l);
  wall1(getX(p)@(getY(p)+l),w);
  wall2((getX(p)+w)@getY(p),l);
  bed(p,w/3,l/2);
  p1 = (getX(p) + w*3/4)@getY(p);
  cabinet(p1, w/4, l/2);
  p1 = getX(p)@(getY(p) + l*4/5);

  desk(p1,w/3,l/5);

  p1 = (getX(p)+w/3)@(getY(p)+3);
  p2 = (getX(p)+w*2/3)@(getY(p)-3);
```

```
  window(p1, p2, black);

  p1 = (getX(p)+w/2)@(getY(p)+l);
  p2 = (getX(p)+w*3/4)@(getY(p)+l);
  door(p1,p2,10,black,1,0,true);
  p1 = (getX(p)+w/2)@(getY(p)+l*2/3);
  label (p1, "Single", black);
  return;
}


point:p,p1,p2,p3,p4;
int : a;
a = 4;
p=50@50;
p1 =p;

while (true)
{single(p,150,200);
 a = a -1;
 if (a==0) {break;}
 p = (getX(p)+152)@50;

}

p2 = getX(p1)@300;


p3 = (getX(p)+152)@50;
p4 = (getX(p)+152)@300;

wall(p1,p2,black);
wall(p2,p4,black);
wall(p1,p3,black);
wall(p4,p3,black);

stair(60@260,5,35,black,white,false);
```

```
dimension(700@50,700@300, "10'",false,black);
dimension(50@325,658@325,"20'",true,black);
```

stdlib.txt

```
//test file for all the standard library;
//written by Xiao Song Lu


point: p1,p2,p3,p4;
color:c1,c2;
c2 = #ffffff;
c1 = #000000;
p1 = 50@50;
p2 = 50@250;
p3 = 400@50;
p4 = 400@250;

wall(p1,p2,c1);
wall(p2,p4,c1);

wall(p3,p1,c1);
wall(p3,p4,c1);


door (100@50,140@50, 10, c1, 1, 1, true);

door (310@50, 350@50, 10, c1, 1, 0, true);


door (310@250, 350@250, 10, c1, 0, 0, true);
door (100@250,140@250, 10, c1, 0, 1, true);

door (50@70,50@110, 10, c1, 1, 1, false);
```

```
door (50@190,50@230, 10, c1, 1, 0, false);
door (400@70,400@110, 10, c1, 0, 1, false);

door (400@190,400@230, 10, c1, 0, 0, false);

stair (100@100,7,40,c1,c2,false);
stair (320@170,40,7,c1,c2,true);

window (397@130, 403@160,c1);
window (47@130, 53@160,c1);

window (185@253, 215@247,c1);

dimension (50@270, 400@270, "length" , true, c1);


dimension (430@50, 430@250, "width" , false, c1);
```

studyroom.txt

```
//test & demo
//written by Xiao Song Lu

function void wall1 (point: p, int: l) //horizon
{ point:p1;
  p1 = (getX(p)+l)@getY(p);
  line(p,p1, black);
  return;
}

function void wall2 (point: p, int: l)//vertical
{point:p1;
 p1 = getX(p)@(getY(p) + l);

 line(p,p1, black );
  return;
}
```

```
//face south direction
function void desk (point:p, int:l, int:w)
{point:p1;


 rectangle(p,l,w,white,brown);
 p1 = (getX(p) + l/3)@ (getY(p) - w/3);
 rectangle(p1,l/3,w/3,white,yellow);
return;
}

function void bed (point: p, int: w, int: l)
{ point:p1,p2;
  color:c;
  c = #B9D3EE;

  p1 = (getX(p))@ (getY(p) + l/3);

  rectangle (p1,w,l*2/3,white,blue);

  rectangle (p1,w,l,white,blue);

  p2 = (w/4+ getX(p))@(l/12 +  getY(p));
  rectangle(p,w,l/3,white,c);
  rectangle(p2,w/2,l/6,white,white);
  return;
}

function void cabinet (point: p, int: w, int:l)
{ rectangle(p,w,l,white,#bb0000);
  return;
}
```

```
function void single (point:p, int: w, int: l)

{ point: p1,p2;

  wall1(p,w);
  wall2(p,l);
  wall1(getX(p)@(getY(p)+l),w);
  wall2((getX(p)+w)@getY(p),l);
  bed(p,w/3,l/2);
  p1 = (getX(p) + w*3/4)@getY(p);
  cabinet(p1, w/4, l/2);
  p1 = getX(p)@(getY(p) + l*4/5);

  desk(p1,w/3,l/5);

  p1 = (getX(p)+w/3)@(getY(p)+3);
  p2 = (getX(p)+w*2/3)@(getY(p)-3);
  window(p1, p2, black);

  p1 = (getX(p)+w/2)@(getY(p)+l);
  p2 = (getX(p)+w*3/4)@(getY(p)+l);
  door(p1,p2,10,black,1,0,true);
  p1 = (getX(p)+w/2)@(getY(p)+l*2/3);
  label (p1, "Single", black);
  return;
}

single(50@50,150,200);
dimension(225@50,225@250,"7'",false,black);
dimension(50@275,200@275,"5'",true,black);
```

testbool.txt

```
//test case for type boolean
//written by Xiao Song Lu
function boolean f(boolean:a, boolean:b) {

boolean: c;
c= a&&b;
return c;
}

int: a,b;
boolean:bool,b1,b2;


a=2;
b=10;
b1 = true;
b2 = false;
print(b1);
print(b2);
bool = b1&&b2;
print(bool);
bool = b1||b2;
print(bool);
bool = f(b1,b2);
print(bool);
print (f(true,false));
bool = b1!=b2;
print(bool);
bool = b1==b2;
print(bool);
bool = a+b==b+a;
print(bool);
bool = a+b!=b+a;
print(bool);
```

```
bool = a > b;
print(bool);
bool = a< b;
print(bool);
bool = a+b > a-b;
print(bool);
bool = a+b < a-b;
print(bool);
bool = a*b-a < a*b-b;
print(bool);
bool = a*b-a > a*b-b;
print(bool);

bool = a*b-a > a*b-b || a*b-a < a*b-b;
print(bool);
bool = a*b-a > a*b-b && a*b-a < a*b-b;
print(bool);
bool = a*b-a > a*b-b && a*b-a < a*b-b && a+b < a-b;
print(bool);

bool = a*b-a > a*b-b || a*b-a < a*b-b || a+b < a-b;
print(bool);
bool = a*b-a > a*b-b || a*b-a < a*b-b && a+b < a-b;
print(bool);
```

testint.txt

```
//test case for type int
//written by Xiao Song Lu
function int sum (int:a, int: b)
{ int:c;
  c = a +b;
return c;
}

int: a,b,c,d;
boolean: bool;
a = 2;
b = 10;

print(a);
print(b);
c = a +b;
print(c);
c= a -b;
print(c);
c = a*b;
print(c);
c= b/a;
print(c);
c= a*b + b/a;
print(c);
c= (a+b)*(b-a);
print(c);
c= sum(a,b);
print (c);

bool = a+b==b+a;
print(bool);
bool = a > b;
print(bool);
```

```
bool = a< b;
print(bool);
bool = a >= b;
print(bool);
bool = a <= b;
print(bool);
bool = a >= a;
print(bool);
bool = a <= a;
print(bool);
bool = a+b > a-b;
print(bool);
bool = a+b < a-b;
print(bool);
bool = a*b-a < a*b-b;
print(bool);
bool = a*b-a > a*b-b;
print(bool);
bool = a+b >= a-b;
print(bool);
bool = a+b <= a-b;
print(bool);
bool = a*b-a <= a*b-b;
print(bool);
bool = a*b-a >= a*b-b;
print(bool);
if (b>a) {
 print (a);
}
else {
print(b);
}
d = 5;
c = 0;
while ( d > 0)
{c = c + b;
 d = d -1;}
```

```
print(c);
```

testpoint.txt

```
//test case for type point
//written by Xiao Song Lu
function point f (int: a,int: b)
{ point: p;
  p = a@b;
  return p;
}

point: p1,p2,p3;
boolean: bool;

p1 = 1@2;

p2 = 2@3;

p3 = p1 +p2;

print (p3);

p1 = (1 + 5 *9) @ (4-10/5);

print (p1);

p3 = f(1,2);
print (p3);
```

testrecursion.txt

```
//a factorial function use to test recursion
//written by Xiao Song Lu

function int f (int:a){

if (a == 1) { return 1;}
else{ return a*f(a-1);}
}
int:x;

x = f(5);
print (x);
```

teststring.txt

```
//test case for type string
//written by Xiao Song Lu
function string f (string: a)
{ string:c;
 c = a;

 return c;}

string:a,b,c;
boolean: bool;

a = "fool";
b = "bar";
c = "fool";

print(a,b,c);
//bool = a==b;
//print (bool);

//bool = a!=b;

//print (bool);

a=f(a);
print( a);
print( "fool");
```