Floor Plan Language (FPL)
Final Report

Xinwei Zhang (xz2663) Manager
Chih-Hung Lu (cl3519) Language Guru
Dongdong She (ds3619) System Architect
Yipeng Zhou (yz3169) Tester

I. Introduction

Nowadays, AutoCAD has been a dominated professional software to draw house floor plans. However, AutoCAD is neither easy nor free to use. For someone who wants to decorate the house by himself, it takes hundreds of dollars to buy AutoCAD and dozens of hours to watch AutoCAD tutorials. To avoid this painful experience, our group intends to develop a user-friendly programming language called Floor Plan Language (FPL), which is specifically designed for drawing floor plans.

As we doing research on how AutoCAD works, we find that the basic idea is to build the framework of the floor plan, which is the wall. So, we design our language to be a C-like language. Users can easily and precisely define the position, size, and color of all these elements of the floor plan with FPL. With built-in control flow such as "for", and "if-else", users can easily generate dozens of furniture without repetitive operations. For professional floor plan designers, FPL can be used to generate hundreds of floor plans with just a few code modification. In the following sample code, we show a simple example of how FPL works.

From the high-level view, we will implement a library fplComponent to render all types of our components like a wall, chair, desk,...etc. In the library fplComponent, we will invoke the APIs from OpenGL to do rendering job. Moreover, our FPL compiler will parse our FPL and generate the corresponding object file, which contains function call invoking APIs provided by fplComponent. At the final stage, all the object file (*.o), fplComponent (*.so/*.a), and OpenGL (*.so/*.a) will be linked together and compiled as an executable file FPLRender. Users can execute this FPLRender to render their floor plan graph.

II. Language Tutorial

A. Introduction

FPL is a C-like language. It is intended to draw a floor plan graph using a third-party package OpenGL. It provides some primitive built-in data type for presenting some basic components of a floor plan. For example, the data type “chair” is to present a chair graph, which will appear on our generated floor plan. There are three main steps

to draw a graph via FPL. First, users need to declare some FPL objects with built-in types like “wall” or “bed”. Second, users need to invoke built-in function “put“ with two float-type parameters “x”, and “y”, which means that put the object on the specific position (x, y). Finally, users need to invoke another built-in function “render” to translate all the FPL objects into a complete floor plan graph.

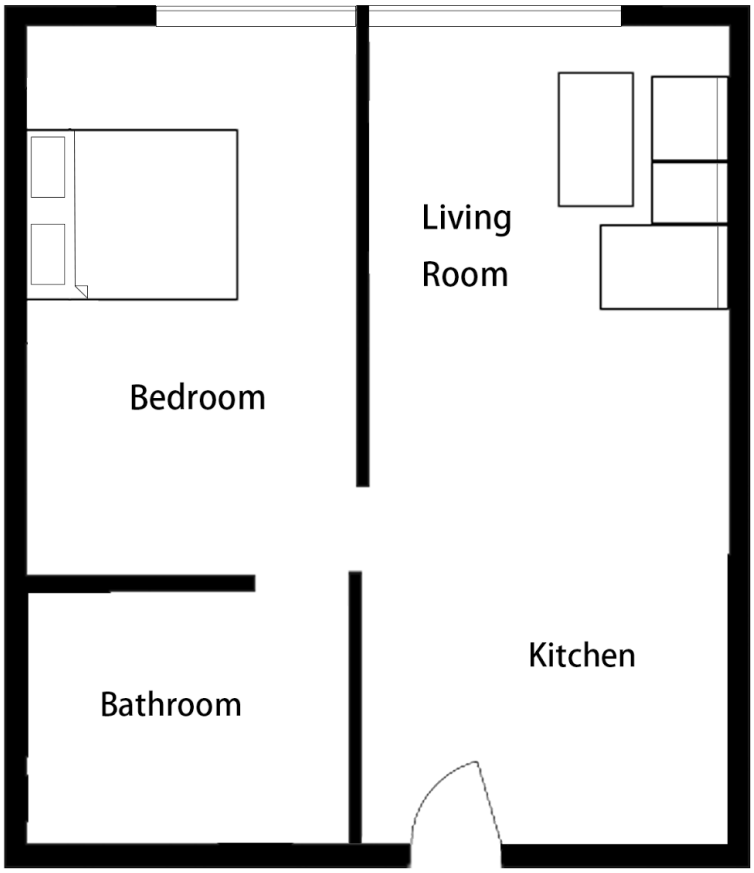
B. Sample Code

This section present a sample code to demonstrate a 1b1b apartment. And the corresponding rendered graph is on the next section.

```
/* user defined struct */
struct Sofa{
   chair c0;
   chair c1;
   chair c2;
}

/* used defined function */
void livingRoomMake(){
    /* invoke user-defined struct */
    Sofa s;
    s.c0 = chair(1, 1);
    s.c1 = chair(2, 1);
    s.c2 = chair(3, 1);
    rotate(s, 90);
    put(9, 15);

    desk d = desk(2, 6);
    put(desk, 6, 12);
}

int main(){
    wall wall = wall(10, 1);
    /* top wall */
    put(wall, 0, 19);

    /* bottom wall */
    put(wall, 0, 0);
```

```
    wall sideWall = wall(1, 20);
    /* left wall */
    put(sideWall, 0, 0);

    /* right wall */
    put(sideWall, 19, 0);
    wall middleWall = wall(1, 8);
    /* middle wall0 */
    put(middleWall, 5, 11);

    /* middle wall1 */
    put(middleWall, 5, 0);

    /* invoke user-defined function to build the living room */
    livingRoomMake();

    door door = door(3, 1);
    put(door, 12, 0);

    bed bed = Bed(3, 3);
    put(bed, 0, 12);

    window window = window(0, 0, 3, 1);
    put(window, 2, 19);
    put(window, 6, 19);
    render();
    return 0;
}
```

C. Sample Graph



III. Language Manual

A. Lexical Elements

1. Identifiers

Identifiers are strings used for naming different elements, such as variables, functions, and the words "if" in control flow, etc. The identifiers are consist of letters, digits, and underscore ‘_’, and the letters are case sensitive. Each identifier should always start with a letter. These rules are described by the definitions involving regular expressions below:

identifier := (letter) (letter | digit | underscore)*
digit := '0'-'9'
letter := ('A'-'Z') | ('a'-'z')
underscore := '_'

2. Keywords

int	double	string	char	bool	if

else	for	main	void	null	put
rotate	struct	true	false	put	rotate
wall	bed	desk	door	window	rectangle
circle	return	while	break	continue	

3. Literals

\”	Insert a double in the string
\\	Insert a backslash in the string
\n	Insert a newline in the string
\t	Insert a tab in the string
true	Boolean true value
false	Boolean false value

4. Delimiters

Parentheses ()	Enclose arguments for a function
Brackets {}	Array initialization and assignment
Commas ,	Separate different function components
Semicolon ;	Terminate a sequence of code
Curly Braces{...}	Enclose function/struct definitions and code in if statements/ for loops
Periods .	Access fields of an object
Whitespace	Separate tokens. Include spaces, tabs, newlines

B. Data types

PLT maintains primitive data types like int, bool for general computation and other built-in data types like Wall, Bed, Desk to represent some basic element in a floor plan graphic. User can also define their own structure in PLT like C to satisfy their specific needs.

1. Primitive data types

int	a 32-bit unsigned integer
char	a single ASCII character
bool	a boolean variable (True and False)
string	a null-terminated sequence of characters
float	floating-point type, usually referred to as a single-precision floating-point type

2. Built-in data types

wall	a data structure to demonstrate a wall on floor plan, take two diagonal coordinates as parameters, e.g wall(x1, y1, x2, y2)
bed	a data structure to demonstrate a bed on floor plan, take two diagonal coordinates as parameters, e.g bed(x1, y1, x2, y2)
desk	a data structure to demonstrate a desk on floor plan, take two diagonal coordinates as parameters, e.g desk(x1, y1, x2, y2)
door	a data structure to demonstrate a door on floor plan, take two diagonal coordinates as parameters, e.g door(x1, y1, x2, y2)
window	a data structure to demonstrate a window on floor plan, take two diagonal coordinates as parameters, e.g window(x1, y1, x2, y2)
rectangle	a data structure to demonstrate a basic rectangle on floor plan, take two diagonal coordinates as parameters, e.g rectangle(x1, y1, x2, y2)
circle	a data structure to demonstrate a basic circle on floor plan, take one coordinate as center and a positive integer as radius.

3. User-defined data types

User can define their own data structure to generate special element in data floor such as a fancy sofa, a square toilet. Here is an example,

```
Struct SquareToilet{
        Rectangle seat = Rectangle(0, 0, 3, 3)
        Rectangle cistern = Rectangle(0, 3, 3, 4)
}
```

C. Expression and Operators

1. Expressions

Expressions are made of at least one operand and zero or more operators. Innermost expressions are evaluated first and the priority of an expression is determined by parentheses. The direction of evaluation is from left to right.

2. Operators

Operator	Description	Associativity
!	Logical Not	Right to Left
=	Assignment	
* / %	Multiplication, Division, Remainder	Left to Right
+ -	Addition, Subtraction	
==	Equality	
!=	Not equal	
>	Greater than	
<	Less than	
>=	Greater than or equal to	
<=	Less than or equal to	
&&	Logical AND	
||	Logical OR	

Punctuation	Purpose
;	Used to end a statement
{ }	Used to enclose functions, while and for loops, and if statements. In other words, they are used to delineate the scope of blocks of code in the program.
( )	Used to specify and pass arguments for a function and the

	precedence of operators. Also used to enclose conditions in for and while loops and if statements.
,	Used to separate function arguments
“ ”	Used to declare a variable of string data type
/* */	Block comment

D. Control Flow

if (*expression*) {
statement
}
	The expression is evaluated and if it is non-zero, the statement is executed

if (*expression*) {
statement
} else {
statement
}
	Second substatement is executed if the expression is 0. As usual the ‘‘else’’ ambiguity is resolved by connecting an else with the last encountered elseless if.

while (*expression*) {
statement
}
	The substatement is executed repeatedly so long as the value of the expression remains non-zero. The test takes place before each execution of the statement.

for (*expression*; *expression*; *expression*) {
statement
}
	The first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression becomes 0; the third expression typically specifies an incrementation which is performed after each iteration.

return;	No value is returned.
return *expression*;	The value of the expression is returned to the caller of the function.

E. Functions

1. Function Definition

In FPL, the definition of a function consists of a keyword “fun”, a return type, a function identifier and some parameters and their types. Then, there is a block of code enclosed by curly braces. An example of a function definition is like this: int multiply(int a, int b){return a*b;}

2. Function Call

A function call in FPL is the function's identifier followed by its params enclosed by parentheses. An example of a function call is like this: multiply(2,5);

F. Built-in Functions

API		
Name	Function expression	Description
put	put(object, x, y)	Render the object to the specific position
rotate	rotate(object, direction)	Rotate the object with the specific direction
render	render()	A command to notify OpenGL to render the whole floor plan

IV. Project Plan

We are lucky to take in advices from students who took PLT before to start as early as possible. With careful planning and organization, we divide the whole project to four part, core logic of FPL, native graph library, semantic check and testing. Each team member is assigned one part and work simultaneously, then we would merge together.

A. Project process

1. Planning

In order to synchronize every team member's update, we meet consistently twice every week on Wednesday and Sunday. During the regular meeting, every team member would share his progress, knowledge and experience learned from his work to make sure everyone is aware of technical detail of the whole project. In the end, we would make goal for next week to ensure we meet each mile stone in the plan.

2. Specification

In the development of FPL, we strictly follow the LRM we made at the beginning of the project. Thanks to the careful planning of LRM at the beginning, we manage to achieve every functions claimed in the original LRM.

3. Development

We divide the whole project into three step. First we tried to finish a rough version FPL only with built-in function and built-in data types, second we tried to add user-defined struct to it. Finally, we integrate graph library to draw a real floor plan graph.

4. Testing

At a high level, we set a specific goal for each stage of our project. For example, for stage 1, we test FPL to handle simple primitive data types and built-in functions; for stage 2, we ensure FPL could handle user-defined struct while maintain functionality of stage 1; for stage 3, we test FPL to draw a real floor plan with graph library integrated.

B. Style Guide

We used the following rules when writing our code to ensure maximum readability:

- Each line of code should remain under 100 characters
- Use block comments for each large section of code
- Write utility functions for commonly reused code
- Use underscore separated function names and capitalized type names

C. Team Responsibilities

Core logic of FPL: Chih-Hung Lu
Graph library: Dongdong She
Semantic check: Yipeng Zhou
Testing: Xinwei Zhang

D. Project Timeline

Sept 26th: Proposal due date
Oct 11th: LRM due date
Oct 14th: Finish stage 1
Oct 28th: Finish stage 2
Nov 11th: Finish testing graph library
Nov 25th: Finish stage 3
Dec 19th: Final project report
Dec 20th: Final project due date

E. Development environment

The FPL team developed on a variety of environments including mac OS X, Debian, Arch Linux. We use OCaml, OCaml and OCamlyacc for the compiler itself. The graph library is developed in C with OpenGL API. We host our code on Github for version control. We used bash scripts and makefiles to ease the work of compiling and testing the code.

F. Project Log

```
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Mon Dec 18 09:34:23 2017 -0500

    Comment the unused case

commit 68326190f9768d7792dfd0f0871e865c714c8713
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Mon Dec 18 09:07:58 2017 -0500

    Comment debug functions

commit f7527b2a9768be82c5ba5f870a9cd903f7f0896d
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date:   Sun Dec 17 22:34:09 2017 -0500

    Semantic Check for struct and test cases

    Remove some warnings.

commit 21e0f7c182993dcbd18d877f8de80a5b89800e77
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Dec 17 15:48:25 2017 -0500

    Optimize demo testing case

commit dc16f0f002ce544f0a3091e54358f8048a3d6456
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Dec 17 13:28:47 2017 -0500

    Fix naming bug

commit 9628ee530c1daf0db54234f6c95a78e6ae63b14a
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Dec 17 13:08:16 2017 -0500

    Remove unused file and code

commit 42f1b97c2696c09198a81d1ef53981ecf3ae05ae
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Dec 17 12:32:16 2017 -0500

    Refine testing case for demo

commit 56043d69022c0896eaa20f313a24d87646170835
Author: Dongdongshe <shrek.s@hotmail.com>
```

```
Date:   Sat Dec 16 19:44:49 2017 -0500

    add dummy circle function.

commit 5b40e22b70945a0e1c26c2671770b999a0689910
Author: Xinwei <zxw018018@hotmail.com>
Date:   Sat Dec 16 18:29:48 2017 -0500

    test demo

commit c0639ee11aa0da61f11fec43f9ccda3180837368
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date:   Wed Dec 13 17:55:30 2017 -0500

    Test for OSX and Linux

commit 111ba11dd0329cdb8f9926a3a9cc7480408fbb67
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Wed Dec 13 15:48:59 2017 -0500

    Fix drawing for rectangle

commit 7d3b587db5602b2a45c232d7bfbcc75c1815f391
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Wed Dec 13 11:43:53 2017 -0500

    Integrate openGL to testing script

commit 09352f1874866f90976776bdcf73e9492ff63779
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Tue Dec 12 18:51:44 2017 -0500

    Update README.md

    add math library when compiling

commit ce4552309aa6682a43902164dc916849af083888
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Tue Dec 12 18:49:42 2017 -0500

    add spiritual drawing using openGL

commit fa32d5df73217473188bde949a5d04e09151f3a8
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Thu Dec 7 13:32:31 2017 -0500

    Fix the scope of fplObject and struct map
```

commit ab4c479d66919b97e039bf625c8a85f8ea79c321
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date: Mon Dec 4 23:20:11 2017 -0500

    Remove unused code

commit c99fdc3ca2c40546be7757b2823eefef8dc053dc
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date: Mon Dec 4 23:05:31 2017 -0500

    Support user-defined struct

commit 03e4a24ce08fc039f368f7e047eb4c1e6e4cb88f
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date: Sun Dec 3 18:34:24 2017 -0500

    Add parser for user-defined struct

commit 5a80764c323b79fc43d9827390d907d2dcee0981
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date: Sun Dec 3 17:42:33 2017 -0500

    Add parser for user defined struct

commit 1ec69590f452661038a89115d86e5460e1b2ca58
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date: Wed Nov 29 17:33:55 2017 -0500

    Check for type of numbers

commit c2f69ad9e3b07b0360ee5c83dcf6d33e5da14284
Author: Xinwei <zxw018018@hotmail.com>
Date: Wed Nov 29 15:20:03 2017 -0500

 check name

    check name of every construction

commit 9518754db2148baff7f83f8813c0f62ca8a14f53
Author: Xinwei <zxw018018@hotmail.com>
Date: Mon Nov 27 23:30:35 2017 -0500

    actuals

    Check whether the inputs of wall, bed, desk, door, window, rectangle and circle are numbers or not

```
commit 4a5fbb9d9bbf6f9cdb4d04fc362277009416866a
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Mon Nov 27 15:22:20 2017 -0500

    add render() and put_wall() put_bed()

commit 0dc95738346f216b7414b90c5dca442a4088f44d
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date:   Sun Nov 26 23:15:56 2017 -0500

    Link OpenGL in Mac OS

commit 685d7694ea5cf981aa952e24d566cc21f23c8f0e
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Fri Nov 24 15:34:59 2017 -0500

    Add build-in render function

commit bca9fdd510399a219571ea4c8554bc34debf7398
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Fri Nov 24 15:16:09 2017 -0500

    1. Change parameter type of fpl Object to float; 2.Add desk/door/window/rectangle/circle type

commit 04b1e4ceba53e91c98f143e78e5d9ee31fa14cb7
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Nov 19 22:21:19 2017 -0500

    Add build-in function rotate

commit 776b30995ee822f4500c6eea3151dd9518c2e2b2
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Nov 19 13:18:28 2017 -0500

    Refine code base structure

  commit f4ed09af779ec765bedb7b99d183f9d16badcac7
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Sun Nov 19 10:52:44 2017 -0500

    Add wall/bed type, and their associated put function

commit 20cbe0ecb9b061a5544a29671cfd31ce70b15847
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Thu Nov 16 22:26:37 2017 -0500

    1. add built-in data type Wall.
```

```
    2. add built-in access expr.
    3. add test-wall.mc

commit 328b88d4da59b7648e414b58c3ad06edd0ba5ea8
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date:   Wed Nov 15 14:00:33 2017 -0500

    Add String

commit 2f631b7876c4a4957ac8e23775740b306b9072d3
Author: Yipeng Zhou <zyp499131910@gmail.com>
Date:   Wed Nov 15 12:49:25 2017 -0500

    Add Char

commit b4f06e556805d5d7393c0b5e19066d33a95a9330
Author: Xinwei <zxw018018@hotmail.com>
Date:   Wed Nov 15 12:12:32 2017 -0500

    Add error message

    Add error message during float operations

commit 9b0059f0d63296e7a55c3ed3bb30963e129a3549
Author: Xinwei <zxw018018@hotmail.com>
Date:   Wed Nov 15 12:04:01 2017 -0500

    float test

commit e31234cec10a4c142cf97c76c919e4b548d3ad00
Author: Xinwei <zxw018018@hotmail.com>
Date:   Wed Nov 15 12:02:47 2017 -0500

    Add float operations

 commit afc2d07b1a6be62530ea43003aa89152fa38613b
Author: Xinwei <zxw018018@hotmail.com>
Date:   Tue Nov 14 23:44:06 2017 -0500

    Add float

commit ed13710e4a9aaf4c8db33f149d99306fdbc5f2c6
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Tue Nov 14 21:57:20 2017 -0500

    Add printChar()
```

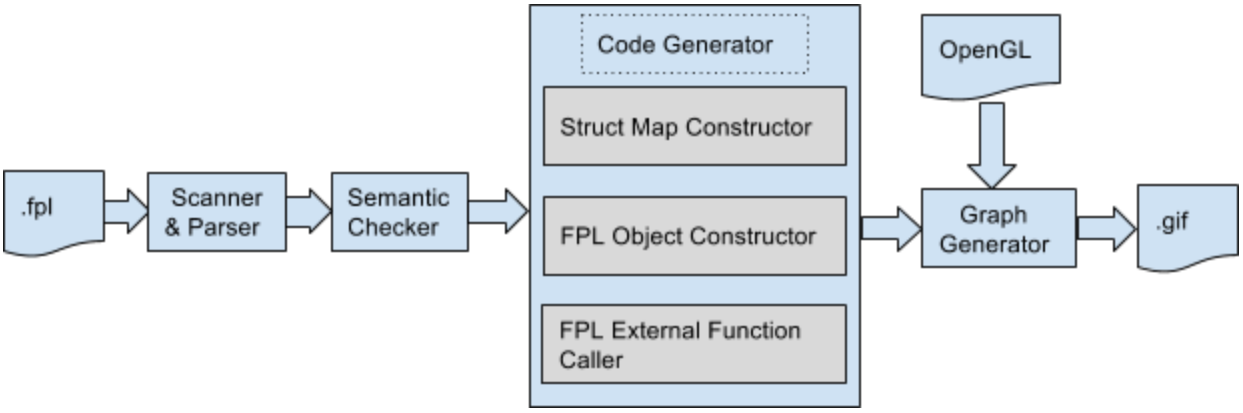
```
commit 7ddca116f9fdae7674a1a2411bf53d21d81dfdad
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Sat Nov 11 17:45:58 2017 -0500

    Update README.md

commit 2ce48726510e009879c47f238e59a7aa33200b8c
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Sat Nov 11 17:45:33 2017 -0500

    Update README.md

commit f4d9f5099f2ccaf7e246ca1cc0a974d48a3f4b32
Author: Dongdongshe <shrek.s@hotmail.com>
Date:   Sat Nov 11 17:37:26 2017 -0500

    Add built-in function drawLine() and drawRec() to draw line and rectangle using OpenGL

commit 8ef47a5fd67470d28f58d9c3a84eadafe0258dec
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Wed Nov 8 10:05:50 2017 -0500

    Update README

commit 75132452087f387c0b5ae5f5eebfcfd6c2ede2b5
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Wed Nov 8 00:01:42 2017 -0500

    Add putc for Hello World demo

commit 30efcb276d23a926b64cbbda753764b15c2ce1fb
Author: Namo.Lu <chih-hung.lu@columbia.edu>
Date:   Fri Nov 3 10:26:31 2017 -0400

    Initila commit

commit 042e422258f34bdeaa921a0095a3e035b0fdd20e
Author: Chih-Hung Lu (Namo) <mud2man@gmail.com>
Date:   Fri Nov 3 10:23:01 2017 -0400

    Initial commit
```

V. Architectural Design

A. Diagram

The following graph is the block diagram showing the major components of FPL translator. The input file is an FPL language source code. After it goes through the AST

parser and semantic checker. It is guaranteed no syntax error, and the abstract syntax tree is built up. Then the symbol table, the list of the function bodies, the list of global variables, and the list of local variables will be feed into the code generator. There are three main components in this part, struct map constructor, FPL object constructor, and external function caller. The following section will explain more details. Then, the assembly code is generated by the code generator, and it will invoke the C functions implemented in graph generator. Finally, the floor plan graph will be generated by the graph generator in the last stage.



B. Components

1. Scanner and Parser

The scanner and parser of FPL are built on the top of MicroC language. The main feature of this component is to parse our language, and build a abstract syntax tree in Ocaml data structure. Besides the original language syntax of MicroC, we added some built-in data type like “wall”, “bed”, “chair”, ...etc. They are represented as different graph components in our generated floor plan. Also, we can support user-defined struct. For, example, users can declare a struct named “sofa”, and put three members with three chair-type variables in the struct. This feature can make our language more compact and modularized.

2. Semantic checker

In semantic checking, the compiler get the symbol tables from the result of AST. The tables are consist of functions, globals and structs. In the table “functions”, there are subtable function.formals and function.locals.

In these tables, we implemented all the semantic check of MicroC. Besides that, we design some new features for our language:

a) Built-in Type Checker

Our semantic checker verifies the type name and arguments. If the number or type of arguments wrong, it will print out the failure message and show the reason. In this example, the semantic checker could find the second line of d definition has a wrong number of arguments. It can also find the definition of e used a wrong type.

```
int main()
{
  window d;
  door e;
  d = window(0.7, 0.8);
  d = window(0.8);
  e = desk(0.9, 1.0);
  put(d, 5.0, 5.0);
}
```

b) Struct Checker

From table program.struct, the semantic checker verified the element name and type for every struct. In this example, it could find the wrong type definition “desk” on b.f where should be “rectangle”. The semantic checker could check the wrong element call for a struct, too. The definition on b.t is illegal because we can’t find t in b. Also we can check duplicate struct if we define another struct whose name is as same as “Bedroom”.

```
struct Bedroom {
    window e;
    rectangle f;
    circle g;
};

int main()
{
  Bedroom b;
  door e;
  b.e = window(0.7, 0.8);
  b.f = desk(0.9, 1.0);
  b.t = 1;
}
```

3. Code Generator

Code generator is the core part in FPL compiler. It was made up of three components: struct map constructor, FPL object constructor, and external functionaller caller.

a) Struct Map Constructor

In order to handle user-defined struct, FPL compiler has a map to keep the relationship between struct and its members. We define the data structure as “struct map”. The map is limit its scope to a single function. To be more specific, there are two struct maps if there are two function body in a single .fpl source code. Take the following sample code as an example.

```
struct Bedroom {
    window e;
    rectangle f;
    circle g;
};

void foo(){
  Bedroom r;
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.1, 1.2, 1.0);
  put(r, 5.0, 5.0);
}

int main()
{
  Bedroom r;
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.1, 1.2, 1.0);
  put(r, 5.0, 5.0);
    .
    .
}
```

There are two different struct maps. One of them is visible from the function “main”, the details are on the following table “Struct Map @ main”. And the other one is visible from the function “foo”, the details in on the table “Struct Map @ foo”. With the information, FPL compiler can loop up the name of members of the targeted struct, and do another look up from the map “FPL object map”, which will be introduced on the next section. So our FPL operation like “put” and “rotate” can accept a user-defined struct as a parameter by the means of looking up.

Struct Map @ main	
r	r.e
	r.f
	r.g

Struct Map @ foo	
r	r.e
	r.f
	r.g

b) FPL Object Constructor

First, we need to define FPL object. An FPL object is a component with primitive graph type, which includes “wall”, “bed”, “door”, “desk”, “window”, “rectangle” and ”circle”. And every FPL object can be identified by a 6-tuple, which is (type, degree, x coordinate of region, y coordinate of region, x coordinate of position, y coordinate of position). The following table details all the fields in the 6-tuple.

6-tuple	
type	0: wall
	1: bed
	2: desk
	3: door
	4: window
	5: rectangle
	6: circle
degree	The angle of rotation of this object, which is between from 0 and 359
x coordinate of region	The x coordinate of the top-right corner of draw region

y coordinate of region	The y coordinate of the top-right corner of draw region
x coordinate of position	The x coordinate of the position
y coordinate of position	The y coordinate of the position

c) FPL Function Caller

FPL compiler has three built-in graph-related functions “put”, “rotate”, and render. Let’s illustrate how “put” works in FPL compiler. When the function “put” appear in the .fpl source code, our FPL compiler will link it to the outer c function “put” which located “fplFunctions.c”. The outer c function “put” will do a dispatch job based on the type field of the given 6-tuple. Take the following sample code an example, the first step of FPL compiler does when “put(r, 5.0, 5.0)” is scanned is to look up the struct map. And compiler will learn that there are three tuples “r.e”, “r.f” and “r.g”. Then, the compiler will invoke the c function “put” three times with the individual 6-tuples, which is gained by looking up the FPL object map.

```
int main()
{
  Bedroom r;
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.1, 1.2, 1.0);
  rotate(r, 90);
  put(r, 5.0, 5.0);
}
```

For the built-in function “rotate”, FPL compiler will look up the struct map and FPL object map to identify the associated tuple. And modify the “degree” field in the tuple. For the built-in function “render”, FPL compiler will call the outer c function “render” to notify OpenGL to render the whole floor plan.

4. Graph Generator

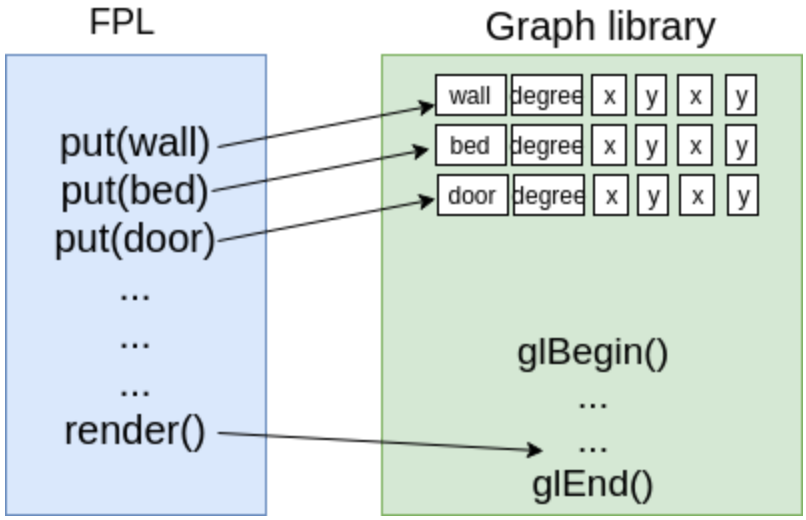
Graph generator is a native graph library based on OpenGL in C. To simplify the our FPL design, we put most of heavy work to native graph library. The native graph library will take in parameters passed from FPL built-in functions such as put(), render() and perform corresponding graphic task using OpenGL API. A floor plan consists of multiple basic graphs such as a rectangle representing a bed, a line representing a wall and a circle sector as a door. Users can first our FPL to define a basic graph and size

of it, then call built-in function put() to draw it to any location he or she wants. Finally, users call render() to generate the floor plan graph.

a) Graph library design

OpenGL is a rather ancient and complicated drawing API in C code. Although there are many other modern libraries provide more powerful and user-friendly wrapper for OpenGL drawing, most of them are implemented in C++ and would cause linking issue because of C++ mangle issue. So we have to use almost deprecated ancient OpenGL API in pure C to implement all the drawing tasks. The good news is that, a floor plan graph is fairly simple graph composed of several basic graphs like rectangle, line and circle. We manage to use original OpenGL api to draw a complete floor plan graph.

In our FPL, user need to draw each basic elements of a floor plan, then compose them into a final graph. However, OpenGL requires a fairly complicated order of API usage to render a graph. To solve this problem, we design a special parameters handling to store all the basic elements to be drawn locally, and only call OpenGL drawing API in the last step to render the final graph.



b) Parameters handling

When users use our FPL to draw a floor plan, he or she need to first draw some basic element with built-in put(), then compose them into a final graph with built-in render(). OpenGL has rather complicated drawing procedure, it requires correct order of environment initialization, coordinates based parameters feeding and final rendering. To simplify the design, we store all the parameters passed from built-in put() into a static global array. During the final call to render(), graph library would read every elements stored in static global array, perform corresponding drawing procedure and generate the final graph.

c) Drawing implementation

OpenGL supports basic graph drawing such as rectangle and line by using the following snippet code

```
glBegin(GL_LINE);                 //starts drawing of line
glVertex2f(1.0f,1.0f);            //use coordinate to define right point
glVertex2f(-1.0f,-1.0f);          //use coordinate to define left corner
glEnd();                          //end drawing of line
```

As for special graph like a circle, we implement a approximation of circle using large number polygon

```
glBegin(GL_LINE_GROUP);    //draw a polygon of 600 angle to approximate a circle
for(ii = 0; ii < 600; ii++)
 {
    float theta =  2 * 3.1415926f * ii / 600;        //get the current angle
    float x = r * cosf(theta);                       //calculate the x component
    float y = r * sinf(theta);                       //calculate the y component
    glVertex2f(x , y);                               //output vertex
}
glEnd();                          //End quadrilateral coordinates
```

C. Who did what?

Namo focused on inventing the logic concept of FPL objects, implemented it and the associated operations in scanner, parser, and code generator, then designed some testing patterns for FPL objects. Yipeng implemented the associated operation on the new data type “string” and “char”, and responsible for the semantic check. Xinwei implemented the associated operations on the new data type “float” and was responsible for the test plan. Dongdong focused on identifying OpenGL API, figured out how to set up the environment for running OpenGL, and implemented the graph drawing for every FPL object type.

VI. Test Plan

A. Representative source language programs

1. Test rotate method for struct

```
struct Room {
   wall a;
   bed  b;
   desk c;
```

```
    door d;
};

int main()
{
  Room r;
  r.c = desk(0.5, 0.6);
  r.d = door(0.7, 0.8);

  printS("before roate:");
  put(r, 1.0, 2.0);

  printS("after rotate 90 degree.....");
  rotate(r, 90);
  put(r, 3.0, 4.0);

  render();
  return 0;
}
```

2. Test structs

```
struct Kitchen {
    wall a;
    bed  b;
    desk c;
    door d;
};

struct Bedroom {
    window e;
    rectangle f;
    circle g;
};

int test(){
  Bedroom r;
  r.e = window(0.7, 0.8);
```

```
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.1, 1.2, 1.0);
  put(r, 5.0, 5.0);
}

int main()
{
  Kitchen k;
  Bedroom r;
  wall a;

  k.a = wall(0.1, 0.2);
  k.b = bed(0.3, 0.4);
  k.c = desk(0.5, 0.6);
  k.d = door(0.7, 0.8);
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.0, 1.0, 1.0);
  test();
  a = wall(0.5, 0.5);
  put(k, 1.0, 2.0);
  put(r, 3.0, 4.0);
  put(a, 5.0, 6.0);
  render();
  return 0;
}
```

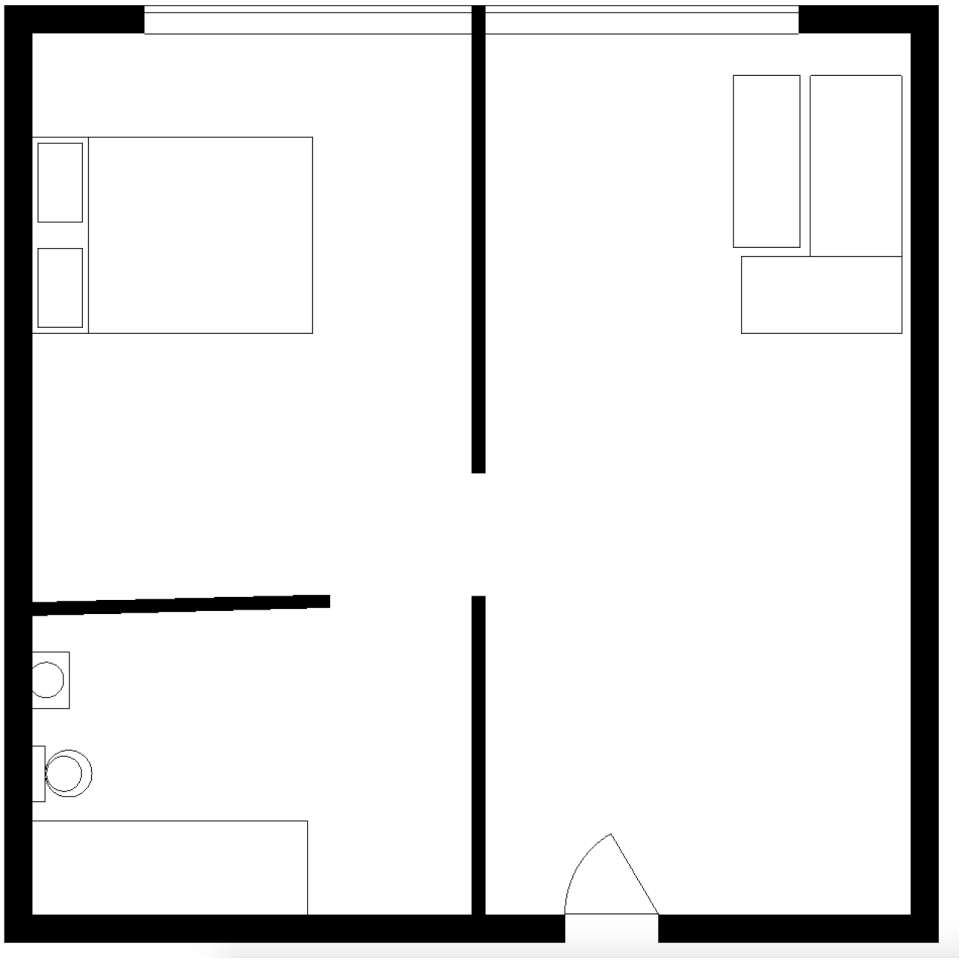
3. Test case for demo

```
/* user-defined struct Toilet*/
struct Toilet {
    circle c0;
    circle c1;
    rectangle r;
};

/* user-defined struct Sink*/
struct Sink {
    circle c;
    rectangle r;
```

```
};

/* user-defined function makeBathroom*/
void makeBathroom(){
  Toilet toilet;
  Sink sink;
  rectangle bathtub;

  toilet.c0 = circle(0.4, 0.3, 0.15);
  toilet.c1 = circle(0.45, 0.3, 0.2);
  toilet.r = rectangle(0.2, 0.6);
  sink.c = circle(0.15, 0.3, 0.15);
  sink.r = rectangle(0.4, 0.6);
  bathtub = rectangle(3.0, 1.1);

  put(bathtub, 0.24, 0.2);
  put(toilet, 0.24, 1.5);
  put(sink, 0.3, 2.5);
}

int main()
{
  /* declare basic FPL objects */
  bed b;
  desk d;
  door dr;
  wall upperMiddle;
  wall lowerMiddle;
  wall upperLeft;
  wall upperRight;
  wall lowerLeft;
  wall lowerRight;
  wall left;
  wall right;
  wall bedBathWall;
  window w;

  /* instantiate FPL objects */
  b = bed(3.0, 2.1);
  d = desk(1.8, 2.75);
  dr = door(1.0, 0.5);
  upperMiddle = wall(0.15, 5.0);
  lowerMiddle = wall(0.15, 3.7);
  upperLeft = wall(1.5, 0.3);
```

```
  upperRight = wall(1.5, 0.3);
  lowerLeft = wall(6.0, 0.3);
  lowerRight = wall(2.7, 0.3);
  left = wall(0.3, 10.0);
  right = wall(0.3, 10.0);
  bedBathWall = wall(0.15, 3.2);
  w = window(7.0, 0.3);

  /* put FPL objects on the specific position*/
  put(b, 0.3, 6.5);
  put(d, 7.8, 6.5);
  put(dr, 6.0, 0.3);
  put(upperMiddle, 5.0, 5.0);
  put(lowerMiddle, 5.0, 0.0);
  put(upperLeft, 0.0, 9.7);
  put(upperRight, 8.5, 9.7);
  put(lowerLeft, 0.0, 0.0);
  put(lowerRight, 7.0, 0.0);
  put(left, 0.0, 0.0);
  put(right, 9.7, 0.0);
  put(w, 1.5, 9.7);

  /* call user defined function */
  makeBathroom();

  /* rotate and put */
  rotate(bedBathWall,90);
  put(bedBathWall, 1.8 , 2.0);

  render();
  return 0;
}
```

Result:



B. Test suites for the translator

We continue to use the test scripts from MicroC, and change it a little to fit our own test needs. Our test flow is in two parts, semantic check (testcheck.sh) and drawing graph (testall.sh). The basic C features tests are also in semantic check test flow. The tests use the same script from MicroC to check these basic features. In drawing graph tests, we delete the comparison parts in MicroC because that is designed for basic C features test and semantic check. This script runned the drawing graph fpls, and the graphs shown on the screen one by one. We can easily check the output of the graphs through the scripts. In linking the OpenGL libraries, it is different in Mac OS X and Linux. These scripts help us to use correct option linking these libraries.

```
LFLAGS=" -lGL -lGLU -lglut -lm"

if [ "$(uname)" == "Darwin" ]; then
    LFLAGS=" -framework OpenGL -framework GLUT"
```

```
fi
```

C. Why and how these test cases were chosen

In our test flow, we make the test Helloworld at first (test-helloWorld.fpl). It tested some basic elements which from MicroC and also tested basic char and string types.
Through our work continues, we test every unit of our languages. Lines, circle and rectangle are the first elements we implemented (test-drawLine.fpl), so that the tests are written together. Later, we add some built-in types such as desk, door and wall. Each of these types is tested singly (test-desk.fpl). In this part, we wrote semantic check for these types, too.
Lastly, the struct and rotate function is written into our compiler. We test the struct and rotate function individually(test-struct.fpl, test-rotate.fpl). Then we combined the two new features to make a test (test-struct-rotate.fpl). In this step, we also do semantic check on the new feature of struct. Our final test file is test-demo.fpl, which draws the entire floor plan.

D. Automated testing

The testcheck.sh test script in the root directory compiles, runs, and links all the files within the check directory. The files must begin with either “test-” or “fail-” and must have the extension “.fpl”. They must also be accompanied by a file with the same base name and the extension “.out” or “.err” if it start with a ”test-” or “fail-” respectively. The output of the test script is shown as follows:

```
-n fail-assign1...
OK
-n fail-assign2...
OK
-n fail-assign3...
OK
-n fail-dead1...
OK
-n fail-dead2...
OK
-n fail-expr1...
OK
-n fail-expr2...
OK
-n fail-for1...
OK
-n fail-for2...
OK
-n fail-for3...
OK
-n fail-for4...
OK
-n fail-for5...
OK
-n fail-func1...
OK
-n fail-func2...
OK
-n fail-func3...
OK
-n fail-func4...
OK
-n fail-func5...
OK
-n fail-func6...
OK
-n fail-func7...
OK
```

```
-n fail-func8...
OK
-n fail-func9...
OK
-n fail-global1...
OK
-n fail-global2...
OK
-n fail-if1...
OK
-n fail-if2...
OK
-n fail-if3...
OK
-n fail-nomain...
OK
-n fail-object1...
OK
-n fail-object2...
OK
-n fail-object3...
OK
-n fail-return1...
OK
-n fail-return2...
OK
-n fail-struct1...
OK
-n fail-struct2...
OK
-n fail-struct3...
OK
-n fail-while1...
OK
-n fail-while2...
OK
-n
test-add1...
```

```
OK
-n
test-arith1...
OK
-n
test-arith2...
OK
-n
test-arith3...
OK
-n
test-fib...
OK
-n
test-for1...
OK
-n
test-for2...
OK
-n
test-func1...
OK
-n
test-func2...
OK
-n
test-func3...
OK
-n
test-func4...
OK
-n
test-func5...
OK
-n
test-func6...
OK
-n
```

```
test-func7...
OK
-n
test-func8...
OK
-n
test-gcd...
OK
-n
test-gcd2...
OK
-n
test-global1...
OK
-n
test-global2...
OK
-n
test-global3...
OK
-n
test-hello...
OK
-n
test-if1...
OK
-n
test-if2...
OK
-n
test-if3...
OK
-n
test-if4...
OK
-n
test-if5...
OK
```

```
-n
test-local1...
OK
-n
test-local2...
OK
-n
test-ops1...
OK
-n
test-ops2...
OK
-n
test-printbig...
OK
-n
test-var1...
OK
-n
test-var2...
OK
-n
test-while1...
OK
-n
test-while2...
OK
```

Moreover, we have testall.sh test script in the root directory compiles, runs, and links all the files within the tests directory. These files runs to show the floor plan test images. For example, the following image shows the test for bed:

Dongdong and Namo did all the test cases of different user defined structures and user defined types. Xinwei did the test case of the demo in the final project presentation. Yipeng did all the test cases of basic language features such as arithmetic operations and general semantic correctness.

VII. Lessons Learned

A. Dongdong She

I learned basic functional programing skill in OCaml and LLVM API by developing compiler code. I also gained experience to write graph programming in OpenGL since I spend most of time to develop the native graph library. Last lesson I learned is gcc compiling, linking technique. One advice I would leave for future team using OpenGL library, don't try to develop in C++, since it has a mangle issue which cause linking error, write your OpenGL code with the ancient original API in C.

B. Yipeng Zhou

In this project, I got more thorough understanding in semantic checking techniques and the whole compiler. It is my first time to use a functional programing language such as OCaml to do some efficitive work. The test automation tools are also helped me a lot in the design of the compilers. After finishing my work, I want to remind the future

students to understand the code of your teammates as clear as possible, which will save your time on many unnecessary debugs.

C. Xinwei Zhang

This was the first sizeable group programming project I have worked on. This was also my first time using Github as version control tool to manage the project. I realized that writing code in real world is not just write something works, but write something that everyone can follow. It was hard for me to read other members' code at first. As time went on, I began more and more comfortable reading others' code and writing effective and clear code by myself. In project management perspective, the most important lesson I learnt from was that we should make everyone on the same page during weekly meeting. Weekly meeting is an important and needful tool to summarize and review the work we have done and to assign tasks for the following week. So my advice is that make sure the weekly meeting is effective and efficient, and this will become the key to success.

D. Chih-Hung Lu

From this project, I learned the development process of compilers. Also, I spent most of the time to figure out how to translate FPL objects into LLVM IR. Fortunately, I came up with an idea, and discuss with my teammates about the feasibility. Finally, I was able to implement the framework from the scratch to a real system. I enjoyed working on this project because building something from the ground is really awesome. And all of the team members made their effort and time on this project reasonably. I believed that It is the reason making this project successfully.

VIII. Appendix

A. Makefile

```
# Author: Chih-Hung Lu
# Make sure ocamlbuild can find opam-managed packages: first run
#
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

CPATH = source/c
MLPATH = source/ml

.PHONY : all
all : fpl.native $(CPATH)/printbig.o $(CPATH)/fplFunctions.o
```

```
.PHONY : fpl.native
fpl.native : cleanObjectC
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
		$(MLPATH)/fpl.native


# "make clean" removes all generated files


.PHONY : clean
cleanObjectC:
	@rm -rf $(CPATH)/printbig.o
	@rm -rf $(CPATH)/fplFunctions.o

clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff fpl scanner.ml parser.ml parser.mli
	rm -rf $(CPATH)/printbig.o
	rm -rf $(CPATH)/fplFunctions.o
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe *.S

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx fpl.cmx

fpl : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o
fpl

scanner.ml : scanner.mll
	ocamllex $(MLPATH)/scanner.mll

parser.ml parser.mli : parser.mly
	ocamlyacc $(MLPATH)/parser.mly

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<
```

```
# Testing the "printbig" example

printbig : $(CPATH)/printbig.c
	cc -o printbig -DBUILD_TEST $(CPATH)/printbig.c

fplFunctions : $(CPATH)/fplFunctions.c
	cc -o fplFunctions  -DBUILD_TEST $(CPATH)/fplFunctions.c

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
fpl.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
fpl.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3      \
   func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3   \
   hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2      \
   while1 while2 printbig

FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2        \
   for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8     \
   func9 global1 global2 if1 if2 if3 nomain return1 return2 while1      \
   while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
            $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = $(MLPATH)/ast.ml $(MLPATH)/codegen.ml Makefile _tags
$(MLPATH)/fpl.ml $(MLPATH)/parser.mly README \
      scanner.mll semant.ml testall.sh $(CPATH)/printbig.c $(CPATH)/fplFunctions.c
arcade-font.pbm font2c \
	$(TESTFILES:%=tests/%)
```

```
fpl-llvm.tar.gz : $(TARFILES)
	cd .. && tar czf fpl-llvm/fpl-llvm.tar.gz \
		$(TARFILES:%=fpl-llvm/%)
```

B. testDemo.sh

```
# Author: Xinwei Zhang
#!/bin/sh

# Regression testing script for Fpl
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

CPATH="source/c"
LFLAGS=" -lGL -lGLU -lglut -lm"

if [ "$(uname)" == "Darwin" ]; then
   LFLAGS=" -framework OpenGL -framework GLUT"
fi



# Path to the fpl compiler.  Usually "./fpl.native"
# Try "_build/fpl.native" if ocamlbuild was unable to create a symbolic link.
FPL="./fpl.native"
#FPL="_build/fpl.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
```

```
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.fpl files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
```

```
    echo $* 1>&2
    eval $* && {
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                        s/.fpl//'`
    reffile=`echo $1 | sed 's/.fpl$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "\n$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
    Run "$FPL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${LFLAGS}""
${CPATH}/printbig.o" "${CPATH}/fplFunctions.o"
    Run "./${basename}.exe"
    #Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    #if [ $error -eq 0 ] ; then
        #if [ $keep -eq 0 ] ; then
        #    rm -f $generatedfiles
        #fi
        #echo "OK"
        #echo "###### SUCCESS" 1>&2
    #else
        #echo "###### FAILED" 1>&2
        #globalerror=$error
    #fi
}
```

```
CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                        s/.fpl//'`
   reffile=`echo $1 | sed 's/.fpl$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
   RunFail "$FPL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
   Compare ${basename}.err ${reffile}.err ${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
         if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
         fi
         echo "OK"
         echo "###### SUCCESS" 1>&2
   else
         echo "###### FAILED" 1>&2
         globalerror=$error
   fi
}

while getopts kdpsh c; do
   case $c in
         k) # Keep intermediate files
            keep=1
            ;;
         h) # Help
            Usage
            ;;
   esac
done
```

```
shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f ${CPATH}/printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ ! -f ${CPATH}/fplFunctions.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-demo.fpl"
fi

for file in $files
do
    case $file in
         *test-*)
            Check $file 2>> $globallog
            ;;
         *fail-*)
            CheckFail $file 2>> $globallog
            ;;
         *)
            echo "unknown file type $file"
            globalerror=1
            ;;
```

```
   esac
done

exit $globalerror
```

C. testall.sh

```
# Author: Chih-Hung Lu
#!/bin/sh

# Regression testing script for Fpl
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

CPATH="source/c"
LFLAGS=" -lGL -lGLU -lglut -lm"

if [ "$(uname)" == "Darwin" ]; then
   LFLAGS=" -framework OpenGL -framework GLUT"
fi


# Path to the fpl compiler.  Usually "./fpl.native"
# Try "_build/fpl.native" if ocamlbuild was unable to create a symbolic link.
FPL="./fpl.native"
#FPL="_build/fpl.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
```

```
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.fpl files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
```

```
    eval $* && {
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\//
                        s/.fpl//'`
    reffile=`echo $1 | sed 's/.fpl$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "\n$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
    Run "$FPL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${LFLAGS}""
${CPATH}/printbig.o" "${CPATH}/fplFunctions.o"
    Run "./${basename}.exe"
    #Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    #if [ $error -eq 0 ] ; then
        #if [ $keep -eq 0 ] ; then
        #    rm -f $generatedfiles
        #fi
        #echo "OK"
        #echo "###### SUCCESS" 1>&2
    #else
        #echo "###### FAILED" 1>&2
        #globalerror=$error
    #fi
}
```

```
CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\//
                        s/.fpl//'`
   reffile=`echo $1 | sed 's/.fpl$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
   RunFail "$FPL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
   Compare ${basename}.err ${reffile}.err ${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
         if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
         fi
         echo "OK"
         echo "###### SUCCESS" 1>&2
   else
         echo "###### FAILED" 1>&2
         globalerror=$error
   fi
}

while getopts kdpsh c; do
   case $c in
         k) # Keep intermediate files
            keep=1
            ;;
         h) # Help
            Usage
            ;;
   esac
done

shift `expr $OPTIND - 1`
```

```
LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f ${CPATH}/printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ ! -f ${CPATH}/fplFunctions.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/*.fpl"
fi

for file in $files
do
    case $file in
         *test-*)
            Check $file 2>> $globallog
            ;;
         *fail-*)
            CheckFail $file 2>> $globallog
            ;;
         *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
```

```
done

exit $globalerror
```

D. testcheck.sh

```
# Author: Yipeng Zhou
#!/bin/sh

# Regression testing script for Fpl
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

CPATH="source/c"
LFLAGS=" -lGL -lGLU -lglut -lm"

if [ "$(uname)" == "Darwin" ]; then
    LFLAGS=" -framework OpenGL -framework GLUT"
fi


# Path to the fpl compiler.  Usually "./fpl.native"
# Try "_build/fpl.native" if ocamlbuild was unable to create a symbolic link.
FPL="./fpl.native"
#FPL="_build/fpl.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
```

```
keep=0

Usage() {
   echo "Usage: testall.sh [options] [.fpl files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
         echo "FAILED"
         error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
         SignalError "$1 differs"
         echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
         SignalError "$1 failed on $*"
         return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
   echo $* 1>&2
   eval $* && {
```

```
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                        s/.fpl//'`
    reffile=`echo $1 | sed 's/.fpl$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "\n$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
    Run "$FPL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${LFLAGS}""
${CPATH}/printbig.o" "${CPATH}/fplFunctions.o"
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

CheckFail() {
```

```
    error=0
    basename=`echo $1 | sed 's/.*\\///
                          s/.fpl//'`
    reffile=`echo $1 | sed 's/.fpl$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$FPL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
          if [ $keep -eq 0 ] ; then
             rm -f $generatedfiles
          fi
          echo "OK"
          echo "###### SUCCESS" 1>&2
    else
          echo "###### FAILED" 1>&2
          globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
          k) # Keep intermediate files
             keep=1
             ;;
          h) # Help
             Usage
             ;;
    esac
done

shift `expr $OPTIND - 1`
```

```
LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail



if [ $# -ge 1 ]
then
   files=$@
else
   files="check/*.fpl"
fi

for file in $files
do
   case $file in
        *test-*)
           Check $file 2>> $globallog
           ;;
        *fail-*)
           CheckFail $file 2>> $globallog
           ;;
        *)
           echo "unknown file type $file"
           globalerror=1
           ;;
   esac
done

exit $globalerror
```

E. fplFunctions.c

```
/*
 * Author: Chih-Hung Lu Dongdong She
 * Here we gonna draw a FPL using openGL with spiritual style
~~~~~~~~~~~!!!!!!!!!!@@@@@@@@@@############
 *
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>

/**************super important********************/
/* OpenGL coordinate range X = [-2,2], Y = [-2, 2], coordinate obtained from FPL is
X=[0,10], Y = [0,10] , so we would do a coordinate calibration */

//global float array to store every object's data to be drawn
static float data[64][6] =  {{0,0,0,0,0,0}};
static int index = 0;

//Called when a key is pressed
void handleKeypress(unsigned char key, int x, int y) {
	switch (key) {
		case 27: //Escape key
			exit(0);
	}
}

//Initializes 3D rendering
void initRendering() {
	//Makes 3D drawing work when something is in front of something else
	glEnable(GL_DEPTH_TEST);
}

//Called when the window is resized
void handleResize(int w, int h) {
	//Tell OpenGL how to convert from coordinates to pixel values
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective
	//Set the camera perspective
	glLoadIdentity(); //Reset the camera
	gluPerspective(45.0,                  //The camera angle
				(double)w / (double)h, //The width-to-height ratio
				1.0,                   //The near z clipping coordinate
				200.0);                //The far z clipping coordinate
}

// real wall
void put_wall(float x, float y, int degree, float shiftX, float shiftY){
   data[index][0] = 1.0;
   data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
```

```
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}

// real bed
void put_bed(float x, float y, int degree, float shiftX, float shiftY){
    data[index][0] = 2.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}

//not a desk in your studying room, but a bunch of sofa and coffee table in living room
void put_desk(float x, float y, int degree, float shiftX, float shiftY){
    data[index][0] = 3.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}

// real door
void put_door(float x, float y, int degree, float shiftX, float shiftY){
    data[index][0] = 4.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}
```

```
// real window
void put_window(float x, float y, int degree, float shiftX, float shiftY){
    data[index][0] = 5.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}

// real rectangle
void put_rectangle(float x, float y, int degree, float shiftX, float shiftY){
    data[index][0] = 6.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) degree;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, degree:%d, shiftX:%f, shigtY:%f\n", x, y, degree, shiftX, shiftY);
}

// no circle at all.....
void put_circle(float x, float y, float diameter, float shiftX, float shiftY){
    data[index][0] = 7.0;
    data[index][1] = x*0.4 - 2.0 + shiftX * 0.4;
    data[index][2] = y*0.4 - 2.0 + shiftY * 0.4;
    data[index][3] = (float) diameter;
    data[index][4] = shiftX*0.4 -2;
    data[index][5] = shiftY*0.4 - 2;
    index = index + 1;
    printf("x:%f, y:%f, diameter:%f, shiftX:%f, shigtY:%f\n", x, y, diameter, shiftX,
shiftY);
}

void put(int degree, int type, float x, float y, float r, float shiftX, float shiftY){
    printf("Hello put\n");
    printf("type:%d, degree:%d, x:%f, y:%f, r:%f, shiftX:%f, shigtY:%f\n", type, degree,
x, y, r, shiftX, shiftY);

    switch (type) {
        case 0:
```

```
            put_wall(x, y, degree, shiftX, shiftY);
            break;
        case 1:
            put_bed(x, y, degree, shiftX, shiftY);
            break;
        case 2:
            put_desk(x, y, degree, shiftX, shiftY);
            break;
        case 3:
            put_door(x, y, degree, shiftX, shiftY);
            break;
        case 4:
            put_window(x, y, degree, shiftX, shiftY);
            break;
        case 5:
            put_rectangle(x, y, degree, shiftX, shiftY);
            break;
        case 6:
            put_circle(x, y, r, shiftX, shiftY);
            break;
        default:
            break;
    }
}

// here we start to draw a spiritual FPL ~~~~~~!!!!!@@@@@########$$$$$$$$$$
void draw()
{
          //Clear information from last draw
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glMatrixMode(GL_MODELVIEW); //Switch to the drawing perspective
          glLoadIdentity(); //Reset the drawing perspective

    int len = index;
    printf("len : %d\n",len);
    int i;
    for(i = 0; i < len; i++)
    {
        // draw a wall using retangle
        if((int) data[i][0] == 1)
        {
            glTranslatef(0.0f, 0.0f, 0.0f);
            glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
                glBegin(GL_QUADS); //Begin quadrilateral coordinates
```

```
            glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][4], data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][2], -5.0f);
                glVertex3f(data[i][4], data[i][2], -5.0f);
                glEnd(); //End quadrilateral coordinates
        }
        // draw a bed using bunch of lines and shapes, with messy coordinates
computation
        if((int) data[i][0] == 2)
        {
            glTranslatef(0.0f, 0.0f, 0.0f);
            glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
            glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][4], data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][2], -5.0f);
                glVertex3f(data[i][4], data[i][2], -5.0f);
                glEnd(); //End quadrilateral coordinates
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
            glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][1]/50 + data[i][4]*49/50, data[i][2]*29/30+data[i][5]/30,
-5.0f);
                glVertex3f(data[i][1]/50 + data[i][4]*49/50,
data[i][2]*17/30+data[i][5]*13/30, -5.0f);
                glVertex3f(data[i][1]*9/50 + data[i][4]*41/50,
data[i][2]*17/30+data[i][5]*13/30, -5.0f);
                glVertex3f(data[i][1]*9/50 + data[i][4]*41/50, data[i][2]*29/30+data[i][5]/30,
-5.0f);
                glEnd(); //End quadrilateral coordinates
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
            glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][1]/50 + data[i][4]*49/50,
data[i][2]*13/30+data[i][5]*17/30, -5.0f);
                glVertex3f(data[i][1]/50 + data[i][4]*49/50, data[i][2]/30+data[i][5]*29/30,
-5.0f);
                glVertex3f(data[i][1]*9/50 + data[i][4]*41/50, data[i][2]/30+data[i][5]*29/30,
-5.0f);
                glVertex3f(data[i][1]*9/50 + data[i][4]*41/50,
data[i][2]*13/30+data[i][5]*17/30,  -5.0f);
                glEnd(); //End quadrilateral coordinates
            glBegin(GL_LINES); //Begin quadrilateral coordinates
            glColor3f(0, 0, 0);    //Trapezoid
```

```
                glVertex3f((data[i][1]/5 + data[i][4]*4/5), data[i][2], -5.0f);
                glVertex3f((data[i][1]/5 + data[i][4]*4/5), data[i][5], -5.0f);
                glEnd(); //End quadrilateral coordinates
        }
        // draw a living sofa and coffe table using bunch of shapes
        if((int) data[i][0] == 3)
        {
           glTranslatef(0.0f, 0.0f, 0.0f);
           glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
           glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][4], data[i][5]*2/3 + data[i][2]/3, -5.0f);
                glVertex3f(data[i][1]*2/5 + data[i][4]*3/5, data[i][5]*2/3 + data[i][2]/3, -5.0f);
                glVertex3f(data[i][1]*2/5 + data[i][4]*3/5, data[i][2],  -5.0f);
                glVertex3f(data[i][4], data[i][2], -5.0f);
                glEnd(); //End quadrilateral coordinates
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
           glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][4]*27/50 + data[i][1]*23/50, data[i][2], -5.0f);
                glVertex3f(data[i][4]*27/50 + data[i][1]*23/50, data[i][5]*21/30 +
data[i][2]*9/30, -5.0f);
                glVertex3f(data[i][1], data[i][5]*21/30 + data[i][2]*9/30, -5.0f);
                glVertex3f(data[i][1], data[i][2], -5.0f);
                glEnd(); //End quadrilateral coordinates
                glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
           glColor3f(0, 0, 0);    //Trapezoid
                glVertex3f(data[i][4]*19/20 + data[i][1]/20, data[i][5]*21/30 +
data[i][2]*9/30, -5.0f);
                glVertex3f(data[i][4]*19/20 + data[i][1]/20, data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][5], -5.0f);
                glVertex3f(data[i][1], data[i][5]*21/30 + data[i][2]*9/30, -5.0f);
                glEnd(); //End quadrilateral coordinates
        }
        // draw a door
        if((int) data[i][0] == 4)
        {
           glTranslatef(0.0f, 0.0f, 0.0f);
           glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
                glBegin(GL_LINE_LOOP);
           glColor3f(0, 0, 0);    //Trapezoid
           float x = data[i][1];
           float y = data[i][5];
                float r = data[i][1] - data[i][4];
           int ii;
```

```
    for(ii = 0; ii < 100; ii++)
        {
            float theta =  3.1415926f * ii / 300;//get the current angle
            float x1 = r * cosf(theta);//calculate the x component
            float y2 = r * sinf(theta);//calculate the y component
            glVertex3f(x - x1, y + y2,-5.0f);//output vertex
        }
          glVertex3f(data[i][1], data[i][5],-5.0f);//output vertex
    glEnd(); //End quadrilateral coordinates
}
// draw a window using retangle
if((int) data[i][0] == 5)
{
    glTranslatef(0.0f, 0.0f, 0.0f);
    glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
        glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
    glColor3f(0, 0, 0);   //Trapezoid
        glVertex3f(data[i][4], data[i][5], -5.0f);
        glVertex3f(data[i][1], data[i][5], -5.0f);
        glVertex3f(data[i][1], data[i][2], -5.0f);
        glVertex3f(data[i][4], data[i][2], -5.0f);
        glEnd(); //End quadrilateral coordinates
    glBegin(GL_LINES); //Begin quadrilateral coordinates
    glColor3f(0, 0, 0);   //Trapezoid
        glVertex3f(data[i][4], data[i][2]*3/4+data[i][5]/4, -5.0f);
        glVertex3f(data[i][1], data[i][2]*3/4+data[i][5]/4, -5.0f);
        glEnd(); //End quadrilateral coordinates
}
if((int) data[i][0] == 6)
{
    glTranslatef(0.0f, 0.0f, 0.0f);
    glRotatef(data[i][3], (data[i][1]+data[i][4])/2, (data[i][2]+data[i][5])/2, -5.0f);
        glBegin(GL_LINE_LOOP); //Begin quadrilateral coordinates
    glColor3f(0, 0, 0);   //Trapezoid
        glVertex3f(data[i][4], data[i][5], -5.0f);
        glVertex3f(data[i][1], data[i][5], -5.0f);
        glVertex3f(data[i][1], data[i][2], -5.0f);
        glVertex3f(data[i][4], data[i][2], -5.0f);
        glEnd(); //End quadrilateral coordinates
}
if((int) data[i][0] == 7)
{
        glBegin(GL_LINE_LOOP);
    glColor3f(0, 0, 0);   //Trapezoid
```

```
            float x = data[i][1];
            float y = data[i][2];
                  float r = data[i][3]/2;
            int ii;
            for(ii = 0; ii < 600; ii++)
                  {
                        float theta =  2 * 3.1415926f * ii / 600;//get the current angle
                        float x1 = r * cosf(theta);//calculate the x component
                        float y2 = r * sinf(theta);//calculate the y component
                        glVertex3f(x + x1, y + y2,-5.0f);//output vertex
                  }
            glEnd(); //End quadrilateral coordinates
         }
      }
            glutSwapBuffers(); //Send the 3D scene to the screen
}

void render(void)
{
   int argc = 1;
   char *argv[1]={""};
         //Initialize GLUT
         glutInit(&argc, argv);
         glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
         glutInitWindowSize(1000, 1000); //Set the window size
         //Create the windoW
         glutCreateWindow("Basic Shapes - programming-technique.blogspot.com");
         initRendering(); //Initialize rendering
         glClearColor(1.0f, 1.0f, 1.0f, 1.0f);//Set handler functions for drawing,
keypresses, and window resizes
         glutDisplayFunc(draw);
         glutKeyboardFunc(handleKeypress);
         glutReshapeFunc(handleResize);
         glutMainLoop(); //Start the main loop
}


#ifdef BUILD_TEST
int main()
{
  put(0, 0, 0.0, 1.0, 1.0, 400.0, 500.0);
  put_wall(0.0, 1.0, 90, 400.0, 500.0);
  put_bed(0.0, 1.0, 180, 600.0, 700.0);
  put_desk(0.0, 1.0, 270, 100.0, 200.0);
```

```
  put_door(0.0, 1.0, 0, 200.0, 300.0);
  put_window(0.0, 1.0, 90, 300.0, 400.0);
  put_rectangle(0.0, 1.0, 180, 400.0, 500.0);
  put_circle(0.0, 1.0, 1.0, 500.0, 600.0);
  render();
}
#endif
```

F. printbig.c

```
/*
 *  A function illustrating how to link C code to code generated from LLVM
 */

#include <stdio.h>

/*
 * Font information: one byte per row, 8 rows per character
 * In order, space, 0-9, A-Z
 */
static const char font[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
  0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
  0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
  0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
  0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
  0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
  0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
  0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
  0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
  0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
  0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
  0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
  0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
  0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
  0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
  0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
  0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
  0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
  0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
```

```
  0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
  0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
  0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
  0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
  0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
  0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
  0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
  0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
  0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
  0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
  0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
  0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
};

void printbig(int c)
{
  int index = 0;
  int col, data;
  if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
  else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
  do {
    data = font[index++];
    for (col = 0 ; col < 8 ; data <<= 1, col++) {
      char d = data & 0x80 ? 'X' : ' ';
      putchar(d); putchar(d);
    }
    putchar('\n');
  } while (index & 0x7);
}


#ifdef BUILD_TEST
int main()
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif
```

G. ast.ml

```
(* Author: Xinwei Zhang, Chih-Hung Lu *)
(* Abstract Syntax Tree and functions for printing it *)

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

type uop = Neg | Not


type typ = Int | Bool | Void | Float | Char | String | Wall | Bed | Desk | Door | Window |
Rectangle | Circle | Struct of string

type bind = typ * string

type expr =
    Literal of int
  | FLiteral of float
  | CharLit of char
  | StringLit of string
  | BoolLit of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | WallStructConstruct of string * string * expr list
  | BedStructConstruct of string * string * expr list
  | DeskStructConstruct of string * string * expr list
  | DoorStructConstruct of string * string * expr list
  | WindowStructConstruct of string * string * expr list
  | RectangleStructConstruct of string * string * expr list
  | CircleStructConstruct of string * string * expr list
  | WallConstruct of string * expr list
  | BedConstruct of string * expr list
  | DeskConstruct of string * expr list
  | DoorConstruct of string * expr list
  | WindowConstruct of string * expr list
  | RectangleConstruct of string * expr list
  | CircleConstruct of string * expr list
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
```

```
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type struct_decl = {
    members: bind list;
    struct_name: string;
  }

type program = {
    globals: bind list;
    functions: func_decl list;
    structs: struct_decl list;
}

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
```

```
let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | FLiteral(f) -> string_of_float f
  | CharLit(c) -> String.make 1 c
  | StringLit(s) -> s
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | WallConstruct(n, el) -> n ^ " = wall(" ^ String.concat ", " (List.map string_of_expr el)
^ ")"
  | BedConstruct(n, el) -> n ^ " = bed(" ^ String.concat ", " (List.map string_of_expr el)
^ ")"
  | DeskConstruct(n, el) -> n ^ " = desk(" ^ String.concat ", " (List.map string_of_expr
el) ^ ")"
  | DoorConstruct(n, el) -> n ^ " = door(" ^ String.concat ", " (List.map string_of_expr
el) ^ ")"
  | WindowConstruct(n, el) -> n ^ " = window(" ^ String.concat ", " (List.map
string_of_expr el) ^ ")"
  | RectangleConstruct(n, el) -> n ^ " = rectangle(" ^ String.concat ", " (List.map
string_of_expr el) ^ ")"
  | CircleConstruct(n, el) -> n ^ " = circle(" ^ String.concat ", " (List.map string_of_expr
el) ^ ")"
  | WallStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = wall(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | BedStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = bed(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | DeskStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = desk(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | DoorStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = door(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | WindowStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = window(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | RectangleStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = rectangle(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | CircleStructConstruct(structName, memberName, el) -> structName ^ "." ^
memberName ^ " = circle(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
```

```
let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Int -> "int"
  | Float -> "float"
  | Bool -> "bool"
  | Void -> "void"
  | Char -> "char"
  | String -> "string"
  | Wall -> "wall"
  | Bed -> "bed"
  | Desk -> "desk"
  | Door -> "door"
  | Window -> "window"
  | Rectangle -> "rectangle"
  | Circle -> "circle"
  | Struct(id) -> id

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_sdecl sdecl =
  "struct" ^ sdecl.struct_name ^ String.concat "{\n" (List.map string_of_vdecl
sdecl.members) ^ "\n}\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program prg =
```

```
  String.concat "" (List.map string_of_vdecl prg.globals) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl prg.functions) ^
  String.concat "\n" (List.map string_of_sdecl prg.structs)
```

H. codegen.ml

```
(* Author: Yipeng Zhou, Xinwei Zhang, Chih-Hung Lu, Dongdong She*)
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

open Ast
module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

(* Error message *)
exception FPL_err of string;;

let translate program =
  let context = L.global_context () in
  let the_module = L.create_module context "Fpl"
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  and flt_t  = L.float_type context
  and str_t  = L.pointer_type (L.i8_type context)
  and void_t = L.void_type context
  and fplObject_t = L.i16_type context
  and struct_t _ = L.i16_type context in

  let ltype_of_typ = function
      A.Int -> i32_t
```

```
  | A.Bool -> i1_t
  | A.Float -> flt_t
  | A.Char -> i8_t
  | A.String -> str_t
  | A.Void -> void_t
  | A.Wall -> fplObject_t
  | A.Bed -> fplObject_t
  | A.Desk -> fplObject_t
  | A.Door -> fplObject_t
  | A.Window -> fplObject_t
  | A.Rectangle -> fplObject_t
  | A.Circle -> fplObject_t
  | A.Struct n -> struct_t n (* unitialized struct type *)
in

(* debug helper *)
(*
let rec getMap map = function
  [] -> map
  | pair::pairs -> getMap (StringMap.add (snd pair) (fst pair) map) pairs in

let printList l =
    List.iter (fun n -> Printf.printf "%s, " (A.string_of_expr n)) l;  Printf.printf "\n" in

let printObjectValueMap m =
   StringMap.iter (fun key value -> Printf.printf "%s: " key;  printList value) m in

let printStringList l =
    List.iter (fun n -> Printf.printf "%s, " n) l;  Printf.printf "\n" in

let printStructMemberMap m =
   StringMap.iter (fun key value -> Printf.printf "%s: " key;  printStringList value) m in
*)

  (* Declare ensureInt and ensureFloat function *)
let ensureInt c =
    if L.type_of c = flt_t then (L.const_fptosi c i32_t) else c in

let ensureFloat c =
    if L.type_of c = flt_t then c else (L.const_sitofp c flt_t) in


(* Declare each global variable; remember its value in a map *)
let global_vars =
```

```
  let global_var m (t, n) =
    let init = L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty program.globals in

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* Declare the built-in printbig() function *)
let printbig_t = L.function_type i32_t [| i32_t |] in
let printbig_func = L.declare_function "printbig" printbig_t the_module in

(* Declare the built-in put() function *)
let put_t = L.function_type i32_t [|i32_t; i32_t; flt_t; flt_t; flt_t; flt_t; flt_t|] in
let put_func = L.declare_function "put" put_t the_module in

(* Declare the built-in render() function *)
let render_t = L.function_type i32_t [||] in
let render_func = L.declare_function "render" render_t the_module in

(* Define each function (arguments and return type) so we can call it *)
let function_decls =
  let function_decl m fdecl =
    let name = fdecl.A.fname
    and formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
    in let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty program.functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  (* fpl map in the function scope *)
  let fplObjectValueMap = ref StringMap.empty in
  let structMemberMap = ref StringMap.empty in
  let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in

  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
  let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder in
  let float_format_str = L.build_global_stringptr "%f\n" "fmt" builder in
  let str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
```

```
    (* Construct the function's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map *)
    let local_vars =
      let add_formal m (t, n) p = L.set_value_name n p;
          let local = L.build_alloca (ltype_of_typ t) n builder in
          ignore (L.build_store p local builder);
          StringMap.add n local m in


    let add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder;
    in StringMap.add n local_var m in

      let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.A.locals in

    (* Return the value for a variable or formal argument *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in
    List.iter (fun pair -> if not(StringMap.mem (snd pair) !structMemberMap) then
        (structMemberMap := StringMap.add (snd pair) ["Dummy"] !structMemberMap))
fdecl.A.locals;
    (*printStructMemberMap !structMemberMap;*)

    (* Construct code for an expression; return its value *)
    let rec expr builder = function
          A.Literal i -> L.const_int i32_t i
      | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
      | A.FLiteral f -> L.const_float flt_t f
      | A.CharLit c -> L.const_int i8_t (Char.code c)
      | A.StringLit s -> L.build_global_stringptr (s^"\x00") "strptr" builder
      | A.Noexpr -> L.const_int i32_t 0
      | A.Id s -> L.build_load (lookup s) s builder
      | A.WallStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
```

```
            (*printStructMemberMap !structMemberMap;*)
            (* append type wall:0 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(0)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            (*printObjectValueMap !fplObjectValueMap;*)
            L.const_int i32_t 0
    | A.BedStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type bed:1 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(1)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.DeskStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type desk:2 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(2)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.DoorStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type door:3 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(3)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.WindowStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
```

```
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type window:4 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(4)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
        | A.RectangleStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type rectangle:5 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(5)] @
act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
        | A.CircleStructConstruct (structName, memberName, act) ->
            let fullName = structName ^ "." ^ memberName in
            let members = StringMap.find structName !structMemberMap in
            let membersWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) members in
            let newMembers = membersWithotDummy @ [fullName] in
            structMemberMap := StringMap.add structName newMembers
!structMemberMap;
            (* append type circle:6 and degree:0 *)
            fplObjectValueMap := StringMap.add fullName ([A.Literal(0); A.Literal(6)] @
act) !fplObjectValueMap;
            L.const_int i32_t 0
        | A.WallConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type wall:0 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(0)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
```

```
            (*printObjectValueMap !fplObjectValueMap;*)
            L.const_int i32_t 0
    | A.BedConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type bed:1 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(1)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.DeskConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type desk:2 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(2)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.DoorConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type door:3 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(3)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
    | A.WindowConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type window:4 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
```

```
A.Literal(4)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
      | A.RectangleConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type rectangle:5 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(5)] @ act @ [A.FLiteral(0.0)]) !fplObjectValueMap;
            L.const_int i32_t 0
      | A.CircleConstruct (variableName, act) ->
            let member = StringMap.find variableName !structMemberMap in
            let memberWithotDummy = List.filter (fun member -> (compare member
"Dummy") != 0) member in
            let newMember = memberWithotDummy @ [variableName] in
            structMemberMap := StringMap.add variableName newMember
!structMemberMap;
            (* append type circle:6 and degree:0 *)
            fplObjectValueMap := StringMap.add variableName ([A.Literal(0);
A.Literal(6)] @ act) !fplObjectValueMap;
            L.const_int i32_t 0
      | A.Binop (e1, op, e2) ->
          let e1' = expr builder e1
    and e2' = expr builder e2 in

    (* Check whether e1 and e2 are float or not.
       If one of them is float, do float operation.
       If not, do int operation *)
    if (L.type_of e1' = flt_t || L.type_of e2' = flt_t) then
    (match op with
      A.Add     -> L.build_fadd
    | A.Sub     -> L.build_fsub
    | A.Mult    -> L.build_fmul
    | A.Div     -> L.build_fdiv
    | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
    | A.Neq     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Leq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq     -> L.build_fcmp L.Fcmp.Oge
    | _         -> raise (FPL_err "Invalid operands for operator")
```

```
) (ensureFloat e1') (ensureFloat e2') "tmp" builder
else
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
  | A.Div     -> L.build_sdiv
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal   -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
) (ensureInt e1') (ensureInt e2') "tmp" builder
  | A.Unop(op, e) ->
  let e' = expr builder e in
  (match op with
            A.Neg     -> L.build_neg
  | A.Not     -> L.build_not) e' "tmp" builder
  | A.Assign (s, e) -> let e' = expr builder e in
                       ignore (L.build_store e' (lookup s) builder); e'
  | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
        L.build_call printf_func [| int_format_str ; (expr builder e) |] "printf" builder
  | A.Call ("printChar", [e])->
L.build_call printf_func [| char_format_str ; (expr builder e) |] "printf" builder
  | A.Call ("printS", [e])->
L.build_call printf_func [| str_format_str ; (expr builder e) |] "printf" builder
  | A.Call ("printFloat", [e])->
L.build_call printf_func [| float_format_str ; (expr builder e) |] "printf" builder
  | A.Call ("putc", [e])->
        L.build_call printf_func [| char_format_str ; (expr builder e) |] "printf" builder
  | A.Call ("printbig", [e]) ->
        L.build_call printbig_func [| (expr builder e) |] "printbig" builder
  | A.Call ("put", act) ->
 let fplStruct = A.string_of_expr (List.hd act) in
 let structMembers = StringMap.find fplStruct !structMemberMap in
 let invokePut offset structMember =
   let attributes = StringMap.find structMember !fplObjectValueMap in
         let parameters = List.map (expr builder) (attributes @ offset) in
   (*printList attributes;*)
         ignore (L.build_call put_func (Array.of_list parameters) "put" builder) in
 let offset = List.tl act in
```

```
      List.iter (fun structMember -> invokePut offset structMember) structMembers;
      L.const_int i32_t 0
    | A.Call ("rotate", act) ->
    let fplStruct = A.string_of_expr (List.hd act) in
    let structMembers = StringMap.find fplStruct !structMemberMap in
    let rotate degree structMember =
      let attributes = StringMap.find structMember !fplObjectValueMap in
      let newAttributes = [degree] @ (List.tl attributes) in
      (*printList attributes;*)
      fplObjectValueMap := StringMap.remove structMember !fplObjectValueMap;
      fplObjectValueMap := StringMap.add structMember newAttributes
!fplObjectValueMap in
    let degree = List.nth act 1 in
      List.iter (fun structMember -> rotate degree structMember) structMembers;
      L.const_int i32_t 0
    | A.Call ("render", act) ->
        let parameters = List.map (expr builder) (act) in
       L.build_call render_func (Array.of_list parameters) "render" builder
    | A.Call (f, act) ->
       let (fdef, fdecl) = StringMap.find f function_decls in
          let actuals = List.rev (List.map (expr builder) (List.rev act)) in
          let result = (match fdecl.A.typ with A.Void -> ""
                                       | _ -> f ^ "_result") in
       L.build_call fdef (Array.of_list actuals) result builder
  in

  (* Invoke "f builder" if the current block doesn't already
     have a terminal (e.g., a branch). *)
  let add_terminal builder f =
    match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
    | None -> ignore (f builder) in

  (* Build the code for the given statement; return the builder for
     the statement's successor *)
  let rec stmt builder = function
        A.Block sl -> List.fold_left stmt builder sl
    | A.Expr e -> ignore (expr builder e); builder
    | A.Return e -> ignore (match fdecl.A.typ with
          A.Void -> L.build_ret_void builder
        | _ -> L.build_ret (expr builder e) builder); builder
    | A.If (predicate, then_stmt, else_stmt) ->
       let bool_val = expr builder predicate in
          let merge_bb = L.append_block context "merge" the_function in
```

```
        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          (L.build_br merge_bb);

        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          (L.build_br merge_bb);

        ignore (L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

    | A.While (predicate, body) ->
          let pred_bb = L.append_block context "while" the_function in
          ignore (L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body" the_function in
          add_terminal (stmt (L.builder_at_end context body_bb) body)
            (L.build_br pred_bb);

          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr pred_builder predicate in

          let merge_bb = L.append_block context "merge" the_function in
          ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
          L.builder_at_end context merge_bb

    | A.For (e1, e2, e3, body) -> stmt builder
            ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
  in

  (* Build the code for each statement in the function *)
  let builder = stmt builder (A.Block fdecl.A.body) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.A.typ with
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in

List.iter build_function_body program.functions;
the_module
```

I. fpl.ml

```
(* Top-level of the Fpl compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

module StringMap = Map.Make(String)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./fpl.native [-a|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;
  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

J. parser.mly

```
/* Author: Xinwei Zhang, Yipeng Zhou, Chih-Hung Lu, Dongdong She */
/* Ocamlyacc parser for Fpl */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token DOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID FLOAT CHAR STRING
```

```
STRUCT
%token WALL WALLCONSTRUCT BED BEDCONSTRUCT DESK
DESKCONSTRUCT DOOR DOORCONSTRUCT
%token WINDOW WINDOWCONSTRUCT RECTANGLE
RECTANGLECONSTRUCT CIRCLE CIRCLECONSTRUCT
%token <int> LITERAL
%token <float> FLOAT_LITERAL
%token <string> ID
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <string> STRUCT_ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left WALL
%left BED
%left DESK
%left DOOR
%left WINDOW
%left RECTANGLE
%left CIRCLE
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { {globals=[]; functions=[]; structs=[]} }
 | decls vdecl { {globals = ($2 :: $1.globals); functions = $1.functions; structs =
$1.structs} }
 | decls fdecl { {globals = $1.globals; functions = ($2 :: $1.functions); structs =
```

```
$1.structs} }
 | decls struct_decl { {globals = $1.globals; functions = $1.functions; structs = ($2 ::
$1.structs)} }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
    { { typ = $1;
          fname = $2;
          formals = $4;
          locals = List.rev $7;
          body = List.rev $8 } }

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

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

struct_decl:
    STRUCT STRUCT_ID  LBRACE vdecl_list RBRACE SEMI
    { { members = $4;
        struct_name = $2; } }

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | CHAR { Char }
  | STRING { String }
  | WALL { Wall }
  | BED { Bed }
  | DESK { Desk }
  | DOOR { Door }
  | WINDOW { Window }
  | RECTANGLE { Rectangle }
  | CIRCLE { Circle }
  | STRUCT_ID { Struct($1) }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }
```

```
vdecl:
   typ ID SEMI { ($1, $2) }

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

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

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

expr:
    LITERAL          { Literal($1) }
  | FLOAT_LITERAL    { FLiteral($1) }
  | CHAR_LITERAL     { CharLit($1) }
  | STRING_LITERAL   { StringLit($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | ID ASSIGN WALL LPAREN actuals_opt RPAREN {WallConstruct($1, $5)}
  | ID ASSIGN BED LPAREN actuals_opt RPAREN {BedConstruct($1, $5)}
  | ID ASSIGN DESK LPAREN actuals_opt RPAREN {DeskConstruct($1, $5)}
  | ID ASSIGN DOOR LPAREN actuals_opt RPAREN {DoorConstruct($1, $5)}
  | ID ASSIGN WINDOW LPAREN actuals_opt RPAREN {WindowConstruct($1, $5)}
  | ID ASSIGN RECTANGLE LPAREN actuals_opt RPAREN {RectangleConstruct($1,
$5)}
  | ID ASSIGN CIRCLE LPAREN actuals_opt RPAREN {CircleConstruct($1, $5)}
  | ID DOT ID ASSIGN WALL LPAREN actuals_opt RPAREN
{WallStructConstruct($1, $3, $7)}
  | ID DOT ID ASSIGN BED LPAREN actuals_opt RPAREN {BedStructConstruct($1,
$3, $7)}
  | ID DOT ID ASSIGN DESK LPAREN actuals_opt RPAREN
```

```
{DeskStructConstruct($1, $3, $7)}
  | ID DOT ID ASSIGN DOOR LPAREN actuals_opt RPAREN
{DoorStructConstruct($1, $3, $7)}
  | ID DOT ID ASSIGN WINDOW LPAREN actuals_opt RPAREN
{WindowStructConstruct($1, $3, $7)}
  | ID DOT ID ASSIGN RECTANGLE LPAREN actuals_opt RPAREN
{RectangleStructConstruct($1, $3, $7)}
  | ID DOT ID ASSIGN CIRCLE LPAREN actuals_opt RPAREN
{CircleStructConstruct($1, $3, $7)}
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }

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

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

K. scanner.mll

```
(* Author: Xinwei Zhang, Yipeng Zhou, Chih-Hung Lu, Dongdong She*)
(* Ocamllex scanner for Fpl *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
```

```
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| '"'      { read_string (Buffer.create 17) lexbuf }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "char"   { CHAR }
| "string" { STRING }
| "void"   { VOID }
| "true"   { TRUE }
| "false"  { FALSE }
| "'(_ as mychar)'" { CHAR_LITERAL(mychar) }
| "wall"   { WALL }
| "bed"    { BED }
| "desk"   { DESK }
| "door"   { DOOR }
| "window" { WINDOW }
| "rectangle" { RECTANGLE }
| "circle" { CIRCLE }
| "struct" { STRUCT }
```

```
| '.'      { DOT }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['0'-'9']*'.'['0'-'9']+ | ['0'-'9']+'.'['0'-'9']* as lxm { FLOAT_LITERAL(float_of_string lxm)}
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as structLit { STRUCT_ID(structLit) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }

(*
This "read_string" function was borrowed directly from this link
https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html
*)
and read_string buf =
  parse
  | '"'       { STRING_LITERAL (Buffer.contents buf) }
  | '\\' '/'  { Buffer.add_char buf '/'; read_string buf lexbuf }
  | '\\' '\\' { Buffer.add_char buf '\\'; read_string buf lexbuf }
  | '\\' 'b'  { Buffer.add_char buf '\b'; read_string buf lexbuf }
  | '\\' 'f'  { Buffer.add_char buf '\012'; read_string buf lexbuf }
  | '\\' 'n'  { Buffer.add_char buf '\n'; read_string buf lexbuf }
  | '\\' 'r'  { Buffer.add_char buf '\r'; read_string buf lexbuf }
  | '\\' 't'  { Buffer.add_char buf '\t'; read_string buf lexbuf }
  | [^ '"' '\\']+
    { Buffer.add_string buf (Lexing.lexeme lexbuf);
      read_string buf lexbuf
    }
  | _ { raise (Failure ("Illegal string character: " ^ Lexing.lexeme lexbuf)) }
  | eof { raise (Failure ("String is not terminated")) }
```

L. semant.ml

```
(* Author: Yipeng Zhou, Xinwei Zhang, Chih-Hung Lu *)
(* Semantic checking for the Fpl compiler *)

open Ast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.
```

```
   Check each global variable, then check each function *)

let check program =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
          n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if (lvaluet=Int) then lvaluet
     else if (Pervasives.(=) lvaluet rvaluet) then lvaluet else raise err
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) program.globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd program.globals);
  (**** Checking structs ****)
  report_duplicate (fun n -> "duplicate struct " ^ n)
    (List.map (fun fd -> fd.struct_name) program.structs);
  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) program.functions)
  then raise (Failure ("function print may not be defined")) else ();

  report_duplicate (fun n -> "duplicate function " ^ n)
    (List.map (fun fd -> fd.fname) program.functions);

  (* Function declaration for a named function *)
```

```
let built_in_decls =  StringMap.add "render"
   { typ = Void; fname = "render"; formals = [];
     locals = []; body = [] } (StringMap.add "rotate"
   { typ = Void; fname = "rotate"; formals = [(Int, "x"); (Int, "y")];
     locals = []; body = [] } (StringMap.add "put"
   { typ = Void; fname = "put"; formals = [(Int, "x"); (Float, "y"); (Float, "z")];
     locals = []; body = [] } (StringMap.add "putc"
   { typ = Void; fname = "putc"; formals = [(Int, "x")];
     locals = []; body = [] }  (StringMap.add "print"
   { typ = Void; fname = "print"; formals = [(Int, "x")];
     locals = []; body = [] } (StringMap.add "printS"
   { typ = Void; fname = "printS"; formals = [(String, "x")];
     locals = []; body = [] } (StringMap.add "printFloat"
   { typ = Void; fname = "printFloat"; formals = [(Float, "x")];
     locals = []; body = [] } (StringMap.add "printChar"
   { typ = Void; fname = "printChar"; formals = [(Char, "x")];
     locals = []; body = [] } (StringMap.add "printb"
   { typ = Void; fname = "printb"; formals = [(Bool, "x")];
     locals = []; body = [] } (StringMap.add "printbig"
   { typ = Void; fname = "printbig"; formals = [(Int, "x")];
     locals = []; body = [] } (StringMap.add "drawLine"
   { typ = Void; fname = "drawLine"; formals = [(Int, "x")];
     locals = []; body = [] }  (StringMap.singleton "drawRec"
   { typ = Void; fname = "drawRec"; formals = [(Int, "x")];
    locals = []; body = [] } ))))))))))))
 in

let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                     built_in_decls program.functions
in

let function_decl s = try StringMap.find s function_decls
     with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

let _ = function_decl "main" in (* Ensure "main" is defined *)

let check_function func=

  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
    " in " ^ func.fname)) func.formals;

  report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
    (List.map snd func.formals);
```

```
  List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
    " in " ^ func.fname)) func.locals;

  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
    (List.map snd func.locals);
  (* Type of each variable (global, formal, or local *)
  let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
          StringMap.empty (program.globals @ func.formals @ func.locals)
  in
  let struct_decls = List.fold_left (fun m sd -> StringMap.add sd.struct_name sd m)
                StringMap.empty program.structs
  in
  let struct_decl s = try StringMap.find s struct_decls
    with Not_found -> raise (Failure ("unrecognized struct " ^ s))
  in

  let type_of_identifier s =
    try StringMap.find s symbols
    with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in
  let type_of_identifier_s s k t=
    try StringMap.find s k
    with Not_found -> raise (Failure ("undeclared identifier " ^ t ^ "." ^s))
  in

(* check if given type is an int or float *)
  let isNumType t = if (t = Int || t = Float) then true else false in

  (* Return the type of an expression or throw an exception *)
  let rec expr = function
          Literal _ -> Int
    | BoolLit _ -> Bool
    | FLiteral _ -> Float
    | CharLit _ -> Char
    | StringLit _ -> String
    | Id s -> type_of_identifier s
    | WallStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Wall) then Wall
```

```
        else raise(Failure ("Excepted type wall for "^ structName ^"."^ memberName
^", got "^string_of_typ name))
    | BedStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Bed) then Bed
        else raise(Failure ("Excepted type bed for "^ structName ^"."^ memberName
^", got "^string_of_typ name))
    | DeskStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Desk) then Desk
        else raise(Failure ("Excepted type desk for "^ structName ^"."^ memberName
^", got "^string_of_typ name))
    | DoorStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Door) then Door
        else raise(Failure ("Excepted type door for "^ structName ^"."^ memberName
^", got "^string_of_typ name))
    | WindowStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Window) then Window
        else raise(Failure ("Excepted type window for "^ structName ^"."^
memberName ^", got "^string_of_typ name))
    | RectangleStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
```

```
      if(name=Rectangle) then Rectangle
        else raise(Failure ("Excepted type rectangle for "^ structName ^"."^
memberName ^", got "^string_of_typ name))
    | CircleStructConstruct(structName, memberName, _) ->
      let sd = struct_decl (string_of_typ (type_of_identifier structName)) in
      let ss = List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (sd.members)
      in
      let name =  type_of_identifier_s memberName ss structName in
      if(name=Circle) then Circle
        else raise(Failure ("Excepted type circle for "^ structName ^"."^ memberName
^", got "^string_of_typ name))
    | WallConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==2) then
      if(List.for_all(isNumType) f)then
        if(name=Wall) then Wall
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for wall"))
      else raise (Failure ("expected numeric input for wall"))
    else raise(Failure("Wrong number of parameters for wall"))

    | BedConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==2) then
      if (List.for_all(isNumType) f)then
        if(name=Bed) then Bed
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for bed"))
      else raise (Failure ("expected numeric input for bed"))
    else raise(Failure("Wrong number of parameters for bed"))

    | DeskConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==2) then
      if (List.for_all(isNumType) f) then
        if(name=Desk) then Desk
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for desk"))
      else raise (Failure ("expected numeric input for desk"))
    else raise(Failure("Wrong number of parameters for desk"))

    | DoorConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==2) then
      if (List.for_all(isNumType) f) then
        if(name=Door) then Door
```

```
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for door"))
      else raise (Failure ("expected numeric input for door"))
    else raise(Failure("Wrong number of parameters for door"))

    | WindowConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==2) then
      if (List.for_all(isNumType) f) then
        if(name=Window) then Window
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for window"))
      else raise (Failure ("expected numeric input for window"))
    else raise(Failure("Wrong number of parameters for window"))

    | RectangleConstruct(n, actuals) -> let name = type_of_identifier n and f =
List.map expr actuals in
    if (List.length f==2) then
      if (List.for_all(isNumType) f) then
        if(name=Rectangle) then Rectangle
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for rectangle"))
      else raise (Failure ("expected numeric input for rectangle"))
    else raise(Failure("Wrong number of parameters for rectangle"))

    | CircleConstruct(n, actuals) -> let name = type_of_identifier n and f = List.map
expr actuals in
    if (List.length f==3) then
      if (List.for_all(isNumType) f)then
        if(name=Circle) then Circle
        else raise(Failure ("wrong type " ^ string_of_typ name ^" for circle"))
      else raise (Failure ("expected numeric input for circle"))
    else raise(Failure("Wrong number of parameters for circle"))

    | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
        (match op with
       Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
       | Add | Sub | Mult | Div when t1 = Int && t2 = Float -> Float
       | Add | Sub | Mult | Div when t1 = Float && t2 = Int -> Float
       | Add | Sub | Mult | Div when t1 = Float && t2 = Float -> Float
              | Equal | Neq when t1 = t2 -> Bool
       | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool
       | Less | Leq | Greater | Geq when t1 = Float && t2 = Float -> Bool
       | Less | Leq | Greater | Geq when t1 = Int && t2 = Float -> Bool
       | Less | Leq | Greater | Geq when t1 = Float && t2 = Int -> Bool
              | And | Or when t1 = Bool && t2 = Bool -> Bool
        | _ -> raise (Failure ("illegal binary operator " ^
```

```
          string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
          string_of_typ t2 ^ " in " ^ string_of_expr e))
    )
  | Unop(op, e) as ex -> let t = expr e in
       (match op with
 Neg when t = Int -> Int
| Neg when t = Float -> Float
       | Not when t = Bool -> Bool
    | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                           string_of_typ t ^ " in " ^ string_of_expr ex)))
  | Noexpr -> Void
  | Assign(var, e) as ex -> let lt = type_of_identifier var
                        and rt = expr e in
    check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                 " = " ^ string_of_typ rt ^ " in " ^
                                 string_of_expr ex))
  | Call(fname, actuals) as call -> let fd = function_decl fname in
     if List.length actuals != List.length fd.formals then
       raise (Failure ("expecting " ^ string_of_int
         (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
     else
       List.iter2 (fun (ft, _) e -> let et = expr e in
          ignore (check_assign ft et
            (Failure ("illegal actual argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
         fd.formals actuals;
       fd.typ
  (*| _ -> raise (Failure ("illegal expression"))*)
in

let check_bool_expr e = if expr e != Bool
 then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
 else () in

(* Verify a statement or throw an exception *)
let rec stmt = function
       Block sl -> let rec check_block = function
       [Return _ as s] -> stmt s
     | Return _ :: _ -> raise (Failure "nothing may follow a return")
     | Block sl :: ss -> check_block (sl @ ss)
     | s :: ss -> stmt s ; check_block ss
     | [] -> ()
    in check_block sl
  | Expr e -> ignore (expr e)
```

```
    | Return e -> let t = expr e in if t = func.typ then () else
       raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                       string_of_typ func.typ ^ " in " ^ string_of_expr e))

    | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
    | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                             ignore (expr e3); stmt st
    | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)

 in
 List.iter check_function program.functions
```

M. Test-bed.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  bed a;
  bed b;
  a = bed(5.0, 3.5);
  b = bed(5.0, 3.5);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
}
```

N. test-char.fpl

```
/* Author: Yipeng Zhou */
int main()
{
        char a;
        a='c';
        printChar(a);
        a='d';
        printChar(a);
        return 0;
}
```

O. test-circle.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  circle a;
  circle b;
  a = circle(1.0, 2.0, 1.0);
  b = circle(3.0, 4.0, 1.0);
  put(a, 1.0, 1.0);
  put(b, 3.0, 4.0);
  render();
  return 0;
}
```

P. test-demo.fpl

```
/* Author: Xinwei Zhang, Chih-Hung Lu */
/* user-defined struct Toilet*/
struct Toilet {
    circle c0;
    circle c1;
    rectangle r;
};

/* user-defined struct Sink*/
struct Sink {
    circle c;
    rectangle r;
};

/* user-defined function makeBathroom*/
void makeBathroom(){
  Toilet toilet;
  Sink sink;
  rectangle bathtub;

  toilet.c0 = circle(0.4, 0.3, 0.15);
  toilet.c1 = circle(0.45, 0.3, 0.2);
  toilet.r = rectangle(0.2, 0.6);
  sink.c = circle(0.15, 0.3, 0.15);
  sink.r = rectangle(0.4, 0.6);
  bathtub = rectangle(3.0, 1.1);

  put(bathtub, 0.24, 0.2);
  put(toilet, 0.24, 1.5);
```

```
  put(sink, 0.3, 2.5);
}

int main()
{
  /* declare basic FPL objects */
  bed b;
  desk d;
  door dr;
  wall upperMiddle;
  wall lowerMiddle;
  wall upperLeft;
  wall upperRight;
  wall lowerLeft;
  wall lowerRight;
  wall left;
  wall right;
  wall bedBathWall;
  window w;

  /* instantiate FPL objects */
  b = bed(3.0, 2.1);
  d = desk(1.8, 2.75);
  dr = door(1.0, 0.5);
  upperMiddle = wall(0.15, 5.0);
  lowerMiddle = wall(0.15, 3.7);
  upperLeft = wall(1.5, 0.3);
  upperRight = wall(1.5, 0.3);
  lowerLeft = wall(6.0, 0.3);
  lowerRight = wall(2.7, 0.3);
  left = wall(0.3, 10.0);
  right = wall(0.3, 10.0);
  bedBathWall = wall(0.15, 3.2);
  w = window(7.0, 0.3);

  /* put FPL objects on the specific position*/
  put(b, 0.3, 6.5);
  put(d, 7.8, 6.5);
  put(dr, 6.0, 0.3);
  put(upperMiddle, 5.0, 5.0);
  put(lowerMiddle, 5.0, 0.0);
  put(upperLeft, 0.0, 9.7);
  put(upperRight, 8.5, 9.7);
  put(lowerLeft, 0.0, 0.0);
```

```
  put(lowerRight, 7.0, 0.0);
  put(left, 0.0, 0.0);
  put(right, 9.7, 0.0);
  put(w, 1.5, 9.7);

  /* call user defined function */
  makeBathroom();

  /* rotate and put */
  rotate(bedBathWall,90);
  put(bedBathWall, 1.8 , 2.0);

  render();
  return 0;
}
```

Q. test-desk.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  desk a;
  desk b;
  a = desk(3.0, 5.0);
  b = desk(3.0, 5.0);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
}
```

R. test-door.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  desk a;
  desk b;
  a = desk(3.0, 5.0);
  b = desk(3.0, 5.0);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
```

```
}
```

S. test-helloWorld.fpl

```
float add(float x, float y)
{
  return x + y;
}

int main()
{
  float a;
  a = add(1.1 , 1.1);
  printFloat(a);
  putc(104); /* h */
  putc(101); /* e */
  putc(108); /* l */
  putc(108); /* l */
  putc(111); /* o */
  putc(32); /* ' ' */
  putc(119); /* w */
  putc(111); /* o */
  putc(114); /* r */
  putc(108); /* l */
  putc(100); /* d */
  putc(10); /* d */
  return 0;
}
```

T. test-rectangle.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  rectangle a;
  rectangle b;
  a = rectangle(0.1, 0.2);
  b = rectangle(0.3, 0.4);
  put(a, 1.0, 2.0);
  put(b, 3.0, 4.0);
  render();
  return 0;
}
```

U. test-rotate.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  wall a;
  door b;
  a = wall(0.2, 0.4);
  b = door(0.6, 0.6);
  printS("before roate:");
  put(a, 1.0, 1.0);
  put(b, 2.0, 2.0);

  printS("after roate 90 degree.....");
  rotate(a, 90);
  rotate(b, 90);

  put(a, 3.0, 3.0);
  put(b, 2.0, 2.0);
  render();
  return 0;
}
```

V. test-string.fpl

```
/* Author: Yipeng Zhou */
int main()
{
        string a;
        a="abcd";
        printS(a);
        printS("add\nstring");
        return 0;
}
```

W. test-struct-rotate.fpl

```
/* Author: Chih-Hung Lu */
struct Room {
   wall a;
   bed  b;
   desk c;
   door d;
};
```

```
int main()
{
  Room r;
  r.c = desk(0.5, 0.6);
  r.d = door(0.7, 0.8);

  printS("before roate:");
  put(r, 1.0, 2.0);

  printS("after rotae 90 degree.....");
  rotate(r, 90);
  put(r, 3.0, 4.0);

  render();
  return 0;
}
```

X. test-struct.fpl

```
/* Author: Chih-Hung Lu */
struct Kitchen {
    wall a;
    bed  b;
    desk c;
    door d;
};

struct Bedroom {
    window e;
    rectangle f;
    circle g;
};

int test(){
  Bedroom r;
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.1, 1.2, 1.0);
  put(r, 5.0, 5.0);
}

int main()
{
  Kitchen k;
```

```
  Bedroom r;
  wall a;

  k.a = wall(0.1, 0.2);
  k.b = bed(0.3, 0.4);
  k.c = desk(0.5, 0.6);
  k.d = door(0.7, 0.8);
  r.e = window(0.7, 0.8);
  r.f = rectangle(0.9, 1.0);
  r.g = circle(1.0, 1.0, 1.0);
  test();
  a = wall(0.5, 0.5);
  put(k, 1.0, 2.0);
  put(r, 3.0, 4.0);
  put(a, 5.0, 6.0);
  render();
  return 0;
}
```

Y. test-wall.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  wall a;
  wall b;
  a = wall(5.0, 0.15);
  b = wall(0.3, 5.0);
  put(a, 0.0, 0.0);
  put(b, 3.0, 3.0);
  render();
  return 0;
}
```

Z. test-window.fpl

```
/* Author: Chih-Hung Lu */
int main()
{
  window a;
  window b;
  a = window(4.0, 1.0);
  b = window(1.0, 4.0);
  put(a, 3.0, 3.0);
```

```
  put(b, 0.0, 0.0);
  render();
  return 0;
}
```

AA. fail-object1.fpl

```
/* Author: Yipeng Zhou */
int main()
{
  door a;
  door b;
  a = desk(4.0, 2.0);
  b = door(2.0, 4.0);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
}
```

BB. fail-object1.err

```
Fatal error: exception Failure("wrong type door for desk")
```

CC. fail-object2.fpl

```
/* Author: Yipeng Zhou */
int main()
{
  door a;
  door b;
  a = door(true, 2.0);
  b = door(2.0, 4.0);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
}
```

DD. fail-object2.err

```
Fatal error: exception Failure("expected numeric input for door")
```

EE. fail-object3.fpl

```
/* Author: Yipeng Zhou */
int main()
{
  door a;
  door b;
  a = door(4.0, 2.0, 3.5);
  b = door(2.0, 4.0);
  put(a, 0.0, 0.0);
  put(b, 5.0, 5.0);
  render();
  return 0;
}
```

FF. fail-object3.err

```
Fatal error: exception Failure("Wrong number of parameters for door")
```

GG. fail-struct1.fpl

```
/* Author: Yipeng Zhou */
struct Kitchen {
    wall a;
    bed  b;
    desk c;
    door d;
};


int main()
{
  Kitchen k;

  k.a = wall(0.1, 0.2);
  k.b = door(0.3, 0.4);
  k.c = desk(0.5, 0.6);
  k.d = door(0.7, 0.8);

  return 0;
}
```

HH. fail-struct1.err

```
Fatal error: exception Failure("Excepted type door for k.b, got bed")
```

II. fail-struct2.fpl

```
/* Author: Yipeng Zhou */
struct Kitchen {
    wall a;
    bed  b;
    desk c;
    door d;
};


int main()
{
  Kitchen k;

  k.e = wall(0.1, 0.2);


  return 0;
}
```

JJ. fail-struct2.err

```
Fatal error: exception Failure("undeclared identifier k.e")
```

KK. fail-struct3.fpl

```
/* Author: Yipeng Zhou */
struct Kitchen {
    wall a;
    bed  b;
    desk c;
    door d;
};
struct Kitchen {
    wall d;
    bed  e;
    desk f;
    door g;
};

int main()
{
  Kitchen k;
```

```
  k.a = wall(0.1, 0.2);
  k.b = door(0.3, 0.4);
  k.c = desk(0.5, 0.6);
  k.d = door(0.7, 0.8);
  return 0;
}
```

LL. fail-struct3.err

```
Fatal error: exception Failure("duplicate struct Kitchen")
```