Mapping **A**pplication **P**rogramming language **M**ade for **e**veryone (**MAPMe**)

Group Members:
Denelle Baptiste - dgb2122
Eric Ellis - ede2106
Jonecia Keels - jtk2128
Changyu Liu - cl2997

□
I. Introduction
    1. Abstract
        1.1 The Problem
    2. Solution
    3. Conclusion
II. Tutorial
III. Language Reference Manual
    1. Introduction
    2. Lexical conventions
        2.1 Comments
        2.2 Key Words
        2.3 Identifier
    3. Types
        A hashtable that takes a string as a key and a list of strings as the value.
        3.2 Object Types
    4. Operators
        4.1. Logical Operators
        4.2. I/O operation
    5. Statement
        5.1 Declaration/Assignment:
        5.2 Looping Construct
        5.3 Built In Methods
    6. Sample Program
IV. Project Plan
    1. Design Process Overview
    2. Code Style Guide
    3. Project Timeline
    4. Team Membership Roles
    5. Software Development Environment
    6. Project Log
*
V. Architectural Design
    1. High-Level Design Overview
    2. Abstract Syntax Tree
    3. Parser
        3.1 XML Parser
    4. Compiler
VI. Test Plan
    1. Testing Suite Overview
    2. Team Member Contribution
VII. Lessons Learned
    1. Team Members thoughts
    2. Advice for Future Teams
VIII. Appendix
    1. Code Listing

I. Introduction

1. Abstract

MAPMe: A simply created, object-applicative, imperative language that allows designers to define, manipulate, and analyze real geographical data.

1.1 The Problem

Currently there is no fully featured, non-commercial solution to map-building available to those outside of a corporate or academic setting. This provides no way for the non-expert demographic to easily find statistical data based on the arrangement of a geographic location. Services like Google Earth and Google Maps provide an interface to view existing geographies but do not provide methods to capture this data into a more useful form.

Solutions to this problem can be developed in widely-used languages like Java and C/C++ but without extremely complex data structures and a strong programming background, using these languages as solutions can be convoluted.

2. Solution

Mapping **A**pplication **P**rogramming language **M**ade for **e**veryone (MAPMe) is designed to allow users the ability to create objects that represent geographical data and apply custom functions to these objects in order to solve a number of different problems. MAPMe is syntactically similar to Java, allowing the language to be comfortably comprehensible for beginners and quickly adaptable for programmers experienced in any other imperative language. In this language we combine the visual and cognitive simplicity of the Document Object Model with the style and functionality of popular programming languages to deliver a mechanism that will allow for construction of detailed geographic models for a wide range of purposes.

Object Applicative
Users begin by describing their map in a XML-styled configuration file where details for object creation are indicated. Since this is a descriptor of the physical world, each item placed on the map is treated like an object. Users can then create functions and apply them to these objects to use in mapping algorithms and to create custom applications.

Dual Front Programming
MAPMe completely isolates the physical description of a map from the operations to be processed on a map. This allows MAPMe code to be instantly modular and applicable to many maps, and makes XML layouts portable and easy to maintain.

XML files are extremely simple to assemble and isolating the two coding structures allows maps to be extremely interchangeable when executing code on several maps.

The real work happens in the .map files. Here users can access the geographical data described in their XML files and develop user defined functions to analyze and operate on that data. Provided are a few basic functions frequently used in a geographical context that will allow users to build their own powerful functions to provide a wealth of physical data.

3. Conclusion

MAPMe provides solid foundation for developing accurate modules for geographic analysis using the pros of the object-oriented paradigm in combination with the complex structures made achievable using the document object model. MAPMe makes geographical analysis easier for the masses by putting it in a familiar package ready to be picked up by first-timers and professionals alike.

II. Tutorial

First Step: Create XML File

Each line of the XML file should be of the form:

<myobject description="Object Type" name="Identifier" type="subtype of Object Type" numOfPoints="N" points="latitude1, longitude1; latitude2, longitude2; … latitudeN, longitudeN";/>

Object Types:

- mapObject
- path

Identifier:

- the variable name by which this object can be referred to in the program

SubTypes:

- mapObject-any tag the user would like to give an object such as restaurant, library, school, tree, dog, person, etc.
- path- any path types such as Road, Trails, Walkways, Sidewalks

N:

- the number of latitude, longitude points being supplied for Object

Latitude, Longitude:

- N latitude, longitude coordinates that describe the position of the object. Precision increases with number of points supplied
-

Second Step: Create MAPMe File

```
~Comments are written like this~

~Stream command loads xml file and create all the objects defined~
stream("ColumbiaU.xml");

~Variable Declaration~
int x;
Float y;
String text;
StringList slist;
SlistHash hashlist;

~Use of Variables~
x = 10;
x = (x +1);
Slisthash hashlist;
StringList slist;
hashlist.add "car" {"BMW", "Volkswagon"};
slist = hashlist.find "car"
print(slist);

~Conditionals~
int i;
string error;
i=0;

if ((i<5))
{ print(i);}
else
{ error = "i is not less than 5"; print (error);}
while ((i<5))
{
   print(i);
}

~users can now refer to objects by the name given in the xml file and xml fields~

if (myCar.type == "Vehicle")
{
float speed;
float distance;
float estTimeArr;
speed = myCar.speed;
distance = getDistance (myDorm.name, Chipotle.name);
estTimeArr = speed/distance;
}

~Users can also reference global arrays of objects~
String tag;
tag = pathArray[1].pathType;

~Calculate intersections~
String interName;
name = intersection(Broadway, Amsterdam);
if ((!strcmp(name, null)))
{

   string msg;
   msg = "There is an intersection between Broadway and Amsterdam.";
   print(msg);
}
```

III. Language Reference Manual

1. Introduction

This manual is a quick reference guide for using MAPMe language. It includes the specific introduction of lexical conventions, types, operators, statements, and samples.

2. Lexical conventions

2.1 Comments

The '~' character introduces a comment and another '~' character ends a comment, except within a character constant or string literal.

The number of the comment character '~' must be even. So the compiler will automatically scan all '~' which are not in a character constant or string literal. If the number of '~' is odd, the compiler will complain. If it is odd, compiler will divide into pairs and treat the content between a pair of '~' as comments.

2.2 Key Words

MAPMe defines several keywords, each with special meaning to the compiler.

- **Point** (Latitude/Longitude Points)
- **Path** (Paths, Trails, Roads, Sidewalks, etc.)
- **Object** (Buildings, Vehicles, People, etc.)
- **Array**
- **HashTable**
- **int, float, string**
- **if, else, while**

2.3 Identifier

In MAPMe, an identifier is a sequence of characters that represents a name for any the following:

- **Variable**
- **Function**
- **Object**
- **Point**
- **Array**
- **Primitive Types: int, float**

Notice that keywords ***cannot*** be identifiers.

3. Types

3.1 Primitive Data Types

3.1.1 String
A sequence of alphanumeric characters (including symbols and the space key) enclosed by quotes

ex. “This is a string”

3.1.2 Integer
A sequence of numeric characters. Integers are real, whole numbers and can be either positive or negative, the latter denoted by initiating the variable instantiation with a minus (-) sign.

ex. 12345

3.1.3 Array
Arrays can hold a user defined number of a data type indicated by the user upon declaration.

Elements of an array can be statically accessed via an index corresponding to the element's position in the array.

ex. int intArray[10];

3.1.4 Float
Floating point numbers can represent any positive or negative decimal value. Floats must contain exactly one decimal point and cannot begin with a decimal point. All floats must begin with an integer.

ex. 4.555 ~this is OK~
ex. .5559 ~incorrect syntax~

3.1.5 StringList
A list that stores Strings as values.

```
Ex. StringList slist;
    slist = {"eric"; "changyu";};
```

3.1.6 HashList

A hashtable that takes a string as a key and a list of strings as the value.

```
Ex. SlistHash hashlist;
    hashlist.add "male students" slist;
```

3.2 Object Types

3.2.1 Path
3.2.2 Object
3.2.3 Point

4. Operators

4.1. Logical Operators

Logical and Assignment Operators	
>	Greater than
<	Less Than
>=	Greater than or equal to

<=	less than or equal to
==	logial equality
||	logical OR
&&	logical AND
!	logical inverter

4.2. I/O operation

I/O Operations	
print	Prints to console
stream	reads file contents to be parse

4.3 Arithmetic and Assignment Operators

Standard arithmetic operators are available for integers and floats: addition (+), subtraction (-), multiplication (*), division (/), and modulus (%). These binary operators can be applied to two integers or floats in the form:

(float *binop* float)
(int *binop* int)

5. Statement

5.1 Declaration/Assignment:

Declaration Format:

[Primitive Data type] identifier;

ex. *double distanceToChipotle;*
float averages[10];

Assignment Format:

identifier = (expression);

ex. averages[1] = (2*5);
distanceToChipotle = (myApartment.getDistanceTo(chipotle));

5.2 Looping Construct

The iteration statement included is started by the while keyword. The while construct is structured as follows:

```
        while (conditional expression)
        {
                statement
  }
```

As long as the the conditional expression evaulates to *true*, the contents of the while block will be executed. The conditional expression is evaluated once before the loop begins and before each iteration of the loop.

5.3 Built In Methods

MAPMe has built in methods for mapping calculations and manipulation.

Logical Operators	**Function Template**	**Function**
getDistance	distance = getDistance (myDorm.name, Chipotle.name);	Calculates and returns the distance in miles from Object1 to Object2
intersection	edgeName = intersection (pathArrays[i].name, pathArray[j].name);	Finds and creates intersection. Function returns the name of intersection that was created. If no intersection exists, null is returned.
strcmp	strcmp(a,"Test")	compares two strings for equivalency and returns *true* or *false*. argument can be either a string literal or variable referencing a string
Other getter functions	myDorm.name or Chipotle.type	Returns the specified fields that are applicable to to Object being operated on

6. Sample Program

Sample XML file named MapData.xml, that stores all of the input data such as Points of Interests, roads, and longitude/latitude positions:

```
<myobject description="path" name="Broadway" type="Road" numOfPoints="4" points="40.804178,-
73.966463;40.806046,-73.966463;40.806712,-73.966463; 40.808011,-73.966463; 40.81048,-73.966463"/>
<myobject description="path" name="Amsterdam" type="Road" numOfPoints="4" points="40.803057,-
73.963888;40.804958, -73.963888;40.806858, -73.963888;40.80819,-73.963888;40.809246,-73.963888"/>
<myobject description="path" name="110thStreet" type="Road" numOfPoints="2" points="40.804178,-73.966463;
40.804178,-73.963888"/>
<myobject description="path" name="111thStreet" type="Road" numOfPoints="2" points="40.804787,-73.966033;
40.804787,-73.963426"/>
<myobject description="mapObject" name="Chipotle" type="Restaurant" point="40.804471,-73.966591"
speed="0"/>
<myobject description="mapObject" name="FiveGuys" type="Restaurant" point="40.804527,-73.966591"
speed="0"/>
<myobject description="mapObject" name="Citibank" type="Bank" point="40.805023,-73.966323" speed="0"/>
</ColumbiaMap>
```

MAPMe Code:

~This method reads in data which creates object types that the user can reference~
```
stream(MapData.xml);

double distanceToChipotle ;
distanceToChipotle =getDistanceTo(myApartment.name, Chipotle.name);
```

~flexibility to calculate estimated time of arrival given users input data~
```
double timeToGetToDest ;
timeToGetToDest = (myCar.speed / distanceToChipotle);
```

IV. Project Plan

1. Design Process Overview

The development cycle of MAPMe was extensive. The general outline was as follows:

- Finalize the standards and functionality of the initial implementation of MAPMe
- Agree on a particular a coding style and code practice
- Set up a svn repository to store and keep track of all code
- Formulate a project timeline where, for the most part, includes weekly deliverables and status updates
- Construct testing scripts as new code is being developed
- Run extensive test cases

- Record any issues, bugs, defects, or new tasks that need to be resolved in the code

Due to the nature of this project, much time in the starting stages of the development cycle was spent on coming up with a programming language and associated demo that everyone agreed on. The project allowed for limitless creativity which caused for each member of the team to have conflicting ideas.

2. Code Style Guide

All source code written in OCaml was adapted to follow to Caml Programming Guidelines. This was the coding standard followed by the members of our group.

This coding standard can be found at Caml Programming Guidelines.

3. Project Timeline

Meetings were held just about every Monday, Wednesday and sometimes Friday for the entire semester.

Major Milestones

* Weeks indicate starting dates of tasks instead of due dates

Week of September 19

- Selection of proposed language MAPMe

Monday, September 26, 2011 - Friday, October 21, 2011

- Defined MAPMe
- Developed syntax and functionality of MAPMe
- Investigated Java for extending graphical support
- Explored mapping algorithms
- Explored Ocaml support for corresponding MAPMe functionality

Week of September 24

- Redefined MAPMe based on suggestions from Professor and Project TA

Week of October 31

- Code a Scanner, Parser, and AST in OCaml
- Created custom objects in OCaml
- Create internal functions

Week of November 7

- Write Demo code
- Create XML parser

Week of November 14 - Week of December 5

- Complete all tasks previously assigned
- Test all units and functions as they are created

Week of December 12 - Week of December 19

- Compile Demo MAPMe code
- Debug and Trouble Shoot
- Complete Final Presentation and Report

4. Team Membership Roles

Jonecia Keels (Team Leader)

Main Responsibilities:

- Created custom object types in OCaml
- Created XML parser
- Organized group meetings
- Generated Automated Test Suites

Denelle Baptiste

Main Responsibilities:

- Created syntax for MAPMe language
- Wrote MAPMe code for Demo
- Created Demo XML file
- Used Project Management tools to help organize project responsibilities

Changyu Liu

Main Responsibilities:

- Created Compiler, Parser, Scanner, and AST

Eric Ellis

Main Responsibilities

- Created Internal Functions in OCaml

While each group member had main responsibilities that they were officially in charge of overseeing, all group members contributed and assisted with various units of this project.

5. Software Development Environment

The primary environment used for the development of MAPMe is as follows:

- Eclipse with OCaml plugin, OcaIDE
    - used for the development of the OCaml compiler, parsers, and abstract syntax tree
    - used to develop and run the testing suites

- SVN repository: Google Code
    - used for storing the project
    - keeping track of changes and updates
    - includes an issue and task tracker
    - can be found at: http://code.google.com/p/mapme-cs4115/
- OUnit: Unit test for OCaml
    - testing framework for OCaml
    - used directly with Eclipse IDE and OCaml plug in

6. Project Log

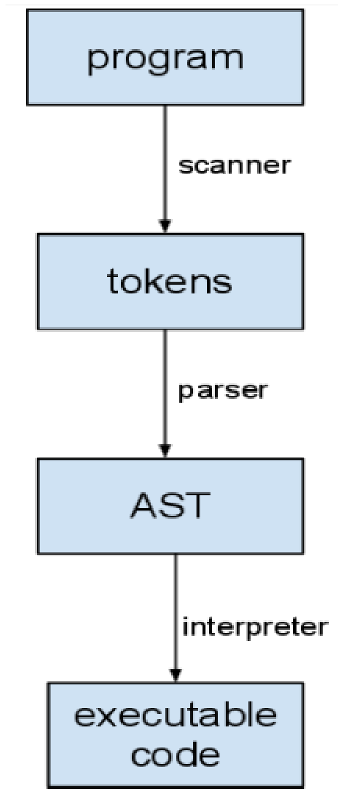
When logging our project tasks we created and resolve issues in Google Code Subversion. This is a spreadsheet comprised of issues that were created and tracked over the course of this project. This is how we were able to keep track of what was being done and by who. If new issues arose, bug defect issues were created and assigned to the corresponding members.

ID	Type	Status	Priority	Owner	Summary	AllLabels
1	Enhancement	Fixed	High	Jonecia Keels	Create function to get length of allObjects and allPaths arrays	Component-Logic, Priority-High, Type-Enhancement
2	Other	Fixed	Critical	Eric Ellis	Finalize Intersection Function	Priority-Critical, Type-Other, Usability
3	Enhancement	Fixed	High	Changyu Liu	Make StringList mutable	Priority-High, Type-Enhancement, Usability
4	Other	Fixed	Critical	Changyu Liu	Create parser to recognize keywords and identifiers	Priority-Critical, Type-Other
5	Task	Fixed	Critical	Eric Ellis	Create Distance Function	Priority-Critical, Type-Task
6	Task	Fixed	Critical	Jonecia Keels	Create Custom Object Types in Ocaml	Priority-Critical, Type-Task
7	Task	Fixed	Critical	Changyu Liu	Add parser logic to recognize conditional statments and loops	Priority-Critical, Type-Task
8	Defect	Fixed	High	Jonecia Keels	array bug in path type	Priority-High, Type-Defect
9	Task	Fixed	High	Jonecia Keels	change mapObject types to store an array of 1 point instead of a single point type	Priority-High, Type-Task
10	Defect	Fixed	High	Jonecia Keels	missing point methods in the edge type	Priority-High, Type-Defect
11	Task	Done	Critical	Jonecia Keels	create function to return the index of a keyword	Priority-Critical, Type-Task
12	Task	Done	Critical	Jonecia Keels	combine syntax parser with the xml config file	Priority-Critical, Type-Task
13	Task	Done	Medium	Changyu Liu	add parsing support for the word 'stream'	Priority-Medium, Type-Task
14	Task	Done	Critical	Denelle Baptiste	code simple demo code	Priority-Critical, Type-Task

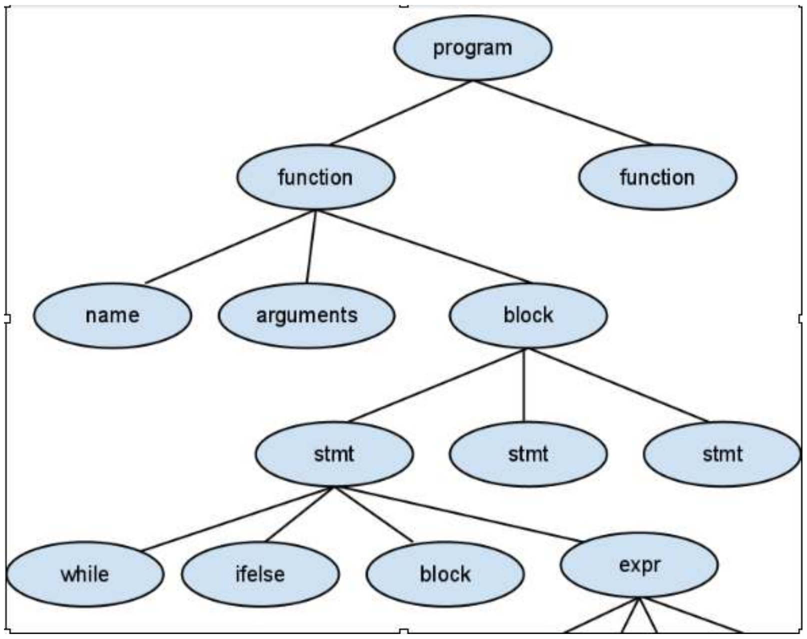
15	Enhancement	Fixed	High	Jonecia Keels	make intersection function take string args instead of path type	Priority-High, Type-Enhancement
16	Defect	Fixed	Critical	Eric Ellis	reading in a file with our language	Priority-Critical, Type-Defect
17	Task	Fixed	Critical	Changyu Liu	support for identifying global arrays	Priority-Critical, Type-Task
18	Defect	WontFix	High	Jonecia Keels	intersect function needs to return a point type	Priority-High, Type-Defect
19	Task	Done	High	Jonecia Keels	create global edge array	Priority-High, Type-Task
20	Defect	Fixed	Critical	Jonecia Keels	support for mapobject in find intersection function	Priority-Critical, Type-Defect
21	Enhancement	Fixed	Medium	Jonecia Keels	change edges name to intersection	Priority-Medium, Type-Enhancement
22	Defect	Accepted	Critical	Changyu Liu	Make changes to MAPMe syntax in parser	Priority-Critical, Type-Defect
23	Task	Done	Medium	Changyu Liu	Add parser logic to recognize comments	Priority-Medium, Type-Task
24	Defect	Fixed	Critical	Jonecia Keels	Add Parser Logic to call Objects by name	Priority-Critical, Type-Defect
25	Task	Done	Critical	Denelle Baptiste	Create Basic Syntax for MAPMe Language	Priority-Critical, Type-Task
26	Defect	Fixed	Medium	Denelle Baptiste	Explore Algorithms Useful for language	Priority-Medium, Type-Defect

V. Architectural Design

1. High-Level Design Overview



2. Abstract Syntax Tree



This is the abstract syntax tree of our language. It is similar in format with the C or C++ language. The tree includes a list of functions. Within each function, there are three fields; name, arguments and block. The block is defined as a list of statements, where the statements has four types: while, if-else, block, and expression. Lastly, the expression includes many cases, which is why they are excluded from the visual diagram.

3. Parser

3.1 XML Parser

Above is a high level overview of the design of our XML parser. The parser is built upon the xml-light library which includes some built-in functions for recognizing xml-like tags. The parser reads in the configuration file and analyzes it. It then checks the ‘description’ tag of each object. The description tag can contain either a the type ‘mapObject’ and ‘path’ which corresponds to our custom mapObject and path types in our language. This lets the compiler know that it needs to allocate memory and initialize and store the specified objects. The parser continues creating objects until it reaches the end of the file.

4. Compiler

There are two main steps in compiling the MAPMe language.

In the first step, the compiler scans the entirity of the program in which it discovers all variable declarations. The program then stores all the variables in the memory. For each type of variable, there are associated maps and arrays, which stores the names and the values of all variables belonging to this specific type. The key of the maps is a string. And the value is an integer. We use a global reference integer that continues increasing every time we add a new variable. This is used to track the value of the map. Now when we go through a variable declaration, we can catch its name and check if it is in the map. If it is in the map, we throw an error message. If not, we add a pair with the key being the name of the variable into the map and the value of the pair point to an element of the associated array. After this, every time we call this variable, we can use its name to find the specific value and then use the value to find the element of the array, which stores the actual value of this variable.

After storing all the variables, we can begin to interpret the program. In this procedure, there are several things we need to pay attention to:

First, the structure of the translator function should be almost the same as the structure of our language. It may not be necessary, but will make the translator function more explicit and reasonable.

Second, the users can define their own functions, but there must be a main function as the entrance of the program. All of the defined functions can be called in the main function. But there is one characteristic about the argument; every function needs to declare local variables in their body whose name is the same as their arguments. So when we call the function, we actually assign its value to the local variables rather than pass the argument into the function. The reason why we implement it this way is because it is a more straight-forward process.

All the expressions return a tuple, which contains several elements with different types like integer, string, float and so on. According to the type of the return value, it will be stored in one of the elements of the tuple, which is the only useful information that is called.

The basic process of the translator function is summarized below:

- The translator function has a local function “stmt”, which execute the statements. And stmt function also has a local function “expr”, which evaluates the expressions.
- the translator function tries to find the main function to enter the program. If there is no main function, it throws an error.
- the translator then executes the body of main function using stmt function.
- the stmt function calls expr function to evaluate the expressions. When we need to call the functions we define, the translator will execute the body of the function, which means every time we call a function, we actually insert the body of the function into the specific place.

VI. Test Plan

1. Testing Suite Overview

We used OUnit testing suite to build, run, and analyze tests for our program.

An example of one of our testing scripts that checks if our getDistance(p1,p2) function works looks like this in OUnit:

```
let test_getDistance = "Compile" >:::
        [
"get_distance" >:: ( fun() ->
                    assert_equal 0.421742034505 (Compile.get_distance “Broadway”,   “110thStreet”);
    assert_equal -1 (Compile.get_distance “Broadway”, “Amsterdam”)
    );
]
```

The above script checks to see if the getDistance() formula outputs the correct calculation given the test variables. An entire testing suite was developed for validating the accuracy of our code.

Below is the results of the tests we developed. After a few fixes after finding bugs, we found that all of our functions passed and the accuracy was accurate as well.

Test	Pass/ Fail
creating correct custom objects from xml	Pass
returning correct distance between two latitude, longitude points using the getDistance(point_1, point_2) function	Pass
returning correct intersection point between two roads using the intersect(path_1, path_2) function	Pass
all objects can be referred to directly by name	Pass
subtypes from objects can be accessed such that users can access attributes from them. Ex. take an object named “Chipotle”. A user can directly access its sub variables by typing: Chipotle.objectType	Pass
Error handling for incorrect MAPMe syntax	Pass
Error handling for incorrect xml syntax	Pass
Data types correctly mapping to its associated value when a user assigns a variable to a value	Pass

2. Team Member Contribution

Everyone helped with developing testing script scenarios. Jonecia was the main person in charge of coding and implementing the testing scripts.

VII. Lessons Learned

1. Team Members thoughts

Changyu Liu:
It is difficult, but interesting. I felt very excited the first time I make my compiler successfully understand what I want to do. And it definitely make me understand much more about what a program language is.

Eric Ellis:
Learn to properly organize an online code repository; a disorganized repository will lead to disorganized code.

Ocaml has an *extremely* high learning curve but once you get it, building projects like compilers become exponentially easier and more straightforward.

Denelle Baptiste:
This project was a great learning experience. It was not without its share of headaches and getting things to work was extremely difficult but the sense of achievement after was a bit satisfying. Nothing about completing this project was easy, but it did allow me to learn lessons that would make things easier in the future.

One important lesson I gathered, group dynamics is very important to a projects success.

Jonecia Keels
This project was both challenging and interesting. Overall I was impressed at what our group accomplished and my coding abilities. I will say that a project of this magnitude not only requires much dedication, but also a group that is team oriented and that collaborates well.

2. Advice for Future Teams

Changyu Liu:
Start early, unless you want to sleep 3 hours every day during the finals and write the final report when everyone go home for Christmas.

Eric Ellis:
Cosign with Changyu - better off sleeping only 6 hours for most of the semester than not sleeping at all in the few weeks after Thanksgiving.

Denelle Baptiste:

Choose your group wisely. It is important to pair with people who you work well with and wouldn’t mind spending days with. The better your group members work together and compliment each other, the easier it will be to get through it.

Jonecia Keels:
The number one advise which is pretty hard to decipher is to choose your team carefully. This project requires that everyone pulls their weight and communicate regularly.

VIII. Appendix

1. Code Listing

Scanner.mll
ast.mli
parser.mly
Compile.ml
Demo
MapObjects.xml

```
(*scanner.mll*) <- Changyu Liu
{ open Parser }

rule token = parse
 [' ' '\t' '\r' '\n'] { token lexbuf }
| '(' {LPAREN}        | ')' {RPAREN}
| '{' {LBRACE}        | '}' {RBRACE}
| '[' {LSQUA}         | ']' {RSQUA}
| '"' {DQUOTATION}    | '\''{SQUOTATION}
| ',' {COMMA}
| '.' {PERIOD}
| ';' {SEMI}
| '+' { PLUS }        | "+."{FPLUS}
| '-' { MINUS }       | "-."{FMINUS}
| '*' { TIMES }       | "*."{FTIMES}
| '/' { DIVIDE }      | "/."{FDIVIDE}
| '=' {ASSIGN}
| "=="{EQ}            | "!="{NEQ}
| '>' {GT}            | '<' {LT}
| ">="{GEQ}           | "<="{LEQ}
| "&&"{AND}           | "||"{OR}
| '!' {NOT}
| "int" {INT}
| "float" {FLO}
| "string" {STRINGV}
| "StringList" {SLIST}
| "return" {RETURN}
| "while"  {WHILE}
| "if"    {IF}      |"else"  {ELSE}
| "StringHash"  {SHASH}
| "SlistHash"    {SLHASH}
| "add"          {ADD}      |"create"    {CREATE}
| "mem"          {MEM}      |"find"      {FIND}
| "print"        {PRINT}
| "intersect"    {INTERSECT}
| "getDistance" {GETDISTANCE}
| "strcmp"       {STRCMP}
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['0'-'9']+ '.' ['0'-'9']+ as lxm {FLOAT(float_of_string lxm)}
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm {ID(lxm)}
| '\'' ['a'-'z' 'A'-'Z' '0'-'9' '_'] '\'' as lxm {CHAR(lxm.[1])}
|'"' ['a'-'z' 'A'-'Z' '0'-'9' ' ' '_']* '"' as lxm {STR(lxm)}
| _ as char {raise (Failure("illegal character" ^ Char.escaped char))}
| eof { EOF }
```

(*ast.mli*) <- Changyu Liu

```
type operator = Add | Sub | Mul | Div | Equal | Greater | Less | Geq | Leq | Neq | And | Or
type foperator = Fadd | Fsub | Fmul | Fdiv


type expr =
   Binop of expr * operator * expr
 | FBinop of expr * foperator * expr
 | Not of expr
 | Lit of int
 | Float of float
 | Id of string
 | Str of string
 | Char of char
 | StringList of string
 | StringListAdd of string * expr
 | Slist of string list
 | Call of string * expr list
 | Int of string
 | Flo of string
 | Assign of string * expr
 | ArrAssign of string * int * expr
 | Stringv of string
 | IntArray of string * int
 | Return of expr
 | Sadd of string * string * expr
 | Smem of string * string
 | Sfind of string * string
 | SLadd of string * string * expr
 | SLmem of string * string
 | SLfind of string * string
 | Scre of string
 | SLcre of string
 | Print of string
 | Intersect of expr * expr
 | GetDistance of expr * expr
 | ObjArray of string * expr * string
 | Objpoint of string  * int * string * string
 | Onedot of string * string
 | Twodot of string * expr * string * expr * string
 | Strcmp of expr * expr

type stmt =
   Expr of expr
 | Block of stmt list
 | While of expr * stmt
 | If of expr * stmt * stmt

type func = {
    fname : string;
    formals : string list;
    body : stmt ;
}

type program = func list
```

(*parser.mly*) <- Changyu Liu

```
%{ open Ast %}

%token PLUS MINUS TIMES DIVIDE EQ NEQ GT LT GEQ LEQ AND OR NOT FPLUS
FMINUS FTIMES FDIVIDE ASSIGN LPAREN RPAREN LBRACE RBRACE COMMA
SEMI INT STRINGV DQUOTATION SQUOTATION LSQUA RSQUA PERIOD RETURN
FLO WHILE IF ELSE SHASH ADD SLIST CREATE SLHASH MEM FIND PRINT
INTERSECT GETDISTANCE STRCMP EOF
%token <int> LITERAL
%token <float> FLOAT
%token <string> ID
%token <string> STR
%token <char> CHAR


%left PLUS MINUS
%left TIMES DIVIDE

%start program
%type < Ast.expr> expr
%type < Ast.stmt> stmt
%type < Ast.func> fdel
%type < Ast.program> program

%%

expr:
 LPAREN expr PLUS   expr RPAREN { Binop($2, Add, $4) }
| LPAREN expr MINUS  expr RPAREN { Binop($2, Sub, $4) }
| LPAREN expr TIMES  expr RPAREN { Binop($2, Mul, $4) }
| LPAREN expr DIVIDE expr RPAREN { Binop($2, Div, $4) }
| LPAREN expr EQ expr RPAREN { Binop($2, Equal, $4) }
| LPAREN expr NEQ expr RPAREN { Binop($2, Neq, $4) }
| LPAREN expr GT expr RPAREN { Binop($2, Greater , $4) }
| LPAREN expr LT expr RPAREN { Binop($2, Less, $4) }
| LPAREN expr GEQ expr RPAREN { Binop($2, Geq, $4) }
| LPAREN expr LEQ expr RPAREN { Binop($2, Leq, $4) }
| LPAREN expr AND expr RPAREN { Binop($2, And, $4) }
| LPAREN expr OR expr RPAREN { Binop($2, Or, $4) }
| LPAREN expr FPLUS   expr RPAREN { FBinop($2, Fadd, $4) }
| LPAREN expr FMINUS  expr RPAREN { FBinop($2, Fsub, $4) }
| LPAREN expr FTIMES  expr RPAREN { FBinop($2, Fmul, $4) }
| LPAREN expr FDIVIDE expr RPAREN { FBinop($2, Fdiv, $4) }
| NOT expr        { Not($2) }
| LITERAL         { Lit($1) }
| FLOAT           { Float($1)}
| ID              { Id($1)}
| CHAR            { Char($1)}
| STR             { Str($1)}
| ID LPAREN actuals_opt RPAREN {Call($1,$3)}
| PRINT LPAREN ID RPAREN      {Print ($3)}
| LSQUA string_list RSQUA     {Slist($2)}
| INT ID          {Int($2)}
| FLO ID          {Flo($2)}
| STRINGV ID      {Stringv($2)}
| SLIST ID        {StringList($2)}
| SLIST PERIOD ADD ID expr {StringListAdd($4,$5) }
| ID ASSIGN expr   {Assign($1,$3)}
| INT ID LSQUA LITERAL RSQUA {IntArray($2,$4)}
| ID LSQUA expr RSQUA PERIOD ID {ObjArray($1,$3,$6)}
| ID LSQUA LITERAL RSQUA PERIOD ID PERIOD ID {Objpoint($1,$3,$6,$8)}
| ID PERIOD ID {Onedot($1,$3)}
| ID LSQUA expr RSQUA PERIOD ID LSQUA expr RSQUA PERIOD ID
{Twodot($1,$3,$6,$8,$11)}
| ID LSQUA LITERAL RSQUA ASSIGN expr {ArrAssign($1,$3,$6)}
| SHASH PERIOD ADD ID STR expr {Sadd ($4,$5,$6)}
| SHASH PERIOD MEM ID STR  {Smem($4,$5)}
| SHASH PERIOD FIND ID STR  {Sfind($4,$5)}
| ID PERIOD ADD STR expr {SLadd ($1,$4,$5)}
| ID PERIOD MEM STR {SLmem($1,$4)}
| ID PERIOD FIND STR {SLfind($1,$4)}
| SHASH PERIOD CREATE ID   {Scre($4)}
| SLHASH ID   {SLcre($2)}
| RETURN expr      {Return($2)}
| INTERSECT LPAREN expr COMMA expr RPAREN {Intersect($3,$5)}
| GETDISTANCE LPAREN expr COMMA expr RPAREN {GetDistance($3, $5)}
| STRCMP LPAREN expr COMMA expr RPAREN  {Strcmp($3,$5)}
```

```
stmt:
 expr SEMI        {Expr($1)}
| LBRACE stmt_list RBRACE        {Block(List.rev $2)}
| WHILE LPAREN expr RPAREN stmt  {While($3,$5)}
| IF LPAREN expr RPAREN stmt ELSE stmt {If($3,$5,$7)}

program:
                {[]}
| program fdel     {$2::$1}

fdel:
ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
{{ fname=$1; formals=$3; body=Block(List.rev $6); }}

formals_opt:
                {[]}
| formal_list     {List.rev $1}

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




var:
 INT ID                {$2}
| STRINGV ID           {$2}
| FLO ID               {$2}



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

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

```
actuals_opt:
             {[]}
| actuals_list {List.rev $1}

string_list:
 STR              {[$1]}
| string_list SEMI STR {$3::$1}
```

```
(*compile.ml*) <- Jonecia Keels, Changyu Liu

open Ast

(*set this to true for debugging output*)
let debug = "false"

(*begin code by Jonecia Keels*)
(*************************************************************************
*************)
let split_str sep str =
 Str.split(Str.regexp_string sep) str

(*custom point type stores longitude/latitude*)
class point =
object
 val mutable longitude = 0.000000
 val mutable latitude = 0.000000
 method get_longitude = longitude
 method get_latitude = latitude
 method set_longitude long = longitude <- long
 method set_latitude lat = latitude <- lat
end;;

(*global test point for storing elements as point type*)
let testPoint = new point
let point = new point

(*custom path type for roads*)
class path =
object
 val mutable name = "null"
 val mutable pathType = "null"
 val mutable numberOfPoints = 0
 val mutable points = Array.make 100 testPoint
 method get_name = name
 method get_pathType = pathType
 method get_numberOfPoints = numberOfPoints
 method get_points = points
 method set_name n = name <- n
 method set_pathType p = pathType <- p
 method set_numberOfPoints n = numberOfPoints <- n
 method set_points p = points <- p
end;;

(*stores the mapObject*)
class mapObject =
object
 val mutable name = "null"
 val mutable objectType = "null"
 val mutable speed = 0
 val mutable point = Array.make 1 testPoint
 method get_name = name
 method get_objectType = objectType
 method get_point = point
 method get_speed = speed
 method set_name n = name <- n
 method set_objectType o = objectType <- o
 method set_speed s = speed <- s
 method set_point p = point <- p
end;;

(*custom intersection object/intersection*)
class intersection =
object
 val mutable name = "null"
 val mutable weight = infinity
 val mutable path_one = "null"
 val mutable path_two = "null"
 val mutable point = new point
 method get_name = name
 method get_weight = weight
 method get_path_one = path_one
 method get_path_two = path_two
 method get_point = point
 method set_name n = name <- n
 method set_weight w = weight <- w
 method set_path_one p1 = path_one <- p1
 method set_path_two p2 = path_two <- p2
 method set_point lat long = point#set_latitude long; point#set_longitude lat
```

```
end;;


(*Global Variables*)
let object_num = ref 0 (*to fill the ObjectArray below*)
let object_count = ref 0 (*keeps track of how many map objecs there are *)

let intersection_num = ref 0
let intersection_count = ref 0

let path_num = ref 0
let path_count = ref 0

(*counts total number of paths*)
let increment_path_count n =
 path_count := !path_count+1;
 if debug = "true" then
   Printf.printf "path count %i\n" path_count.contents

let increment_mapObject_count n  =
 object_count := !object_count+1;
 if debug = "true" then
   Printf.printf "map object count %i\n" object_count.contents

let increment_intersection_count n =
 intersection_count := !intersection_count+1;
 if debug = "true" then
   Printf.printf "map object count %i\n" intersection_count.contents

(*sets a point type*)
let init_point longitude latitude =
 let new_point = new point in
 new_point#set_longitude longitude;
 new_point#set_latitude latitude;
 if debug = "true" then
   Printf.printf "successfully created a new point\n"

(*goes through the xml file first to count the number of each instance of a object type*)
let xml_parser =
 let x = Xml.parse_file "MapMeObjects.xml" in
 Xml.iter (function
 Xml.Element ("myobject", attrs, _) ->
   List.iter (function ("description", desc) ->
     if desc = "path" then
    let name = List.assoc "name" attrs in
    increment_path_count name
     else if desc = "mapObject" then
    let name = List.assoc "name" attrs in
       increment_mapObject_count name
     else if desc = "intersection" then
       let name = List.assoc "name" attrs in
    increment_intersection_count name
     else if desc = "point\n" then
    if debug = "true" then
         Printf.printf "this is a point"
     else Printf.printf "not a recognizable object\n" | _ -> ()) attrs
   | _ -> ()) x

(*arrays that store the objects*)

(*map objects*)
let dummyMapObject = new mapObject (*for initializing*)
let object_named_arr = Array.make object_count.contents "null" (*name array*)
let objArray = Array.make object_count.contents dummyMapObject (*stores the actual map
object*)

(*path objects*)
let dummyPathObject = new path
let path_named_arr = Array.make path_count.contents "null"
let pathArray = Array.make path_count.contents dummyPathObject

(*intersection objects*)
let dummyintersectionObject = new intersection
let inters_named_arr = Array.make 1000 "null"
let intersArray = Array.make 1000 dummyintersectionObject

(*adds map object to the array*)
let add_mapObject name object_type point speed =
 let new_mapObject = new mapObject and
 new_point_array = Array.make 1 testPoint and
 dumb_point = new point and
 p = split_str "," point in
 (*variables := IntMap.add 4 object_num.contents !variables;*)
 new_mapObject#set_name name;
 new_mapObject#set_objectType object_type;
 new_mapObject#set_speed speed;
 (*create a point type for the object*)
 let x = float_of_string(List.nth p 0) and
 y = float_of_string(List.nth p 1) in
 dumb_point#set_latitude x;
 dumb_point#set_longitude y;
 Array.set new_point_array 0 dumb_point;
 new_mapObject#set_point new_point_array;
 objArray.(object_num.contents) <- new_mapObject;
 object_named_arr.(object_num.contents) <- name;
 object_num := !object_num+1;
 if debug = "true" then
   Printf.printf "added map object:%s to array\n" new_mapObject#get_name

(*adds path object to the array*)
let add_path name path_type number_of_points points =
 (*init path array to set in the path object*)
 let new_point_array = Array.make number_of_points testPoint and
 new_path = new path in
 new_path#set_name name;
 new_path#set_pathType path_type;
 new_path#set_numberOfPoints number_of_points;
 (*retrieve the list of points seperated by colons first*)
 let first_parse = split_str ";" points in
 (*let second_parse = split_str "," first_parse in*)
 for i = 0  to (new_path#get_numberOfPoints - 1) do
   let dumb_point = new point in
   let z1 = List.nth first_parse i in
   let x1 = split_str "," z1 in
   let x2 = float_of_string(List.nth x1 0) and
     y1 = float_of_string(List.nth x1 1) in
     dumb_point#set_latitude x2;
     dumb_point#set_longitude y1;
     Array.set new_point_array i dumb_point;
     if debug = "true" then
```

```
    Printf.printf "index: %i Added (%f, %f)\n" i new_point_array.(i)#get_latitude
new_point_array.(i)#get_longitude;
 done;
if debug = "true" then
  Printf.printf " Added (%f, %f)\n" new_point_array.(0)#get_latitude
new_point_array.(0)#get_longitude;
 new_path#set_points new_point_array;


 (*Printf.printf "first list %s \n second list %s";*)
 pathArray.(path_num.contents) <- new_path;
 path_named_arr.(path_num.contents) <- name;
 path_num := !path_num+1;
 if debug = "true" then
   Printf.printf "added path object:%s to array\n" new_path#get_name

(*adds intersection object to the array*)
let add_intersection name weight path_1 path_2 x1 y1 =
 let new_intersection = new intersection in
 new_intersection#set_name name;
 new_intersection#set_weight weight;
 new_intersection#set_path_one path_1;
 new_intersection#set_path_two path_2;
 new_intersection#set_point x1 y1;
 intersArray.(intersection_num.contents) <- new_intersection;
 inters_named_arr.(intersection_num.contents) <- name;
 intersection_num := !intersection_num+1;
 if debug = "true" then
   Printf.printf "added intersection object:%s to array\n" new_intersection#get_name

(*now that we know how many instances of each object is in the xml, we can go through the file
again and add the object to the initialized array*)
let xml_parser_2 =
 let x = Xml.parse_file "MapMeObjects.xml" in
 Xml.iter (function
 Xml.Element ("myobject", attrs, _) ->
   List.iter (function ("description", desc) ->
     if desc = "path" then
    let name = List.assoc "name" attrs and
          p_type = List.assoc "type" attrs and
```

```
        n_points = int_of_string(List.assoc "numOfPoints" attrs) and
        points = List.assoc "points" attrs in
    add_path name p_type n_points points
     else if desc = "mapObject" then
    let name = List.assoc "name" attrs and
        o_type = List.assoc "type" attrs and
        point = List.assoc "point" attrs and
        speed = int_of_string(List.assoc "speed" attrs) in
    add_mapObject name o_type point speed
     else if desc = "intersection" then
    let name = List.assoc "name" attrs and
        weight = float_of_string(List.assoc "weight" attrs) and
        point = float_of_string(List.assoc "point" attrs) and
        point_2 = float_of_string(List.assoc "point" attrs) and
        path_1 = List.assoc "path1" attrs and
        path_2 = List.assoc "path2" attrs in
    add_intersection name weight path_1 path_2 point point_2
     else if desc = "point\n" then
         Printf.printf "this is a point"
     else Printf.printf "not a recognizable object\n" | _ -> ()) attrs
   | _ -> ()) x

(*below are methods used to find the index of the searched array *)
let find_index_objectArray name =
 let rec loop i =
   if i >= Array.length  object_named_arr then -1
   else if  object_named_arr.(i) = name then i
   else loop(i+1) in
loop 0

let find_index_pathArray name =
 let rec loop i =
   if i >= Array.length  path_named_arr then -1
   else if  path_named_arr.(i) = name then i
   else loop(i+1) in
loop 0

(*calculate distance*) <- Eric Ellis
let get_distance object_1 object_2 =
 let point1 = objArray.(find_index_objectArray object_1)#get_point.(0) and
```

```
    point2 = objArray.(find_index_objectArray object_2)#get_point.(0) and
    r = 6371.0 and pi = 4.0 *. atan 1.0 in
   ((acos ((sin (point1#get_latitude *. pi /. 180.0))*.(sin (point2#get_latitude*. pi /. 180.0))+.(cos
(point1#get_latitude*. pi /. 180.0))*.(cos (point2#get_latitude *. pi /. 180.0))*.(cos
((point2#get_longitude*. pi /. 180.0)-.(point1#get_longitude*. pi /. 180.0))))) *. r) *. 0.62137

(*retrieve the total amount of objects in the mapObjects array*)
(*arg should be either the string: mapObject or path*)
let get_num_of_objects object_name =
 if object_name = "mapObject" then
   Array.length objArray
 else if object_name = "path" then
   Array.length pathArray
 else
   raise Not_found

(*find intersection given two string path names*)
let find_intersection p1_name p2_name =
 let index_p1 = find_index_pathArray p1_name and
 index_p2 = find_index_pathArray p2_name and
 intersection_object = new intersection in
 let p1 = pathArray.(index_p1) and
     p2 = pathArray.(index_p2) in
 (*find the index of the path*)
 let intersection_point = new point in
 let x0_1 = p1#get_points.(0)#get_longitude and
 y0_1 = p1#get_points.(0)#get_latitude and
 x1_1 = p1#get_points.((Array.length p1#get_points) - 1)#get_longitude and
 y1_1 = p1#get_points.((Array.length p1#get_points) - 1)#get_latitude and
 x0_2 = p2#get_points.(0)#get_longitude and
 y0_2 = p2#get_points.(0)#get_latitude and
 x1_2 = p2#get_points.((Array.length p2#get_points) - 1)#get_longitude and
 y1_2 = p2#get_points.((Array.length p2#get_points) - 1)#get_latitude in
 if ((y0_1 <= y0_2 && y0_2 < y1_1) && ((x0_1 -. 0.001000 <= x0_2 && x0_2 < x0_1 +.
0.001000) || (x0_1 -. 0.001000 <= x1_2 && x1_2 < x0_1 +. 0.001000))) then begin
   intersection_point#set_longitude x0_1;
   intersection_point#set_latitude y0_2;
   if debug = "true" then
     Printf.printf "intersection at (%f, %f)\n" y0_2 x0_1;
   let string_name = p1_name^"_"^p2_name in
   intersection_object#set_name string_name;
   if debug = "true" then
     Printf.printf "intersection name: %s\n" string_name;
   intersection_object#set_weight 0.0;
   intersection_object#set_path_one p1_name;
   intersection_object#set_path_two p2_name;
   intersection_object#set_point y0_2 x0_1;
   intersArray.(intersection_num.contents) <- intersection_object;
   inters_named_arr.(intersection_num.contents) <- string_name;
   intersection_num := !intersection_num+1;
   if debug = "true" then
     Printf.printf "added intersection object: %s to array\n"
inters_named_arr.(intersection_num.contents -1);
   string_name
 end
 else begin
   let s = "none" in
   intersection_point#set_longitude 10000.000000;
   intersection_point#set_latitude 10000.000000;
   if debug = "true" then
     Printf.printf "NO INTERSECTION\n";
   s
 end

let q = pathArray.(0)#get_name
let w = pathArray.(2)#get_name
let ip = find_intersection q w
let q = pathArray.(0)#get_name
let w = pathArray.(4)#get_name
let ip = find_intersection q w

let pointArray = Array.make 100 point

(****************************************************************************
***************)

(*end code by Jonecia Keels*)
```

```
(*begin code by Chang*)
module StringHash = Hashtbl.Make(struct
 type t = string
 let equal x y = x = y
 let hash = Hashtbl.hash
end)

let myhash = StringHash.create 17;;
StringHash.add myhash "first" ""

let slisthash = StringHash.create 17;;
StringHash.add slisthash "first" [""]

module HashMap = Map.Make(String)
let hmap = HashMap.empty
let hmap = HashMap.add "myhash" 0 hmap

module SlHashMap = Map.Make(String)
let slhmap = SlHashMap.empty


module FunctionMap = Map.Make(String)
let fmap = FunctionMap.empty
let fmap = FunctionMap.add "print_int" 1 fmap
let fmap = FunctionMap.add "print_str" 2 fmap

module NameMap = Map.Make (struct
type t = string
let compare x y = Pervasives.compare x y
end)
let funcmap = NameMap.empty

module TypeMap = Map.Make(String)
let tmap = TypeMap.empty
let tmap = TypeMap.add "int" 1 tmap
let tmap = TypeMap.add "string" 2 tmap

module IdMap = Map.Make(String)
let imap = IdMap.empty
let imap = IdMap.add "Hello" 2 imap
let imap = IdMap.add "hi" 3 imap

module StringListMap = Map.Make(String)
let slistmap = StringListMap.empty

module IntMap = Map.Make(String)
let intmap = IntMap.empty

module FloMap = Map.Make(String)
let flomap = FloMap.empty

module StringMap = Map.Make(String)
let strmap = StringMap.empty

module IntarrMap = Map.Make(String)
let intarrmap = IntarrMap.empty

let intArray = Array.make 50 0
let floArray = Array.make 50 0.0
let strArray = Array.make 50 ""
let strlistArray = Array.make 50 [""]
let array_int = Array.make 30 intArray
let hashArray = Array.make 5 myhash
let slhashArray = Array.make 5 slisthash

let stringTemp = ""
let int_num =ref 0
let flo_num =ref 0
let str_num =ref 0
let intarr_num =ref 0
let func_num = ref 3
let slist_num = ref 0
let shash_num = ref 1
let slhash_num = ref 0
let addone = fun num -> num+1
(*let intAdd = fun id int_num ->if (IntMap.mem id intmap)then( raise (Failure("redeclared
variable")))
                    else(let intmap = IntMap.add id int_num intmap in intmap)*)
(*let intAdd id int_num = if (IntMap.mem id intmap)then( raise (Failure("redeclared variable")))
                    else(let intmap=IntMap.add id int_num intmap )*)

let assignment = fun id expr -> if (IntMap.mem id intmap)then(intArray.(IntMap.find id intmap)
<- expr;expr )
                           else raise (Failure("undeclared int variable"))

let first = fun (x,y,z,s,p,pr) -> x
let second = fun (x,y,z,s,p,pr) -> y
let third = fun (x,y,z,s,p,pr) -> z
let four = fun (x,y,z,s,p,pr) -> s
let five = fun (x,y,z,s,p,pr)->p
let six = fun (x,y,z,s,p,pr)->pr

let execList l f = List.hd (List.rev (List.map f l) )
(*let execList l f m = List.fold_left f m l*)
let print_int = fun s ->let a = print_endline(string_of_int (List.hd s)) in List.hd s
let print_str = fun s ->let a = print_endline strArray.(List.hd s) in 11

let add = fun s ->let a =print_endline(string_of_int( List.hd s + List.hd (List.tl s))) in 2

let callnew = fun index varible -> match index with
   1 -> print_int varible
|   2 -> print_str varible

let getbody funclist = List.map ( fun func->func.body ) funclist

let varSlhash slhmap stmtl =
let slhashAdd = fun slhmap id slhash_num ->if (SlHashMap.mem id slhmap)then( raise
(Failure("redeclared variable")))
                    else(let slhmap = SlHashMap.add id slhash_num slhmap in slhmap )
in
let rec exprid slhmap = function
 SLcre(id) ->
   ((ignore (slhash_num:=!slhash_num+1);slhash_num.contents), (slhashAdd slhmap id
slhash_num.contents))
| _ -> (0,slhmap)
in
    let rec stmtid slhmap= function
            Expr(e) ->
          exprid slhmap e
```

```
      | Block(b) ->
              (1,(List.fold_left (fun slhmap stmt -> snd(stmtid slhmap stmt) ) slhmap b))
        | _ -> (0,slhmap)
in stmtid slhmap stmtl

let sndvarSlhash slhmap stmtl = snd(varSlhash slhmap stmtl)

let vSlhash slhmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun slhmap stmtl -> sndvarSlhash slhmap stmtl) slhmap stmtlist
in a

let varShash hmap stmtl =
let shashAdd = fun hmap id shash_num ->if (HashMap.mem id hmap)then( raise
(Failure("redeclared variable")))
                    else(let hmap = HashMap.add id shash_num hmap in hmap )
in
let rec exprid hmap = function
 Scre(id) ->
   ((ignore (shash_num:=!shash_num+1);shash_num.contents), (shashAdd hmap id
shash_num.contents))
| _ -> (0,hmap)
in
    let rec stmtid hmap= function
             Expr(e) ->
           exprid hmap e
      | Block(b) ->
              (1,(List.fold_left (fun hmap stmt -> snd(stmtid hmap stmt) ) hmap b))
        | _ -> (0,hmap)
in stmtid hmap stmtl

let sndvarShash hmap stmtl = snd(varShash hmap stmtl)

let vShash hmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun hmap stmtl -> sndvarShash hmap stmtl) hmap stmtlist
in a
```

```
let varList slistmap stmtl =
let listAdd = fun slistmap id slist_num ->if (StringListMap.mem id slistmap)then( raise
(Failure("redeclared variable")))
                    else(let slistmap = StringListMap.add id slist_num slistmap in slistmap )
in
let rec exprid slistmap = function
 StringList(id) ->
   ((ignore (slist_num:=!slist_num+1);slist_num.contents), (listAdd slistmap id
slist_num.contents))
| _ -> (0,slistmap)
in
    let rec stmtid slistmap= function
            Expr(e) ->
          exprid slistmap e
      | Block(b) ->
             (1,(List.fold_left (fun slistmap stmt -> snd(stmtid slistmap stmt) ) slistmap b))
        | _ -> (0,slistmap)
in stmtid slistmap stmtl

let sndvarList slistmap stmtl = snd(varList slistmap stmtl)

let vList slistmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun slistmap stmtl -> sndvarList slistmap stmtl) slistmap stmtlist
in a

let varint intmap stmtl =
let intAdd = fun intmap id int_num ->if (IntMap.mem id intmap)then( raise (Failure("redeclared
variable")))
                    else(let intmap = IntMap.add id int_num intmap in intmap )
in
let rec exprid intmap = function
 Int(id) ->
   ((ignore (int_num:=!int_num+1);int_num.contents), (intAdd intmap id int_num.contents))
| _ -> (0,intmap)
in
    let rec stmtid intmap= function
            Expr(e) ->
          exprid intmap e
      | Block(b) ->
              (1,(List.fold_left (fun intmap stmt -> snd(stmtid intmap stmt) ) intmap b))
        | _ -> (0,intmap)
in stmtid intmap stmtl

let sndvarint intmap stmtl = snd(varint intmap stmtl)

let vint intmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun intmap stmtl -> sndvarint intmap stmtl) intmap stmtlist
in a

let varflo flomap stmtl =
let floAdd = fun flomap id flo_num ->if (FloMap.mem id flomap)then( raise (Failure("redeclared
variable")))
                    else(let flomap = FloMap.add id flo_num flomap in flomap )
in
let rec exprid flomap = function
 Flo(id) ->
   ((ignore (flo_num:=!flo_num+1);flo_num.contents), (floAdd flomap id flo_num.contents))
| _ -> (0,flomap)
in
    let rec stmtid flomap= function
            Expr(e) ->
          exprid flomap e
      | Block(b) ->
              (1,(List.fold_left (fun flomap stmt -> snd(stmtid flomap stmt) ) flomap b))
        | _ -> (0,flomap)
in stmtid flomap stmtl

let sndvarflo flomap stmtl = snd(varflo flomap stmtl)

let vflo flomap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun flomap stmtl -> sndvarflo flomap stmtl) flomap stmtlist
in a

let varstr strmap stmt =
let strAdd = fun strmap id str_num ->if (StringMap.mem id strmap)then( raise
(Failure("redeclared variable")))
                    else(let strmap = StringMap.add id str_num strmap in strmap )
in
let rec exprid strmap = function
 Stringv(id) ->
   ((ignore (str_num:=!str_num+1);str_num.contents), (strAdd strmap id str_num.contents))
| _ -> (0,strmap)
in
    let rec stmtid strmap= function
            Expr(e) ->
          exprid strmap e
      | Block(b) ->
             (1,(List.fold_left (fun strmap stmt -> snd(stmtid strmap stmt) ) strmap b))
         | _ -> (0,strmap)
in stmtid strmap stmt

let sndvarstr strmap stmtl = snd(varstr strmap stmtl)

let vstr intmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun strmap stmtl -> sndvarstr strmap stmtl) strmap stmtlist
in a

let intarr intarrmap stmt =
let intarrAdd = fun intarrmap id intarr_num ->if (IntarrMap.mem id intarrmap)then( raise
(Failure("redeclared variable")))
                    else(let intarrmap = StringMap.add id intarr_num intarrmap in intarrmap )
in
let rec exprid intarrmap = function
 IntArray(id,n) ->
   ((ignore (intarr_num:=!intarr_num+1);intarr_num.contents),
((array_int.(intarr_num.contents)<-(Array.make n 0));intarrAdd intarrmap id
intarr_num.contents))
| _ -> (0,intarrmap)
in
    let rec stmtid intarrmap= function
            Expr(e) ->
          exprid intarrmap e
      | Block(b) ->
            (1,(List.fold_left (fun intarrmap stmt -> snd(stmtid intarrmap stmt) ) intarrmap b))
        | _ -> (0,intarrmap)
in stmtid intarrmap stmt

let sndintarr intarrmap stmtl = snd(intarr intarrmap stmtl)

let iarr intarrmap prog=
let stmtlist = getbody prog
in
let a = List.fold_left (fun intarrmap stmtl -> sndintarr intarrmap stmtl) intarrmap stmtlist
in a

let funcdecl prog =
  let funcmap = List.fold_left
      (fun funcs fdecl -> NameMap.add fdecl.fname fdecl funcs)
      funcmap prog
  in funcmap

let stringList = [""]

let traslate stringTemp stringList intmap strmap intarrmap funcmap flomap slistmap hmap
slhmap prog =

let rec call statement=
let rec expr = function
   Id(s) -> ((if(IntMap.mem s intmap) then (intArray.(IntMap.find s intmap)
,stringTemp,0.0,stringList,point,pointArray)
            else if(StringMap.mem s strmap) then (0,strArray.(StringMap.find s
strmap),0.0,stringList,point,pointArray)
            else if(FloMap.mem s flomap) then (0,stringTemp,floArray.(FloMap.find s
flomap),stringList,point,pointArray)
            else if(StringListMap.mem s slistmap) then
(0,stringTemp,0.0,(strlistArray.(StringListMap.find s slistmap)),point,pointArray)
            else raise (Failure("undeclared inttt variable"))))

 | Print(s)->((if(IntMap.mem s intmap) then
(ignore(print_endline(string_of_int(intArray.(IntMap.find s intmap))));(intArray.(IntMap.find s
intmap) ,stringTemp,0.0,stringList,point,pointArray))
            else if(StringMap.mem s strmap) then (ignore(print_endline(strArray.(StringMap.find s
strmap))); (0,strArray.(StringMap.find s strmap),0.0,stringList,point,pointArray))
             else if(FloMap.mem s flomap)
then(ignore(print_endline(string_of_float(floArray.(FloMap.find s flomap))));
(0,stringTemp,floArray.(FloMap.find s flomap),stringList,point,pointArray))
             else if(StringListMap.mem s slistmap) then(ignore(List.map (fun s ->
ignore(print_endline s);s) strlistArray.(StringListMap.find s slistmap));
(0,stringTemp,0.0,(strlistArray.(StringListMap.find s slistmap)),point,pointArray))
             else raise (Failure("undeclared inttt variable"))))
 | Intersect(a1,a2)->
   let s1 = second(expr a1) and s2 = second(expr a2) in
     (*call the intersection here using s1,s2 as arguments*)
      let inters = find_intersection s1 s2 in
     (0,inters,0.0,stringList,point,pointArray)
 | Strcmp(e1,e2) ->
   let s1=second (expr e1) and s2=second(expr e2) in
   let a = if(s1=s2)then 1 else 0 in
   (a,stringTemp,0.0,stringList,point,pointArray)
 | GetDistance(a1,a2)->
   let s1 = second(expr a1) and s2 = second(expr a2) in
   let distance = get_distance s1 s2 in
   (0,stringTemp,distance,stringList,point,pointArray)
 | Int (id) ->
     (0,stringTemp,0.0,stringList,point,pointArray)
     (*((ignore (int_num:=!int_num+1);int_num.contents), (intAdd id int_num.contents))*)
 | ObjArray(id,index,t) ->(*inters[i].name*)
   let i=first(expr index) in
   let a = strArray.(i) in
   if(id="pathArray") then
   (match t with
    "name"->(1,pathArray.(i)#get_name,0.0,stringList,point,pointArray)
   |"pathType" -> (2,pathArray.(i)#get_pathType,0.0,stringList,point,pointArray)
   |"numberOfPoints" ->
(pathArray.(i)#get_numberOfPoints,stringTemp,0.0,stringList,point,pointArray)
   |"points" ->(3,stringTemp,0.0,stringList,point,pathArray.(i)#get_points)
   )
   else if(id="objArray") then
   (match t with
    "name" ->(1,objArray.(i)#get_name,0.0,stringList,point,pointArray)
   |"objectType"-> (2,objArray.(i)#get_objectType,0.0,stringList,point,pointArray)
   |"speed" ->(objArray.(i)#get_speed,stringTemp,0.0,stringList,point,pointArray)
```

```
  |"length"->(Array.length objArray,stringTemp,0.0,stringList,point,pointArray)
  )
  else if(id="intersArray") then
  (match t with
   "name" ->(1,intersArray.(i)#get_name,0.0,stringList,point,pointArray)
  |"weight" ->(1,stringTemp,intersArray.(i)#get_weight,stringList,point,pointArray)
  |"path1"->(2,intersArray.(i)#get_path_one,0.0,stringList,point,pointArray)
  |"path2"->(2,intersArray.(i)#get_path_two,0.0,stringList,point,pointArray)
  |"point" ->(2,stringTemp,0.0,stringList,intersArray.(i)#get_point,pointArray)
  )
  else(3,stringTemp,0.0,stringList,point,pointArray)
| Objpoint(id1,index,id2,t)-> (*intersArray.point.latitude;*)
  let i = index in
  if(id1="intersArray") then (match t with
   "latitude" -> (3,stringTemp,intersArray.(i)#get_point#get_latitude,stringList,point,pointArray)
    |"longitude"-
>(3,stringTemp,intersArray.(i)#get_point#get_longitude,stringList,point,pointArray))
  else(2,stringTemp(*objArray.(i)#get_pathType *),0.0,stringList,point,pointArray)
| Onedot(id,t) -> (*inters.length*)
  if(id="pathArray") then (match t with
   "length" -> (Array.length pathArray,stringTemp,0.0,stringList,point,pointArray))
  else if (id="intersArray")then (match t with
   "length" -> (Array.length intersArray,stringTemp,0.0,stringList,point,pointArray))
  else if (id="objArray")then (match t with
   "length" -> (Array.length objArray,stringTemp,0.0,stringList,point,pointArray))
  else if (find_index_objectArray id != -1) then (match t with
   "name" -> (2,objArray.(find_index_objectArray id)#get_name,0.0,stringList,point,pointArray)
    | "objectType" -> (2,objArray.(find_index_objectArray
id)#get_objectType,0.0,stringList,point,pointArray)
    | "point" -> (2,stringTemp,0.0,stringList,objArray.(find_index_objectArray
id)#get_point.(0),pointArray)
    | "speed" -> (objArray.(find_index_objectArray
id)#get_speed,stringTemp,0.0,stringList,point,pointArray))
  else (3,stringTemp,0.0,stringList,point,pointArray)
| Twodot(id1,i1,id2,i2,t)-> (*pathArray[i].points[j].latitude*)
  let index1 = first(expr i1) and index2 = first(expr i2) in
  if(id1="pathArray") then (match t with
   "latitude" ->
(3,stringTemp,pathArray.(index1)#get_points.(index2)#get_latitude,stringList,point,pointArray)
    |"longitude"-
```

```
>(3,stringTemp,pathArray.(index1)#get_points.(index2)#get_longitude,stringList,point,pointArr
ay))
   else if (id1="objArray")then (match t with
    "latitude" ->
(3,stringTemp,objArray.(index1)#get_point.(index2)#get_latitude,stringList,point,pointArray)
     |"longitude"-
>(3,stringTemp,objArray.(index1)#get_point.(index2)#get_longitude,stringList,point,pointArray
))
   else (3,stringTemp,0.0,stringList,point,pointArray)
 | Flo (id) ->
    (0,stringTemp,0.0,stringList,point,pointArray)
 | Stringv(i) ->
    (1,stringTemp,0.0,stringList,point,pointArray)
 | Scre(id)  ->
    (1,stringTemp,0.0,stringList,point,pointArray)
 | SLcre(id) ->
    (1,stringTemp,0.0,stringList,point,pointArray)
 | Sadd(id,s1,s2) ->
ignore(
    let a2=expr s2 in
    if (HashMap.mem id hmap)
    then(StringHash.add hashArray.(HashMap.find id hmap) s1 (second a2))
    else raise (Failure("undeclared hashmap variable"))
    );
    (66,(StringHash.find hashArray.(HashMap.find id hmap) s1),0.0,stringList,point,pointArray)
 | Smem(id,s1) ->
   let a =
   if(HashMap.mem id hmap)then(if(StringHash.mem hashArray.(HashMap.find id hmap)
s1)then 1 else 0)
   else raise (Failure("undeclared hashmap variable")) in
   (a,stringTemp,0.0,stringList,point,pointArray)
 | Sfind(id,s1) ->
   let a =
   if (HashMap.mem id hmap)
    then(if(StringHash.mem hashArray.(HashMap.find id hmap) s1)then(StringHash.find
hashArray.(HashMap.find id hmap) s1 )else raise (Failure("member not found")))
    else raise (Failure("undeclared hashmap variable")) in
   (0,a,0.0,stringList,point,pointArray)
 | SLadd(id,s1,s2) ->
ignore(
    let a2=expr s2 in
    if (SlHashMap.mem id slhmap)
    then(StringHash.add slhashArray.(SlHashMap.find id slhmap) s1 (four a2))
    else raise (Failure("undeclared slhashmap variable"))
    );
    (66,stringTemp,0.0,(StringHash.find slhashArray.(SlHashMap.find id slhmap)
s1),point,pointArray)
 | SLmem(id,s1) ->
   let a =
   if(SlHashMap.mem id slhmap)then(if(StringHash.mem slhashArray.(SlHashMap.find id
slhmap) s1)then 1 else 0)
   else raise (Failure("undeclared hashmap variable")) in
   (a,stringTemp,0.0,stringList,point,pointArray)
 | SLfind(id,s1) ->
   let a =
   if (SlHashMap.mem id slhmap)
    then(if(StringHash.mem slhashArray.(SlHashMap.find id slhmap) s1)then(StringHash.find
slhashArray.(SlHashMap.find id slhmap) s1 )else raise (Failure("member not found")))
    else raise (Failure("undeclared hashmap variable")) in
   (0,stringTemp,0.0,a,point,pointArray)
 | Slist(slist) ->
    let s= List.map (fun s-> String.sub s 1 ((String.length s)-2)) slist in
    let l = List.rev s in
    (77,stringTemp,0.0,l,point,pointArray)
 | StringList(id)->
    (88,stringTemp,0.0,stringList,point,pointArray)
 | StringListAdd(id,s1)->
   let s = second(expr s1) in
   let a =
   if(StringListMap.mem id slistmap) then (s::strlistArray.(StringListMap.find id slistmap))
   else raise (Failure("undeclared StringList ")) in
   ignore(strlistArray.(StringListMap.find id slistmap) <-
a);(67,stringTemp,0.0,strlistArray.(StringListMap.find id slistmap),point,pointArray)
 | Return(e) -> expr e
 | Not(e) ->
   ((let int_bool = function
      1->true
     | 0->false
   in
   let a=first(expr e) in
```

```
    if (not (int_bool a)) then 1 else 0
   ),stringTemp,0.0,stringList,point,pointArray)
 | IntArray(id,n) ->
    (2,stringTemp,0.0,stringList,point,pointArray)
 | Char (c) -> ((int_of_char c),stringTemp,0.0,stringList,point,pointArray)
 | Str (s) -> (1,(String.sub s 1 ((String.length s)-2)),0.0,stringList,point,pointArray)
(* | Call (fname, actuals) ->
    ((let v1 = FunctionMap.find fname fmap and v2 =List.map (fun a -> fst(expr a)) actuals in
    call v1 v2),stringTemp)*)
 | Call (fname, actuals) ->
    if(NameMap.mem fname funcmap) then

    let actual=List.map expr actuals in
      let rec varAssign = function
      ([],[]) -> 0
     | (v1::tl1,[]) -> raise (Failure("variable not match"))
     | ([],v2::tl2) -> raise (Failure("variable not match"))
     | (v1::tl1,v2::tl2) -> if (IntMap.mem v1 intmap)then((intArray.(IntMap.find v1 intmap) <-
(first(v2)));varAssign (tl1,tl2))
                else if (StringMap.mem v1 strmap) then ((strArray.(StringMap.find v1 strmap) <-
(second(v2)));varAssign (tl1,tl2))
                else if (FloMap.mem v1 flomap) then ((floArray.(FloMap.find v1 flomap) <-
(third(v2)));varAssign (tl1,tl2))
                        else raise (Failure("undeclared variable of the function"))
     in
       ignore(varAssign ((NameMap.find fname funcmap).formals,actual));call(NameMap.find
fname funcmap).body

    else raise (Failure("function is not defined"))
    (*print_endline (fname);FunctionMap.find fname fmap*)
 (*| Seq (expr1, expr2) ->
    ignore(expr expr1); expr expr2*)
 | Lit(x) -> (x,stringTemp,0.0,stringList,point,pointArray)
 | Float(x) -> (*print_endline (
string_of_float(x));*)(99,stringTemp,x,stringList,point,pointArray)
 | Assign(id,e) -> if (IntMap.mem id intmap)then((intArray.(IntMap.find id intmap) <- (first(expr
e));(first(expr e))),stringTemp,0.0 ,stringList,point,pointArray)
                else if (StringMap.mem id strmap) then ((strArray.(StringMap.find id strmap) <-
(second(expr e));(first(expr e))),second(expr e),0.0 ,stringList,point,pointArray)
                else if (FloMap.mem id flomap)then((floArray.(FloMap.find id flomap) <-
(third(expr e));(first(expr e))),stringTemp,(third(expr e)) ,stringList,point,pointArray)
                else if (StringListMap.mem id slistmap)then((strlistArray.(StringListMap.find id
slistmap) <- (four(expr e));(first(expr e))),stringTemp,(third(expr e))
,strlistArray.(StringListMap.find id slistmap),point,pointArray)
                          else raise (Failure("undeclared int variable"))
 | ArrAssign(id,n,e) -> if (IntarrMap.mem id intarrmap) then (((array_int.(IntarrMap.find id
intarrmap).(n) <- first(expr e));first(expr e)),second (expr e),0.0,stringList,point,pointArray)
                          else raise (Failure("undeclared int variable"))
 | Binop(e1, op, e2) ->
     ((let v1 = first(expr e1) and v2 = first(expr e2) in
      let int_bool = function
        1->true
      | 0->false  in
     match op with
    Add -> v1 + v2
     | Sub -> v1 - v2
     | Mul -> v1 * v2
     | Div -> v1 / v2
     | Equal -> if(v1=v2) then 1 else 0
     | Neq -> if(v1!=v2) then 1 else 0
     | Greater -> if(v1>v2) then 1 else 0
     | Less -> if(v1<v2) then 1 else 0
     | Geq -> if(v1>=v2) then 1 else 0
     | Leq -> if(v1<=v2) then 1 else 0
     | And ->
       if((int_bool v1)&&(int_bool v2)) then 1 else 0
     | Or ->
       if((int_bool v1))||((int_bool v2)) then 1 else 0

     ),stringTemp,0.0,stringList,point,pointArray)

 | FBinop(e1, op, e2) ->(0,stringTemp,
     (let v1 = third(expr e1) and v2 = third(expr e2) in
     match op with
    Fadd -> v1 +. v2
     | Fsub -> v1 -. v2
     | Fmul -> v1 *. v2
     | Fdiv -> v1 /. v2),stringList,point,pointArray)
in
 let rec stmt= function
```

```
  Expr(e) ->
    expr  e
  | Block(b) ->
     execList b stmt
  | While(e,s) ->
    let rec loop e s =
     let v = expr e in
      if(first(v)!=0) then ((*print_endline ( string_of_int(first(v)));*)ignore(stmt s);loop e s) else
(99,stringTemp,0.0,stringList,point,pointArray)
        in loop e s
  | If(c,i,e) ->
    let v = expr c in
     if(first(v)!=0) then (stmt i) else (stmt e)

in stmt  statement
in
  if (NameMap.mem "main" funcmap) then call(NameMap.find "main" funcmap).body
  else raise (Failure("cannot find the main function"))

(*let _ =
let lexbuf = Lexing.from_channel stdin in
 let program = Parser.stmt Scanner.token lexbuf in
 let intmap = snd(traslate intmap program) in intmap*)

(*read in files *)
(* files will be read in like: ./compile filename *)
let filename = Sys.argv.(1)
let stdin = open_in filename
(* do whatever you need to do to get the data here *)

let _ =
 let lexbuf = Lexing.from_channel stdin in
 let program = Parser.program Scanner.token lexbuf in
 (*let intmap = snd(varint intmap program) in*)
 let intmap = vint intmap program in
 (*let strmap = snd(varstr strmap program) in*)
 let strmap = vstr strmap program in
 (*let intarrmap = snd(intarr intarrmap program) in*)
 let intarrmap = iarr intarrmap program in
 let funcmap = funcdecl program in
 let flomap = vflo flomap program in
 let slistmap = vList slistmap program in
 let hmap = vShash hmap program in
 let slhmap = vSlhash slhmap program in
 let result = traslate stringTemp stringList intmap strmap intarrmap funcmap flomap slistmap
hmap slhmap program  in
(* print_endline (string_of_float(third(result)));*)
 (*print_endline ( string_of_int(first(result)));*)
 (* print_endline (second(result));*)
 List.map (fun s -> ignore(print_endline s);s) (four result);
 result;"test"

let () = close_in stdin

(*End Changyu Liu*)
```

Sample MAPMe Program
(*Demo*) <- Denelle Baptiste

```
main() {
~read in xml config file~

stream ("myNeighborhood.xml");

~ create a type for modes of transportation (car, walking, bike, etc)~
  ~calculate ETA using mode of transportation

float speed;
float distance;
float estTimeArr;
String message;
speed = myCar.speed;
distance = getDistance (myDorm.name, Chipotle.name);
estTimeArr = speed/distance;
print ("Estimated Time of Arrival is ");
print(estTimeArr);
```

```
~ create HashTable and arrange objects by tags~
SlistHash hashlist;
StringList libList;
StringList bankList;
StringList restList;
StringList vehicleList;

string tag;
string error;
string name;
int test;
int i;
int j;
int paths;
test= 0;
i=0;
j=0;
paths = pathArray.length;
```

```
while ( i<paths)
{
   tag = pathArray[i].pathType;
   name = pathArray[i].name;
   if (tag == “Library”)
   {
      StringList.add libList name;
      i= (i+1);
   }
   else if (tag == “Restaurant”)
   {
      StringList.add restList name;
      i= (i+1);
   }
   else if (tag ==” Building”)
   {
      StringList.add bldgList name;
      i= (i+1);
   }
   else if (tag ==”Bank”)
   {
      StringList.add bankList name;
      i= (i+1);
   }

else if (tag ==”Vehicle”)
   {
      StringList.add vehicleList name;
      i= (i+1);
   }
   else
   {
      error = “this tag is not recognized”;
      print(error);
      i=(i+1);

   }
```

```
}


~print StringLists that were created~
print(“Library: ”);
print(libList);
print(“Bank: ”);
print(bankList);
print(“Restaurant: ”);
print(restList);
print(“Building: ”);
print(bldgList);

~now add StringLists to Hashlist ~
test = haslisht.find “Library”;
if (test !=0)
{

hashlist.add tag libList;
}

else
{
  print(“Library”);

error = “Tag is already added to Hash List”;
  print(error);
}

test = haslisht.find “Restaurant”;
if (test !=0)
{

hashlist.add tag restList;
}

else
{
  print(“Restaurant”)
  error = “Tag is already added to Hash List”;
  print(error);
}


test = haslisht.find “Bank”;
if (test !=0)
{

hashlist.add “Bank” bankList;
}

else
{
  print(“Bank”)
  error = “Tag is already added to Hash List”;
  print(error);
}
test = haslisht.find “Building”;
if (test !=0)
{

hashlist.add “Building” bldgList;
}

else
{
  print(“Building”)

error = “Tag is already added to Hash List”;

print(error);
}
test = haslisht.find “Vehicle”;
if (test !=0)
{

hashlist.add “Vehicle” vehicleList;
}

else
```

```
{
  print("Building")
  error = "Tag is already added to Hash List";
  print(error);
}
```

```
~find all intersections on map~
int length;
length = pathArray.length();
int i;
int cnt;
i =0;
cnt =0;
size = (length*length);
StringList Intersections;

while (i<length)
{
  int j;

j =0;

~compare every path to every other path~
  while(j<length)
  {

String edgeName;
     String name1;
     String name2;

     if (pathArray[i].name != pathArray[j].name)

{

int m;

int num;

m=0;
```

```
num = intersArray.length;

   while (m<num)
   {
       int test;
       test = 0;
       name1 = pathArray[i].name;
       name2 = pathArray[j].name;
if (intersArray[m].path1 ==name1 || intersArray[m].path2 ==name1)
{
if (intersArray[m].path1 == name2 || intersArray[m].path2 ==name2)
{
test =1;
}
}
m=(m+1);
}
if (test ==0)
{
edgeName = intersection (pathArrays[i].name, pathArray[j].name);
if(edgeName != “null”)

{

            StringList.add Intersection edgeName;


        }

      }

}



  j=(j+1);

}
  i=(i+1);
}
```

```
~create directed graph
StringList paths;
SlistHash dirGraph;
string objName;
int interLen;
int pathLen;
int i;

pathLen = pathArray.length;
length = intersArray.length;
i=0;

~check every path to see if it is part of an intersection~
while (i <pathLen)
{

objName = pathArray[i].name;
```

```
  int j=0;

  while (j<interLen)
  {
     string name1;
     string name2;
     name1 =  intersArray[j].path1;
     name2 = intersArray[j].path2;

     ~if you find an intersection to which this path belongs~
     ~add it to a list~
     if( (objName == name1) )

{
        StringList.add paths name2;

     }
     else if((objName == name2))
     {
        StringList.add paths name1;

     }
     else
     {
        print("This path is not a part of this intersection.");
     }
  }
  ~add the list to the direct Graph hash list~
  dirGraph.add objName paths;
}
```

Sample XML File

(*XML File*) <- Denelle Baptiste

```
<MyObjects>
<myobject description="path" name="Broadway" type="Road" numOfPoints="4"
points="40.804178,-73.966463;40.806046,-73.966463;40.806712,-73.966463; 40.808011,-
73.966463; 40.81048,-73.966463"/>
<myobject description="path" name="Amsterdam" type="Road" numOfPoints="4"
points="40.803057,-73.963888;40.804958, -73.963888;40.806858, -73.963888;40.80819,-
73.963888;40.809246,-73.963888"/>
<myobject description="path" name="110thStreet" type="Road" numOfPoints="2"
points="40.804178,-73.966463; 40.804178,-73.963888"/>
<myobject description="path" name="111thStreet" type="Road" numOfPoints="2"
points="40.804787,-73.966033; 40.804787,-73.963426"/>
<myobject description="path" name="112thStreet" type="Road" numOfPoints="2"
points="40.805396,-73.965476; 40.805396,-73.962965"/>
<myobject description="path" name="113thStreet" type="Road" numOfPoints="2"
points="40.806014,-73.965164; 40.806014,-73.962504"/>
<myobject description="path" name="114thStreet" type="Road" numOfPoints="2"
points="40.806769,-73.965025; 40.806769,-73.962085"/>
<myobject description="path" name="CollegeWalk" type="Walkway" numOfPoints="2"
points="40.808068,-73.965092; 40.806826,-73.962104"/>
<myobject description="path" name="120thStreet" type="Road" numOfPoints="2"
points="40.81048,-73.965092; 40.809246,-73.962183"/>
<myobject description="mapObject" name="Chipotle" type="Restaurant" point="40.804471,-
73.966591" speed="0"/>
<myobject description="mapObject" name="myDorm" type="Building" point="40.806485,-
73.966463" speed="0"/>
<myobject description="mapObject" name="myCar" type="Vehicle" point="null" speed="65"/>
<myobject description="mapObject" name="myTruck" type="Vehicle" point="null"
speed="80"/>
<myobject description="mapObject" name="ButlerLibrary" type="Library" point="40.806554,-
73.96304" speed="0"/>
<myobject description="mapObject" name="FiveGuys" type="Restaurant" point="40.804527,-
73.966591" speed="0"/>
<myobject description="mapObject" name="Citibank" type="Bank" point="40.805023,-
73.966323" speed="0"/>
<myobject description="mapObject" name="BancoPopular" type="Bank" point="40.804641,-
73.966162" speed="0"/>
<myobject description="mapObject" name="UrisLibrary" type="Library" point="40.80897,-
```

73.961141" speed="0"/>
</MyObjects>