pubCrawl
Final Report
COMS W4115

Matt Dean, Sireesh Gururaja, Kevin Mangan, Alden Quimby
mtd2121, skg2123, kmm2256, adq2101

December 20, 2013

Contents

Chapter 1

Introduction

pubCrawl is a distributed systems programming language with a focus on list operations. Given some number of networked slave computers, pubCrawl helps the user automatically split work up among them requiring the user only to specify whether a function should distribute work or not. pubCrawl has been designed from the ground up to make taking advantage of the power of distributed programming as easy as possible. To this end, pubCrawl code is syntactically imperative but function calling is side effect free. This allows developers to get writing pubCrawl code quickly but also makes distributing work across slaves easy. Developers won't have to worry about shared resource locking or race conditions.

1.1 Why is it called pubCrawl?

It's Saturday and the world is your oyster, why stay put the entire night? Since 1915, the term "pub crawl" has been defined by *The New Partridge Dictionary of Slang and Unconventional* as "A drinking session that moves from one licensed premise to the next, and so on." Just as a pub crawl is all about making the most out of the diversity of drinking establishments available, pubCrawl the language is about taking advantage of the many devices the modern developer has available to him or her.

1.2 What Constitutes a Program

A pubCrawl program is simply a list of statements. The syntax of a valid statement is defined in detail in a later section.

Chapter 2

Language Tutorial

2.1 Program Execution

2.1.1 Without distribution

To compile (but not run) a .pc program that doesn't take advantage of distribute, no setup is necessary, simply use the compilation shell script with your .pc file as the only argument.

```
./compile.sh yourProgram.pc
```

To run this program, use the run script. This script calls the compilation script first, so no compilation is required before using it.

```
./run.sh yourProgram.pc
```

2.1.2 With distribution

In order to fully take advantage of pubCrawl, some simple setup is required to make pubCrawl aware of the slave machines you intend to use in your program. On each machine that you would like to use as a slave, run the slave.sh script and pass in the port you would like it to listen on.

```
./slave.sh 1099
```

Then, to run the program, on your master machine use the run script, but this time pass in the program you would like to run as the first argument and then for every slave pass in its IP address and the port. You can add as many slaves as you would like.

```
./compile.sh yourProgram.pc ip_address1 port1 ip_address2 port2 ...
```

2.2 Variables

Variables in pubCrawl do not need to be declared as a specific type. To use a variable, just assign it to the value you want like this:

```
myNumberVariable = 5;

myStringVariable = "hello";
```

2.3 Lists

Variables can also be lists:

```
list1 = [1, 2, 3, 4, 5];
```

You can then access an item from a list using bracket notation. It's of course zero-indexed in true computer science fashion:

```
thirdItem = list1[2];
```

You can also use range notation to access a sublist within a list:

```
list1 = [1, 2, 3, 4, 5];

// This will copy the whole list
list2 = list1[:];

// This will copy a sublist from index 0 to 2
list3 = list1[0:2];
```

You can also add and remove from a list:

```
list1 = [1, 2, 3];

// This will add 19 onto the end of the list
list1.add(19);

// This will remove the value at index 0 from the list
list1.remove(0);
```

2.4 Control Flow

pubCrawl supports 'if/elif/else' statements:

```
if (mybool1) {
  print("inside the if");
}
elseif(mybool2) {
  print("inside the else if");
}
else {
  print("inside the else");
}
```

pubCrawl supports 'for loops':

```
for (i = 0 ; i < 5 ; i = i + 1) {
  print(i);
}
```

pubCrawl also supports 'while loops':

```
while (i > 0) {
  print(i);
  i = i - 1;
}
```

2.5 Functions

This is a function that takes in two parameters:

```
add = (a, b) -> {
  return a + b;
};

fun(2, 3);
```

A function can also have no parameters:

```
() -> {
  return 5;
};
```

Functions are first class in pubCrawl so they are assigned to variables just like a number or a list. They can also be passed as arguments to functions like any other variable. For example, here is a function that takes in a function as the first argument and applies it to the second argument:

```
(f,a) -> {
  return f(a);
};
```

2.6 Printing to stdout

To print to stdout, simply use the built-in 'print' function:

```
print(5);

print("pubCrawl");
```

2.7 Objects

Objects are structures that contain key:value pairs:

```
myObj = {
    value1: 'hey',
    value2: 9
};

// This will access value1 and print 'hey'
print(myObj.value1);
```

2.8 Distributing

Similar to mapping a function to every item on a list, 'distribute' can be used to distribute computation accross other computers:

```
list1 = [1, 2, 3];

fun = (x) -> {
  return x+1;
};
```

```
results = distribute(list1, fun);

print(results[0]); // prints 2
print(results[1]); // prints 3
print(results[2]); // prints 4
```

You can also do more complicated procedures by using lists of lists:

```
gcd = (list) -> {
  a = list[0];
  b = list[1];
  while (a != b) {
    if (a > b) {
      a = a - b;
    }
    else {
      b = b - a;
    }
  }
  return a;
};

listOflists = [[2, 14], [3, 15], [99, 121]];

results = distribute(listOflists, gcd);

print(results[0]); // prints 2
print(results[1]); // prints 3
print(results[2]); // prints 11
```

Chapter 3

Language Reference Manual

3.1 Types

3.1.1 Primitive Types

There are only three primitive types in pubCrawl: **booleans**, **numbers**, and **characters**.

Booleans

A boolean in pubCrawl is defined by the `true` and `false` keywords. Booleans are considered their own type, meaning that an expression that uses a boolean operator and a non-boolean variable will cause an error. For example, !3 will cause an error.

Numbers

A **number** in pubCrawl is a 64 bit piece of information representing values between 4.9E-324 and 1.7976931348623157E308. This is similar to the double class commonly found in other languages. Because there is no "int" type and all numbers in pubCrawl can have a fractional part, when using a number in a situation where having a fractional part would be inappropriate – such as when accessing a specific index of a collection – the fractional part will be ignored.

Characters

Much like in the C programming language, a string is not a primitive type in pubCrawl but rather a collection of characters. This allows manipulating strings to take advantage of the powerful and easy to use collection syntax and functionality found in pubCrawl. Luckily and in contrast to C, using the character primitive type will feel much like using a string in other programming languages such as Java due to pubCrawl's support of string literals in addition to character literals. String literals translate directly into character collections. To simplify discussion, character collections will be referred to as strings. All strings and characters in pubcrawl are encoded via the utf-16 standard.

3.1.2 Non-Primitive Types

Collections

pubCrawl has been designed to make creating and manipulating collections as simple as possible. A collection is a list of items that maintains the order with which it was created much like an array. Items in a collection may be of any type, although the type must remain consistent through the life of the collection. For example, attempting to put a number at the end of a collection that contains objects is not allowed.

Objects

Objects in pubCrawl are built using an adapted JSON syntax. Json.org writes that JSON is, "Easy for humans to read and write" but also, "Easy for machines to parse and generate." JSON notation has become the data object representation standard for networking applications and noSQL databases. pubCrawl's JSON syntax is very similar to javascript's.

Functions

In pubCrawl, functions are first class objects. They can be assigned to variables, and passed as arguments to other functions.

3.2 Type Inference

Data in pubCrawl is expressed using a finite and well defined set of data types. However when writing a pubCrawl program, it is not necessary to explicitly declare types. To be more specific, pubCrawl is *statically typed* and *type inferred*, as opposed to *dynamically typed*. The following section outlines the type inference algorithm used for pubCrawl, which is an implementation of the Hindley–Milner type system.

3.2.1 Types

The pubCrawl type inference algorithm defines 8 types.

```
type t =
  TVar of string
  TFunc of t list * t
  TList of t
  TObj of (string * t) list
  TObjAccess of string * t
  TNum
  TChar
  TBool
```

These types all align with the language types, except for TObjAccess. This special type is used to allow functions to remain as generic as possible, for example the follow function can accept ANY object that has a property 'name':

```
myFunc = x -> { return x.name; };
```

3.2.2 Inference

Algorithm 1 Type Inference

```
1: procedure InferTypes(ast)
2:     sast ← AnnotateAst(ast)
3:     constraints ← CollectConstraints(sast)
4:     substitutions ← Unify(constraints)
5:     sast ← ApplySubstitutions(sast, substitutions)
6:     return sast
7: end procedure

8: procedure AnnotateAst(ast)
9:     Walk the AST as follows:
10:    For FuncCreate, make new scope.
11:    For ObjCreate, check unique properties, and check against keywords.
12:    For Assign, check if ID exists in scope, and check against keywords.
13:    For ForLoop, check ending Assign for existing ID.
14:    For everything, create a new TVar type.
15: end procedure

16: procedure CollectConstraints(sast)
17:    walk the SAST, maintaining list of constraints at each node
18:    for literals, add specific constraints by type
19:    for all expressions, add constraint for value of entire expression
20:    for object access, constraint includes property name and type of property
21:    for object create, contraint includes all property names and types
22:    function constraints include parameter types as well as return type
23:    binary operations restrict both member types
24:    list expressions contraint list type and element types to match
25:    add control flow constraints (i.e. boolean in while loop)
26: end procedure

27: procedure ApplySubstitutions(sast, subs)
28:    walk the SAST and at each node
29:    for each substitution A → B in subs
30:    if the node type contains A, replace it with B
31: end procedure
```

Algorithm 2 Unify Type Constraints

```
1: procedure UNIFY(constraints)
2:     subs ← [ ]
3:     for all (type1, type2) ∈ constraints do
4:         type1 ← APPLY(subs, type1)
5:         type2 ← APPLY(subs, type2)
6:         subs ← subs @ UNIFYONE(type1, type2)
7:     end for
8:     return subs
9: end procedure

10: procedure UNIFYONE(type1, type2)
11:     match (type1, type2) with
12:         TVar(x), TVar(y) →
13:             [(x, y)]
14:         TFunc(p1, x), TFunc(p2, y) →
15:             pairs ← ZIP(p1, p2)
16:             UNIFY((x, y) :: pairs)
17:         TObj(p1), TObj(p2) →
18:             REQUIRESAMEPROPNAMES(p1,p2)
19:             pairs ← ZIP(p1, p2)
20:             UNIFY((x, y) :: pairs)
21:         TList(x), TList(y) | TObjAccess(_, x), TObjAccess(_, y) →
22:             UNIFYONE(x, y)
23:         TNum, TNum | TBool, TBool | TChar, TChar →
24:             [ ]
25:         TVar(x), z →
26:             UNIFYONE(x, z)
27:         TObj(props), TObjAccess(name, y) →
28:             found ← FINDPROPBYNAME(props, name)
29:             UNIFYONE(y, found)
30:         TObjAccess(_, y), z →
31:             UNIFYONE(y, z)
32:     end match
33: end procedure

34: procedure APPLY(subs, type)
35:     for each substitution A → B in subs
36:     if type contains A, replace it with B
37: end procedure
```

3.3 Lexical Conventions

3.3.1 Identifiers

An **identifier** is a sequence of letters, digits, or underscores. The first character must be a letter; the underscore is not considered a letter. Upper and lower case letters are different.

3.3.2 Keywords

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

if	elif
else	for
while	return
true	false
distribute	read
print	download
rec	numToString
numFromString	

3.3.3 Literals

Literals or **constants** are the values written in a conventional form whose value is obvious. In contrast to variables, literals do not change in value.

A **number literal**, or **number constant**, consists of one or more digits constituting the integer part of the number, and then an optional decimal point with zero or more digits consituting the fraction. If the integer part of the number will contain more than one digit, a zero cannot be the first. These are all well formed number literals in pubCrawl:

42	42.
0.42	42.4000
420.024	0.42

These are not:

00	.68
17e-6	-3
092.5	6.7.8

Note that negative number literals are not supported. A negative number literal can be achieved in affect by subtracting the positive version of the number from zero.

A **boolean literal** represents boolean values for true and false. The two possible values are represented by `true` and `false`.

A **character literal** consists of a single quotation mark, followed by a single character, ended with a single quotation mark. The only exception to this rule are the following character literals that, although representing one character, are communicated to the compiler via two characters.

single quote	\’
double quote	\”
newline	\n
horizontal tab	\t
carriage return	\r

A **string literal** consists of a double quotation mark, zero or more characters, and finally another double quotation mark. A string literal can exist wherever a character collection expression would be valid. As an example, **”Hello world!”** is a valid string literal.

3.3.4 Punctuation

Some characters are used as **punctuators**, which have their own syntactic and semantic significance. Punctuators are not operators or identifiers.

Punctuator	Use	Example
,	List seperator, function parameters	list1 = [1, 2, 3, 4, 5]
[]	List delimiter, list access	list2 = []
()	Conditional parameter delimiter, expression precedence	if(bool)
{}	Statement list delimiter	if(bool) { statements }
:	List access, object property assignment	list3 = list2[0:2]
;	Statement end	mystring = 'ABC';
'	Character literal delimiter	mycharacter = 'a'
"	String literal delimiter	mystring = "Hello!"

3.3.5 Comments

The characters /* introduce a multi-line comment, which terminates with the characters */. The characters // introduce a single line comment. Multi-line comments cannot be nested within multi-line comments, but single line comments can be nested in multi-line comments.

```
// single line comments start
/*
    multi-line comments
*/
/*
    /* this breaks */
    // this, however, works
*/
/* this will break too /* */
```

3.3.6 Operators

An **operator** is a token that specifies an operation on at least one operand, and yields some result.

Operator	Use	Associativity
.	Access	Left
*	Multiplication	Left
/	Division	Left
%	Modulus	Left
+	Addition	Left
-	Subtraction	Left
^	Concatenation	Left
=	Assignment	Non-associative
==	Equal to	Left
!=	Not equal to	Left
<	Less than	Left
>	Greater than	Left
<=	Less than or equal to	Left
>=	Greater than or equal to	Left

The precedence of operators is the following, from greatest to least precedence:

.
* / %
+ -
<><= >=
= !=
^
= :

3.4 Syntax

3.4.1 Program Structure

A program in pubCrawl consists of a sequence of zero or more valid pubCrawl statements:

`statement-list-opt`

3.4.2 Expressions

An expression is a sequence of operators and operands that produce a value and may have side effects. Expressions have a type and a value. The order of evalution of subexpressions, and therefore the order in which side effects take place, is left to right. The following sections describe the various forms of valid pubcrawl expressions. To get started, here are a few examples:

```
// x+y evaluated first, then a+b, then division.
(x + y) / (a + b)

// func2 evalutated first, then func3, then func1
func1(func2(), func3())
```

Operands of expressions must have compatible types.

Constants

The type of a constant depends on its form. It can either be a number, string or boolean, as discussed in Lexical Conventions.

Identifier

An identifier designates a primitive type, collection, object, or function. The type and value of an identifier is determined by this designation. For example:

```
x = 4;
// here x has type number and value 4
y = true;
// here y has type boolean and value true
```

Binary Operators

Binary operators can be used with variables and constants to create complex expressions. A binary operator is of the form:

`expression binary-operator expression`

- Arithmetic operators

  Arithmetic operators include multiplication (*), division (/), modulus (%), addition (+), and subtraction (-). The operands to an arithmetic operator must be numbers. The type of an arithmetic

operator expression is a number and the value is the result of calculating the appropriate arithmetic. For example, the multiplication operator performs multiplication on its operands.

- Relational operators

  Relational operators include less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal to (==), and not equal to (!=). The operands to a relational operator must be numbers. The type of an relational operator expression is a boolean and the value is true if the relation is true. For example, the less than operator has a value of true if the left operand is strictly less than the right operand.

- Logical operators

  Logical operators include AND (&&) and OR (||). The operands to a logical operator must be booleans, and the result of the expression is also a boolean.

- String operators

  The only string operator is concatentation (^). The operands to a concatenation operator must be strings, and the result is a new string that is the concatenation of the left and right operands.

Parenthesized Expressions

Any expression surrounded by parentheses has the same type and value as it would without parentheses. In general, parentheses are used to alter operator precendence.

Function Creation

Function creation is an expression whose type is function and whose value is a reference to the newly created function. Because functions in pubCrawl are first class objects, functions can be declared anywhere having an expression would be appropriate. Because of this, functions must be stored in variables to be accessed later in the program. A function declaration is made clear with the "->" operator. On the left hand side of the "->" is the parameter declaration and on the right hand side is a block. Specifiying the result of a function is done with a return statement and void functions are not allowed. Parameter declaration is surrounded by parentheses and consists of a list of identifiers separated by commas. Optionally when only one parameter is specified, the parenthesis may be omitted. Additionally, inside the block of a function, the keyword "rec" refers to the function itself to allow for recursion. Although the rec keyword is in some sense equivalent to a variable referring to the function, it is not a valid left hand side of an expression and therefore cannot be reassigned. This is a function with no parameters:

```
() -> {
  statement-list-opt
  return-stmnt
}
```

Both of these functions have one parameter:

```
identifier -> {
  statement-list-opt
  return-stmnt
}
(identifier) -> {
  statement-list-opt
  return-stmnt
}
```

This function has n parameters

```
(identifier-1,identifier-2, ... ,identifier-n) -> {
  statement-list-opt
  return-stmnt
}
```

A function can have multiple return statements, however a function's return type must remain consistent over the life of the function. In other words no function can exist that sometimes returns one type, and sometimes returns a different type and such a function would fail to compile.

Function Call

A function call is an expression whose type and value are determined by the return type and value of the function. Calling a function executes the function and blocks program execution until the function is complete. Parameters are expressions that are separated by commas, surrounded by parenthesis and placed after the identifier representing the function. If there are no parameters, the parenthsis are still required for the function call.

```
identifier(expr1,expr2, ..., exprn)
identifier()
```

When a function is called, the expressions passed into the function as parameters are evaluated in left to right order:

```
identifier(expr1,expr2)
```

In this example, expr1 would be evaluated first. The results of these expressions are then copied by value into the function's scope.

Object Creation

Object creation is an expression that has a type of object and a value of a reference to the object. An object is created as an optional list of properties surrounded by braces:

```
{ properties-opt }
```

The optional list of properties are comma separated property declarations:

```
identifier: expression
```

The identifiers in the property list must be unique for a specific object. Each property declaration assigns the value and type of the expression to the specified identifier within the context of the created object. Expressions in the property list will be evaluated in the order in which they appear.

Object Access

The properties of an object can be accessed via dot notation. The type and value of an object access expression are the type and value of the accessed property. To access via bracket notation, use an identifier to represent the property:

```
object_expr.identifier
```

Collection Creation

Collection creation is an expression that has a type of colletion and a value of a reference to the collection. A collection is created as an optional list of expressions surrounded by brackets:

```
[ expressions-opt ]
```

The expressions in the optional list are comma separated and can be of any type, but must all have the same type.

Collection Access

Items in a collection are accessed using bracket notation. The expressions inside brackets must resolve to numbers. The resulting type will either be another collection, or the type of an item at the specified index. To access a particular item by it's index in a collection:

```
[ expression ]
```

To access a subset of the collection:

```
[ expression1-opt : expression2-opt ]
```

If expression1-opt is omitted, the subset will go from the beginning of the collection to the index of expression2-opt. If expression2-opt is ommited, the subset will go from the index of expression1-opt to the end of the collection. If both expressions are omitted, the collection will be copied.

3.4.3 Statements

A statement in pubCrawl does not produce a value and does not have a type, but can produce side effects. Statements are executed in the order in which they appear. An expression is not a valid statement in pubCrawl.

Assignment

Assignment statements consist of a modifiable lvalue and an expression. An lvalue is either an identifier, an object access expression, or a collection access expression that is not a subset. When an assignment statement is executed, the expression is evaluated and the result is assigned to the lvalue.

```
lvalue = expression;
```

Function Call

A function call statement executes the specified function and specifically ignores the return value of the function. A function call statement is a function call expression followed by a semi-colon.

```
function_call;
```

Because function calling is also an expression, the return value of a function call can be captured with an assignment statement.

Selection Statements

A selection statement executes a set of statments based on the value of a specific expression.

- If-elif-else

  An if-elif-else statement takes multiple boolean expression and selects one statement list to execute corresponding to the first true expression.

  ```
  if (expression-1)
  {
    statement-1-list
  }
  elif (expression-2)
  {
    statement-2-list
  }
  ...
  elif (expression-n)
  {
    statement-n-list
  ```

```
}
else
{
  statement-else-list
}
```

In the above example, if expression-1 is true, only statement-1-list will be executed, and all remaining expressions will not be evaluated. If expression-1 is false, expression-2 will be evaluated. If expression-2 is true, statement-2-list will be executed and all remaining expressions will not be evaluated. If all expressions evaluate to false, statemement-else-list will be executed. Note that all elif clauses are optional, and the else clause is optional, but must come after all elif clauses.

Iteration Statements

- While loop

  The while statement evaluates an expression before each execution of the body. The expression must by of type boolean, and the value of the expression typically changes in the body of the loop. If the expression is true, the loop body is executed. If the expression is false, the while statement terminates. The while statement has the following syntax:

```
while ( expression )
{
  statement-list
}
```

- For loop

  The for statement evaluates two assignments and one boolean expression, and executes the body until the expression evaluates to false.

```
for ( assignment1-opt ; expression-opt ; assignment2-opt )
{
  statement-list
}
```

  The for statement is almost (see below) equivalent to the following while loop:

```
assignment1-opt;
while ( expression-opt )
{
  statement-list
  assignment2-opt;
}
```

  A for statement executes as follows: assignment1-opt is evaluated once before any iteration. expression-opt is a boolean expression evaluated before each loop iteration. If this expression is true, the loop body is executed. If this expression is false, the for statement terminates. assignment2-opt is evaluated after each iteration. However, a new variable cannot be declared in assignment2-opt. Consider the following:

```
// valid for loop, even if i already exists
for ( i = 0 ; i < 100 ; i = i + 10 )
{
  ...
}

// invalid for loop if k does not exist yet
for ( i = 0 ; i < 100 ; k = i + 10 )
{
```

```
  ...
}
```

The reason for this restriction is scoping. assignment2-opt is evaluated for the first time after one iteration of the loop, so k would not exist for the first iteration and therefore be inaccessible. k would also be inaccessible outside the for loop because of block scoping, discussed in the Scope section. A jump statement inside the loop body will cause the for statement to terminate. If the optional expression is omitted, it will be treated as true, making an infinite loop.

Jump Statements

- Return

  A return statement is specified with the return keyword, followed by an expression and ending with a semi-colon.

  ```
  return expression;
  ```

  The return statement terminates execution of the smallest containing function, or the entire program if there is no containing function, and returns control to the caller with the value and type of it's expression.

3.4.4 Scope

Lexical Scoping with Blocks

A block is a set of statements enclosed by braces. For example, the body of a for loop or an if statement is a block. An identifier appearing within a block is visible within the entire block, but can be modified by an inner block. For example, this will print 7 twice, followed by 'hey' five times:

```
x = 7;
print(x);
for (i = 0; i < 5; i = i + 1)
{
  print(x);
  x = 'hey';
}
print(x);
```

The scope of an identifier begins when it is first assigned, and ends at the end of it's smallest containing block or program. All blocks have access to the List standard libary object and all built in functions.

```
a = 7;
for (i = 0; i < 5; i = i + 1)
{
  x = 'hey';
  // a and x are visible here
}
// only a is visible here
x = 'hey';
// a and x are visible here
```

Function Scope

Functions only have access to the identifiers in their parameter list, identifiers declared within their body, the standard libary object, and built in functions. As a note, this restriction is essential to the pubCrawl language because it makes unintended consequences of the **distribute** function virtually impossible. With this paradigm, no two functions running on separate machines are capable of accessing a shared resource at the same time.

```
x = 4;
y = (a,b,c) ->
{
  x = a + b + c; // this x does not affect outer x
  return x*5;
};
z = y(1,2,3);
/*
  x is still 4 here
  z is 30
*/
```

3.5 Distribution

The special function `distribute` is what enables programmers to leverage the processing poewr of multiple machines. Its use is essentially analogous to the `map` function (see section 6.1) - given a function on the list, it will apply the function to each element in the list, the difference being that the work of applying the function to the list is spread across multiple computers. The `distribute` function will return a list of processed values. Its usage is as follows:

```
results_list = distribute(list, function);
```

3.6 Built-in Functions

pubCrawl provides some simple IO. Standard in and out, file read and write, as well as simple web downloading is all supported. Additionally, collections have built in methods to aid their manipulation.

3.6.1 read and readFile

By using the read functions one can read from either stdin or a file to EOF. read takes no arguments and returns stdin until the first newline as a string. readFile takes one argument, a string representing the name of a file, and also returns a list of strings, one string for each line.

```
// read to EOF
input = read(); // input is a single line
input = readFile("path/to/file.txt"); // input is a collection of lines
```

3.6.2 print and printFile

By using the print functions, one can print to stdout or to a file. print takes one argument of any type containing the content to print to stdout. printFile takes two arguments, the first is the content to print to file, and the second is a string representing the name of a file to print to. Both functions return their first argument, an expression of the input type.

```
// write
print("Hello World"); // stdout
printFile("Hello World", "path/to/file.txt");
```

3.6.3 download

The download function takes one arugment, a string representing a url to download, and returns the webpage as a string.

```
//download
pageContent = download("http://www.google.com");
```

3.6.4 numToString and numFromString

pubCrawl provides built in functions to convert strings to numbers as well as numbers to strings. These functions are essential for taking advantage of user input with your pubCrawl program.

```
//converts number to string
stringFive = numToString(5);
//converts string to number
numberFive = stringToNum("5");
//note that this will not run correctly if string is not a valid number
```

3.7 List Utilities

pubCrawl provides a few built-in functions to aid in list manipulation. Because strings are treated as lists in pubCrawl, these functions can be used for both lists and strings. The standard library is accessable from a special object that exists in all scopes named "List".

3.7.1 List.length

The `length` function takes one argument, a list of any type, and returns a number equal to the amount of elements in the list. This function is especially helpful for iterating through a list in a for loop.

`length = List.length(`*`list);`*

Usage:

```
// 4
length = List.length([1,2,3,4]);
```

3.7.2 List.add

The `add` function takes two arguments, a list of any type and an element of that type, and returns a new list with the element appended to the back of the input list.

`myList = List.add(`*`list , element);`*

Usage:

```
// [1,2,3,4]
myList = List.add([1,2,3], 4);
```

3.7.3 List.remove

The `remove` function takes two arguments, a list of any type and a number representing the index to remove, and returns a new list with .

`myList = List.add(`*`list , element);`*

Usage:

```
// [9,11]
myList = List.remove([9,10,11], 1);
```

3.7.4 List.map

The `map` function takes two arguments, a list of any type and a function that takes one argument matching the list type and returning any type. `map` applies the function to each member of the list, and returns a new list with the results as elements.

```
resultList = List.map(list , function);
```

Usage:

```
// [2,4,6]
newList = List.map([1,2,3], x -> { return x*2; });
```

3.7.5 List.where

The `where` function takes two arguments, a list of any type and a function that takes one argument matching the list type and returning a boolean. `where` applies the function to each member of the list, and returns a new list containing only elements where the predicate returns `true`.

```
resultList = where(list , boolean_function);
```

Usage:

```
// [2]
newList = List.where([1,2,3], x -> { return x%2 == 0; });
```

3.7.6 List.find

The `find` function takes two arguments, a list of any type and another list of that type to find as a sublist in the first list. `find` returns a number representing the start index of the first occurance of the sublist in the list, or -1 if it was not found.

```
number = find(string, substring);
```

Usage:

```
// 4
idx = List.find("hey dude", "dude");

// -1
idx = List.find([1,2,3,4], [4,5]);
```

3.7.7 List.split

The `split` function takes two arguments, a list of any type an element of that type to split on. `split` returns a list of lists of the input type, after separating the input list by the token.

```
list = split(source, token);
```

Usage:

```
// ["hey","dude"]
newList = List.split("hey dude", ' ');

// [[1,2],[3,4],[5,6]
newList = List.split([1,2,77,3,4,77,5,6], 77);
```

3.7.8 List.range

The `range` function takes two arguments, both numbers, representing the min and max for a desired range. `range` returns a list of all numbers between the min and max, inclusive.

list = range(*number_min, number_max*);

Usage:

```
// [1,2,3,4,5,6,7,8,9,10]
newList = List.range(1, 10);
```

3.7.9 List.populate

The `populate` function takes two arguments, the first is an element of any type and the second is a number. `populate` returns a list of length equal to the second argument where each element in the list is the first argument. This is helpful when doing certain tasks with distribute, for example if you want each slave to do the same task.

list = populate(*number, element*);

Usage:

```
// ["a", "a", "a", "a", "a"]
manyAs = List.populate("a", 5);
```

3.8 Program Execution

3.8.1 Without distribution

To compile (but not run) a .pc program that doesn't take advantage of distribute, no setup is necessary, simply use the compilation shell script with your .pc file as the only argument.

```
./compile.sh yourProgram.pc
```

To run this program, use the run script. This script calls the compilation script first, so no compilation is required before using it.

```
./run.sh yourProgram.pc
```

3.8.2 With distribution

In order to fully take advantage of pubCrawl, some simple setup is required to make pubCrawl aware of the slave machines you intend to use in your program. On each machine that you would like to use as a slave, run the slave.sh script and pass in the port you would like it to listen on.

```
./slave.sh 1099
```

Then, to run the program, on your master machine use the run script, but this time pass in the program you would like to run as the first argument and then for every slave pass in its IP address and the port. You can add as many slaves as you would like.

```
./compile.sh yourProgram.pc ip_address1 port1 ip_address2 port2 ...
```

Chapter 4

Project Plan

Without careful planning and organization this project would surely have been doomed from the start. Luckily, we took the time to create a roadmap for success as well as implemented a number of simple project management systems. This section outlines the tools and techniques we used to make pubCrawl happen.

4.1 Project Processes

4.1.1 Planning

In order to ensure that every member of our team was fully aware of the status of the project as well as current goals, we made sure to meet consistently each week on Sunday or Monday, and sometimes added supplementary meetings on Thursday. During our meetings we took time to check our project timeline to plan what each member should tackle for the upcoming week.

4.1.2 Specification

After creating the first draft of our LRM, it became a living specification for the pubCrawl language. Each time we realized we had underspecified or incorrectly specified something, we made sure to keep the document up to date. As our LRM grew and evolved, so did pubCrawl!

4.1.3 Development

We took a three step process for developing each feature of pubCrawl. First we attempted to create a simple, first draft of the desired feature, second we tested the feature either by printing out results early on in the process or eventually with full integration tests. After testing we identified a few areas of improvement for each feature, and thus our third step was figuring out which of these was doable and implementing any additional awesomeness.

4.1.4 Testing

At a high level, we made sure to allow feedback at all steps in our design process. To do this, we had our main loop take identifiers for how far in the compilation process we wanted the program to run. This allows us to isolate testing the scanner and parser for example by passing in "-a" for AST or just test the type inference by passing "-s" for SAST. This ensure that we maintained testing capabilites before our full integration tests were ready. We discuss our testing process at great lengths in section 6.

4.2 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

4.3 Team Responsibilities

Divide and conquer was essential to making pubCrawl a reality. In order to facilitate splitting up work, we first identified our interfaces between sections. We then made sure to get everyone onboard with git hosted by GitHub. Version control was essential to making sure everyone could work without stepping on each others' toes. With these set in stone, parallelizing was a breeze! We split up our group into two smaller more focused teams:

Front End: Alden, Matt - scanner, parser, type inference

Back End: Kevin, Sireesh - java generation, distributed computing, testing

4.4 Project Timeline

Our timeline was carefully laid out from the start:

Sept 25th Proposal due date

Oct 7th pubCrawl syntax created

Oct 14th Scanner/parser unambiguous and working

Oct 14th Java Code sketched out

Oct 21th Java distribute working

Oct 21th LRM first draft

Oct 28th LRM due date

Nov 11th Architechural design finalized

Nov 25th Compiler works, all tests pass

Dec 9th Final project report

Dec 16th Final project Slides

Dec 20th Final project due date

4.5 Development Environment

The pubCrawl team developed on a variety of environments including mac OS X, Ubuntu, and Windows 7. We used OCaml version 4.00.1, OCamllex, and OCamlyacc for the compiler itself. We used git hosted on github for version control. Lastly, we used bash scripts and makefiles to ease the work of compiling and testing the code.

4.6 Project Log

The following is our project log from github, authored and dated.

```
Sireesh Gururaja    2013-12-17    final commit.
Alden Quimby    2013-12-17    edit demos
Alden Quimby    2013-12-17    merge
Alden Quimby    2013-12-17    last minute changes
Kevin Michael Mangan    2013-12-17    comments
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-17    type inference test should work (needed an out file)
Kevin Michael Mangan    2013-12-17    fixed tostring test output
Alden Quimby    2013-12-17    Merge branch 'master' of https://github.com/mattydoincodt puse/PLT
Alden Quimby    2013-12-17    add list populate
Kevin Michael Mangan    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-17    tostring test
Sireesh Gururaja    2013-12-17    java7 jar for DS
Alden Quimby    2013-12-17    Merge branch 'master' of https://github.com/mattydot pusincode/PLT
Alden Quimby    2013-12-17    modify server
Sireesh Gururaja    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-17    added type inference test
Alden Quimby    2013-12-17    merge
Alden Quimby    2013-12-17    add shell scripts
Kevin Michael Mangan    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-17    chat client works
Sireesh Gururaja    2013-12-17    x-platforming script
Alden Quimby    2013-12-17    Merge branch 'master' of https://github.com/mt pushattydoincode/PLT
Alden Quimby    2013-12-17    swallow stderr
Kevin Michael Mangan    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-17    mergey mergey mergey
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-17    modified final demo files, they work!
Alden Quimby    2013-12-17    fix script
Alden Quimby    2013-12-17    merge
Alden Quimby    2013-12-17    make script better
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-17    compile needed a trailing slash (facepalm)
Kevin Michael Mangan    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-17    reddit demo works
Alden Quimby    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-17    merge
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-17    scanner parser distribute is a keyword
Sireesh Gururaja    2013-12-17    moving writing to IO
Alden Quimby    2013-12-17    set up IO
Alden Quimby    2013-12-17    temp for sireesh
Kevin Michael Mangan    2013-12-17    toString function works
Kevin Michael Mangan    2013-12-17    toString function works
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-17    writer string of object is public, hello world distribute
Alden Quimby    2013-12-17    fileserver starting on its own
Matt Dean    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-17    asdf
Matt Dean    2013-12-17    fileserver improvements serves bytes  now!
Alden Quimby    2013-12-17    ocaml version fail!!!!
Matt Dean    2013-12-17    static file server ooooof
Alden Quimby    2013-12-17    java 6 revert
Matt Dean    2013-12-17    hello world, moved server into bin folder, deleted distribute folder
Sireesh Gururaja    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
```

```
Sireesh Gururaja    2013-12-17    wrong format for port number
Alden Quimby    2013-12-17    fix testall
Alden Quimby    2013-12-17    fix testall to work with distribute
Alden Quimby    2013-12-17    pass rest of args to java
Alden Quimby    2013-12-17    fix client
Sireesh Gururaja    2013-12-17    a stupid
Sireesh Gururaja    2013-12-17    added back ports
Alden Quimby    2013-12-17    distribute tests work!
Alden Quimby    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-17    refactor for submission
Sireesh Gururaja    2013-12-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-17    removed port number entry for server
Alden Quimby    2013-12-17    get scanner parser working
Alden Quimby    2013-12-16    merge
Alden Quimby    2013-12-16    add some demo examples for final
Kevin Michael Mangan    2013-12-16    download all as one string
Kevin Michael Mangan    2013-12-16    download all as one string
Alden Quimby    2013-12-16    read() one line works!
Alden Quimby    2013-12-16    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-16    read() returns one line at a time
Kevin Michael Mangan    2013-12-16    read one line only
Alden Quimby    2013-12-16    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-16    reader
Alden Quimby    2013-12-16    add write file test, will work when kevins works
Alden Quimby    2013-12-16    file io in javagen
Alden Quimby    2013-12-16    delete java folder!
Alden Quimby    2013-12-16    java stlye foreach
Kevin Michael Mangan    2013-12-16    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-16    downloader
Alden Quimby    2013-12-16    move chat client, this is awesome sireesh
Alden Quimby    2013-12-16    Merge branch 'master' of https://github.cot cm/mattydoincode/PLT
Alden Quimby    2013-12-16    extra concat test
Kevin Michael Mangan    2013-12-16    oops forgot to push the downloadtest.pc file
Sireesh Gururaja    2013-12-14    chatting in 12 lines
Sireesh Gururaja    2013-12-14    everything working
Sireesh Gururaja    2013-12-14    again
Matt Dean    2013-12-14    mutual rec fixed
Sireesh Gururaja    2013-12-14    everything working but mutual rec
Sireesh Gururaja    2013-12-14    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-14    before merge
Kevin Michael Mangan    2013-12-13    download works chyeaaaa
Kevin Michael Mangan    2013-12-13    merge
Alden Quimby    2013-12-13    add type ineference test
Kevin Michael Mangan    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    a stupid
Kevin Michael Mangan    2013-12-13    merge
Kevin Michael Mangan    2013-12-13    merge
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    make clean...
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    add newline
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    distributeReordered
Matt Dean    2013-12-13    heyo, mutual recursion!!
Alden Quimby    2013-12-13    test object setting
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
```

```
Sireesh Gururaja    2013-12-13    cleans
Alden Quimby    2013-12-13    Merge branch 'master' of https://github.ctom/mattydoincode/PLT
Alden Quimby    2013-12-13    fix objects, add double recursion;
Kevin Michael Mangan    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-13    test-ops1 works
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    make clean...
Matt Dean    2013-12-13    ops1 works great
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    fib keyword
Alden Quimby    2013-12-13    Merge branch 'master' of https://github.tcom/mattydoincode/PLT
Alden Quimby    2013-12-13    fix split now the sublist is fixed
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    boolean.getbase
Alden Quimby    2013-12-13    fix sublist
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    make clean
Alden Quimby    2013-12-13    weird
Alden Quimby    2013-12-13    javagen sublist bug
Matt Dean    2013-12-13    for loop booleanwq
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    collection3 works
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    rogue h
Alden Quimby    2013-12-13    where now works
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    changed remove2 out
Alden Quimby    2013-12-13    split works now;
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    remove make stuff
Sireesh Gururaja    2013-12-13    ifstmt
Matt Dean    2013-12-13    modified remove2 test
Kevin Michael Mangan    2013-12-13    hashtag java problems
Kevin Michael Mangan    2013-12-13    hashtag java problems
Kevin Michael Mangan    2013-12-13    hashtag java problems
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    map2 works
Sireesh Gururaja    2013-12-13    fixed range1
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-13    downloader.java
Alden Quimby    2013-12-13    fix split tests
Alden Quimby    2013-12-13    fix compile script
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    map2 fix
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    fixed integer access
Alden Quimby    2013-12-13    fix find2 test
Matt Dean    2013-12-13    make all
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    some tests pass, assign bug in javagen
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    merge to fix sublist
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    add1 works!!
Alden Quimby    2013-12-13    fix id second assing
Alden Quimby    2013-12-13    fix some more tests
Alden Quimby    2013-12-13    fix some more tests
Alden Quimby    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
```

```
Alden Quimby    2013-12-13    asdfasdf
Matt Dean    2013-12-13    make all -> make in compilescript
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    fixed List.find()
Alden Quimby    2013-12-13    Merge branch 'master' of https:/t push/github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    pcobj int fix
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-13    changing print to accept anything
Alden Quimby    2013-12-13    allow print to be anything
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    round 1 at fixing the tests
Matt Dean    2013-12-13    working in noco doing last minute script stuff
Matt Dean    2013-12-13    removed clean
Matt Dean    2013-12-13    trying to fix compile script to rerun java compilation
Matt Dean    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    added PCObject casting
Sireesh Gururaja    2013-12-13    removes classes
Matt Dean    2013-12-13    working on fixing list.add('a').add('b')
Alden Quimby    2013-12-13    Merge branch 'master' of ht pushttps://github.com/mattydoincode/PLT
Alden Quimby    2013-12-13    fix keywords, list now std lib
Sireesh Gururaja    2013-12-13    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-13    list utils implemented
Sireesh Gururaja    2013-12-13    fixed a stupid
Sireesh Gururaja    2013-12-13    script now runs the program.
Sireesh Gururaja    2013-12-13    strings no longer allowed to have non-escaped " chars
Kevin Michael Mangan    2013-12-12    commented out test for now
Kevin Michael Mangan    2013-12-12    not really sure how to unit test these in intellij...
Alden Quimby    2013-12-12    add comment for sireesh
Alden Quimby    2013-12-12    Merge branch 'mastert push' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-12    add List.length
Sireesh Gururaja    2013-12-11    distribute keyword
Sireesh Gururaja    2013-12-11    Distribute now multithreaded
Alden Quimby    2013-12-09    add final tex in case writelatex goes down
Alden Quimby    2013-12-09    fix List. functions in analyzer
Sireesh Gururaja    2013-12-06    more notes
Sireesh Gururaja    2013-12-06    added policy files and notes on running
Sireesh Gururaja    2013-12-06    RMI working from the bin directory, but with constraints
Sireesh Gururaja    2013-12-05    fixed " handling
Sireesh Gururaja    2013-12-05    fixed so java now accepts escape chars
Sireesh Gururaja    2013-12-05    fixed a broken test
Sireesh Gururaja    2013-12-05    scanner accepts escapes, strings allow escapes
Alden Quimby    2013-12-03    get listmap working
Alden Quimby    2013-12-02    list in analyzer
Matt Dean    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-02    added test3
Matt Dean    2013-12-02    write system.out fix
Kevin Michael Mangan    2013-12-02    Msdferge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-02    again
Alden Quimby    2013-12-02    Merge branch 'mt pusaster' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-02    bug fix
Matt Dean    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-12-02    merge
Alden Quimby    2013-12-02    changes from meeting
Kevin Michael Mangan    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-02    PCL
Matt Dean    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-12-02    javagen changes to cast for obj and list access
Kevin Michael Mangan    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-02    publicized call method
```

```
Kevin Michael Mangan    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-12-02    stashing
Sireesh Gururaja    2013-12-02    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-12-02    copied stuff back to java for RMI
Matt Dean    2013-12-02    modified make to clean some bin stuff
Sireesh Gururaja    2013-11-27    stopped the compiler from printing the java to stdout
Sireesh Gururaja    2013-11-27    with pc program as arg, compiles to java exec
Sireesh Gururaja    2013-11-27    writeFunc inherits IPCFunction, test2.pc now SFW
Alden Quimby    2013-11-26    add util java
Alden Quimby    2013-11-26    implement iterable correctly
Alden Quimby    2013-11-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-26    new iterator
Matt Dean    2013-11-26    remove make
Matt Dean    2013-11-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-26    character cast
Alden Quimby    2013-11-26    clean up unused files
Matt Dean    2013-11-26    READ AND WRITE WORKS
Sireesh Gururaja    2013-11-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-26    read and write to stdin
Matt Dean    2013-11-26    equals, not equals, and concat
Matt Dean    2013-11-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-26    mergin with sireesh
Sireesh Gururaja    2013-11-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-26    chaining in add method
Matt Dean    2013-11-26    WORKS... GO JAVA!
Alden Quimby    2013-11-25    handle invalid keyword usage
Alden Quimby    2013-11-25    implement recursion inference with keyword "rec"
Alden Quimby    2013-11-25    clean up unify one into less cases
Alden Quimby    2013-11-25    better error logging
Alden Quimby    2013-11-25    better demo
Alden Quimby    2013-11-25    fix object access type inference bug, better unify errors
Alden Quimby    2013-11-24    print inferences
Alden Quimby    2013-11-24    woah, weird firey magic may have made things work
Alden Quimby    2013-11-24    get function as function param inference working
Alden Quimby    2013-11-24    omg crazy inference case, now im worried there are more unhandled cases...
Sireesh Gururaja    2013-11-24    committing the changes from meeting today
Alden Quimby    2013-11-24    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-24    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-24    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-24    add java to pc
Matt Dean    2013-11-24    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-24    pc object has char
Sireesh Gururaja    2013-11-24    minor changes
Alden Quimby    2013-11-24    Merge brat pushnch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-24    add some utility funcs
Matt Dean    2013-11-24    utiliy change
Alden Quimby    2013-11-24    move global funcs to each env
Alden Quimby    2013-11-24    fix more collect expressions;
Alden Quimby    2013-11-24    fix some collect bugs
Alden Quimby    2013-11-24    inference works
Alden Quimby    2013-11-24    get unification working with functions
Alden Quimby    2013-11-23    ugh, this is such a mind****
Alden Quimby    2013-11-23    temp commit, about to change strategies
Alden Quimby    2013-11-23    not sure what this is
Alden Quimby    2013-11-23    trying to figure out functions
Sireesh Gururaja    2013-11-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-23    javagen compiles
Alden Quimby    2013-11-23    type inference works!
Alden Quimby    2013-11-23    merge with dean
```

```
Alden Quimby    2013-11-23    finish object create unify
Matt Dean    2013-11-23    thiink i'm done with unify omg what
Matt Dean    2013-11-23    half way through unify Conflicts:     src/analyzer.ml
Matt Dean    2013-11-23    oh snap we bout to merge
Alden Quimby    2013-11-23    some unify_one stuff
Alden Quimby    2013-11-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-23    sketch out unify_one
Sireesh Gururaja    2013-11-23    merge resolution
Sireesh Gururaja    2013-11-23    type inference messing with this
Sireesh Gururaja    2013-11-23    working on localhost
Alden Quimby    2013-11-23    merge
Alden Quimby    2013-11-23    start unify
Matt Dean    2013-11-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-23    wooooooot
Kevin Michael Mangan    2013-11-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-11-23    removed distribute
Matt Dean    2013-11-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-23    compiles yay
Sireesh Gururaja    2013-11-21    Further progress on java generation. Minor correction to Makefile
Matt Dean    2013-11-19    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-19    think i finished constraints...
Alden Quimby    2013-11-19    asdfasdf
Alden Quimby    2013-11-19    testing
Alden Quimby    2013-11-19    Met purge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-19    add meeting notes
Kevin Michael Mangan    2013-11-19    scope stuff
Kevin Michael Mangan    2013-11-18    added more tests
Sireesh Gururaja    2013-11-18    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-18    incomplete java generator. Most functions tested and working.
Alden Quimby    2013-11-17    fix recursion, a little ugly but oh well
Alden Quimby    2013-11-16    add note
Alden Quimby    2013-11-16    get a simple program running
Alden Quimby    2013-11-16    omg it compiles
Alden Quimby    2013-11-16    get scanner parser building again
Alden Quimby    2013-11-16    merged
Alden Quimby    2013-11-16    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-16    annotate!
Matt Dean    2013-11-16    all my stuff for constraints part 1
Alden Quimby    2013-11-16    pushing for dean
Alden Quimby    2013-11-16    merge with dean
Alden Quimby    2013-11-16    temp commit
Matt Dean    2013-11-16    workin in uris
Alden Quimby    2013-11-16    start sketching out sast
Alden Quimby    2013-11-13    move things around
Kevin Michael Mangan    2013-11-11    Merge branch 'master' of https://github.com/mattydoincode/PLT
Kevin Michael Mangan    2013-11-11    accidentally pushed old slave.java
Matt Dean    2013-11-11    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-11    just pushing the middle version so i can make it dynamic
Alden Quimby    2013-11-10    t Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-11-10    add meeting notes
Matt Dean    2013-11-10    oops pushed the mo files
Matt Dean    2013-11-10    note on lamdba expressions
Sireesh Gururaja    2013-11-10    Java conversion list
Alden Quimby    2013-11-10    wrap my head around type inference
Kevin Michael Mangan    2013-11-09    m.a.a.d tests
Sireesh Gururaja    2013-11-09    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-11-09    updated haveDone
gsireesh    2013-11-09    Update README.md
Matt Dean    2013-11-07    it compiles!!!
```

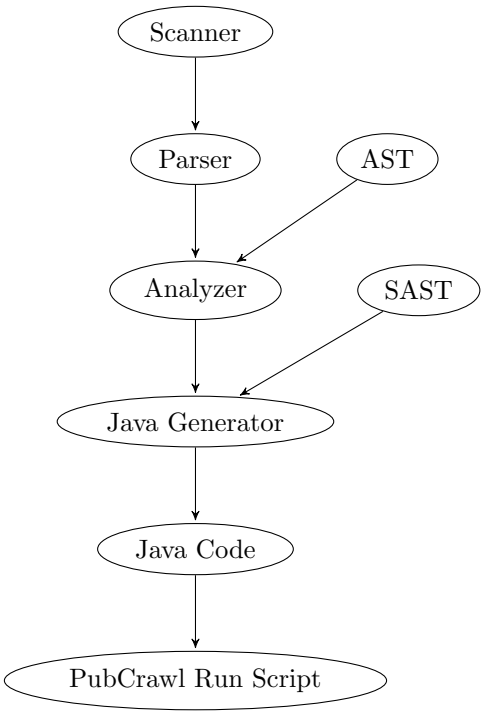
```
Matt Dean    2013-11-07    fixed dumb stmtlist option stuff, still not working
Matt Dean    2013-11-07    working on st tree walking
Alden Quimby    2013-11-04    add rule notes
Alden Quimby    2013-11-03    deans changes from meeting
Matt Dean    2013-11-03    minor syntax fix
Matt Dean    2013-11-03    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-11-03    neq
Sireesh Gururaja    2013-11-03    Incomplete JT
Matt Dean    2013-11-03    First try on semantic tree
Alden Quimby    2013-11-01    need ml file to print syntax tree
Alden Quimby    2013-11-01    parser now produces AST, fix a few parser bugs, gix AST
Alden Quimby    2013-10-31    temp commit to get sireeshs changes
Sireesh Gururaja    2013-10-31    adding properly
Sireesh Gururaja    2013-10-31    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-10-31    Finished AST
Alden Quimby    2013-10-31    add boolean test
Sireesh Gururaja    2013-10-31    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-10-31    Distribute working in this directory.
Alden Quimby    2013-10-30    fix makefile
Alden Quimby    2013-10-30    convert microc tests as starting point for tests, add file skeleton
Alden Quimby    2013-10-28    forgot to make clean
Alden Quimby    2013-10-28    fix access. sublist creation is not lvalue
Alden Quimby    2013-10-28    add boolean and/or/not
Alden Quimby    2013-10-28    clean up docs a little
Matt Dean    2013-10-27    changes made during meeting for LRM
Matt Dean    2013-10-27    scanner works for nums now
Sireesh Gururaja    2013-10-25    Modified PCList to be iterable.
Sireesh Gururaja    2013-10-25    Code for RMI, not working because of security.
mattydoincode    2013-10-21    random stuff?
Alden Quimby    2013-10-17    remove old parser
Alden Quimby    2013-10-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-10-17    formatting changes
Sireesh Gururaja    2013-10-17    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-10-17    The initial, crude draft of the Distribute server.
Alden Quimby    2013-10-17    forgot to save file, (facepalm)
Alden Quimby    2013-10-17    omg parser works
Alden Quimby    2013-10-15    add object access, clean functions and for loops
Alden Quimby    2013-10-15    get small test case working
Alden Quimby    2013-10-15    everything but obj creation working
Alden Quimby    2013-10-15    everything but obj/list creation working
Alden Quimby    2013-10-15    add for, while and if/elif/else
Alden Quimby    2013-10-15    take a step back, get small parser working
mattydoincode    2013-10-14    work in the lounge afterPLT
mattydoincode    2013-10-10    got some CFG going for our language yo
mattydoincode    2013-10-10    got cmicro working with strings
Alden Quimby    2013-10-10    fix interpreter spacing
Alden Quimby    2013-10-09    add microc compiler
Alden Quimby    2013-10-06    ignore misc.xml
Matt Dean    2013-10-06    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-10-06    minor doc fixes
Alden Quimby    2013-10-06    fix gitignore;
Alden Quimby    2013-10-06    trying to figure out why workspace isnt ignored
Matt Dean    2013-10-06    git ignore include .idea
Matt Dean    2013-10-06    git ignore
Kevin Michael Mangan    2013-10-06    added list examples
Alden Quimby    2013-10-03    add possible syntax ambiguity
Alden Quimby    2013-10-03    methods start lowercase in java :(. also add IPCFunction
Alden Quimby    2013-10-03    restructure folders
Matt Dean    2013-10-03    funcs!!
```

```
Matt Dean    2013-10-03    bubble sort! works in java!
Matt Dean    2013-10-03    pc object/list progress
Matt Dean    2013-10-03    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-10-03    java changes!!
Alden Quimby    2013-09-30    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-09-30    add meeting notes
Alden Quimby    2013-09-30    Remove var
Alden Quimby    2013-09-27    add utilities
Alden Quimby    2013-09-26    fix gitignore
Alden Quimby    2013-09-26    Merge branch 'master' of https://github.com/mattydoincode/PLT
Alden Quimby    2013-09-26    add java IO
Alden Quimby    2013-09-25    Add line breaks to description
Alden Quimby    2013-09-25    Update proposal latex with version we submitted
Alden Quimby    2013-09-25    Remove syntax ideas now that its captured in proposal syntax
Alden Quimby    2013-09-25    add meeeting notes
Sireesh Gururaja    2013-09-24    fixed a typo, changed margins
Alden Quimby    2013-09-24    add proposal latex
Sireesh Gururaja    2013-09-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Sireesh Gururaja    2013-09-23    Task list files
Matt Dean    2013-09-23    syntax/examples with team meeting
Matt Dean    2013-09-23    Merge branch 'master' of https://github.com/mattydoincode/PLT
Matt Dean    2013-09-23    shared info‘
Kevin Mangan    2013-09-23    Changed second paragraph
Kevin Mangan    2013-09-23    Update description.txt
Kevin Mangan    2013-09-23    word wrap
Kevin Mangan    2013-09-23    Kevin's part
Matt Dean    2013-09-23    changes in meeting
Matt Dean    2013-09-23    BOOM, added examples
Sireesh Gururaja    2013-09-22    A basic description of the language.
Alden Quimby    2013-09-20    wrap up syntax for proposal
Alden Quimby    2013-09-18    Update syntax ideas from meeting
Alden Quimby    2013-09-17    initial syntax ideas
Alden Quimby    2013-09-17    initial syntax thoughts, similar to coffee script
Kevin Michael Mangan    2013-09-17    possible suggestion
mattydoincode    2013-09-17    Initial commit
```

Chapter 5

Architectural Design

The architectural design of pubCrawl is illustrated in the following diagram:



As can be seen, the steps can be divided into the following sections:

1. Scanning
2. Parsing
3. Analyzing

4. Java code generation
5. Running the Java code.

Scanning, parsing and analyzing were handled by Alden and Matt, while Java code generation and java implementation were handled by Sireesh and Kevin.

5.1 Scanning

The pubCrawl scanner tokenizes the input into pubCrawl readable units. This process involves discarding whitespace and comments. Illegal character combinations, such as malformed escape sequences, are caught here. The scanner was written with `ocamllex`.

5.2 Parsing and Abstract Syntax Tree

The parser generates an abstract syntax tree (AST) from the tokens provided by the scanner. Syntax errors are caught here. The scanner was written with `ocamlyacc`. The AST describes the statements and their associated expressions. However, it is not typesafe.

5.3 Analysis and SAST

The analyzer walks the abstract sytntax tree produced by the parser, generates a typesafe, semantically checked abstract syntax tree (SAST). Since pubCrawl does not require the user to declare types, the inference of those types happens at this stage, using the Hindley-Milner Algorithm for type inference. This process detects all type mismatches, including the passing of wrongly-typed parameters and bad assignments.

The semantic checking portion checks for other errors, such as scope errors, and the reassignment of special functions.

5.4 Java Generation

This module walks the SAST produced by the analyzer and generates java code corresponding to the program. Each function declared in pubCrawl receives its own java file; these are all output into a folder called `java/` at the end of the compilation process. While this module *generates* java code, it does not compile it; that is acheived by the pubCrawl compiler script.

Chapter 6

Test Plan

6.1 Introduction

Our testing suite implements a couple test programs for each individual aspect of pubCrawl. There are individual tests for:

- arithmetic
- booleans
- collections
- contain function
- distribute function
- download function
- find function
- for loops
- function declaration
- function calling
- if/else statements
- map function
- object creation
- object access
- list range
- reading/writing to/from file
- changing between string and number
- type inference
- recursion
- split function
- strings

- variables
- where function
- while loops

6.2 Example Test (Testing Java Output

```
equals = (a,b) -> {return a==b;};

x = 4;
y = 5;

nope = x==5;

yup = (x+1)==5;

print("hello world");

myotherthing = read();

print(myotherthing[0]);
```

Java output:

```
public class output
{
  public static void main(String[] args)
  {
      DistributeClient.setSlaves(args);
    IPCFunction equals = (IPCFunction)(new function_20910());
    PCObject x = (PCObject)(new PCObject(4));
    PCObject y = (PCObject)(new PCObject(5));
    PCObject nope = (PCObject)(new PCObject(x.equals(new PCObject(5))));
    PCObject yup = (PCObject)(new PCObject(new PCObject(x.<Double>getBase()
        + new PCObject(1).<Double>getBase()).equals(new PCObject(5))));
    IO.get("print").call(new PCList("hello world"));
    PCList myotherthing = (PCList)(((PCList)IO.get("read").call()));
    IO.get("print").call(myotherthing.<PCObject>get(new PCObject(0)));
  }
}
```

6.3 Example Test (Testing Actual Output)

```
myObj = {
  gcd: (list) -> {
    a = list[0];
    b = list[1];
    while (a != b) {
      if (a > b) {
        a = a - b;
         }
         else {
        b = b - a;
```

```
          }
      }
      return a;
    },

    listOflists : [[2, 14], [3, 15], [99, 121]]

};



results = distribute(myObj.listOflists, myObj.gcd);

print(results[0]);
print(results[1]);
print(results[2]);
```

Output:

```
2
3
11
```

Chapter 7

Lessons Learned

7.1 Matt Dean

I learned that I take modern programming languages for granted. I'm quick to praise languages for including features I like and even quicker to label languages as subpar because they're missing this or that. Working on this project we made some interesting goals for our language – it should include type inference, dynamic nested objects, distributed computing, etc etc – and only succesfully accomplished a handful of them. Some would argue that's bad planning, but I feel differently. We got to experience deep and difficult problems, the same ones real programming language developers pondered and reconciled when they made some of our favorite languages. Before this class, I couldn't see a property of a certain programming language and truly understand what it took to make it a reality. Now I look at a language like javascript and think about how difficult it would have been to make our language truly dynamic. Or I look at a language like python and think about how impressive (and stable) their type inference algorithm is. Perl still sucks though.

7.2 Sireesh Gururaja

Because I've had to spend so much time debugging finicky network-type problems, I've learned to be really careful about keeping my environments clean and knowing at all times what files might affect what I'm working on. When you're writing a compiler, having old stray tests lying around in your working environment will cause you to waste a lot of time debugging, and it's worth the extra time to organize your files repeatedly.

I think it's also important to keep an eye on the big picture of the compiler and not get too involved with the tasks you're working on. Problems sometimes propagate through the different levels of a compiler, and it makes solving them a lot easier if you can just look at the different modules and know what's going on, rather than having to wait until someone else is free. Having an eye on the big picture also gives you realistic estimates on the project timeline - when our "Hello World" test compiled, I was shocked, because I had been worried about whether we would finish at all. Had I been more aware of the overall progress of the compiler, I could have saved myself a lot of worry.

7.3 Kevin Mangan

Creating a programming language from scratch sounds like an incredibly daunting task - and in reality, it is. However, by diving into such a task, I have learned the incredible amount of work that goes into creating a language and have greater repect and understanding for the languages I use every day. Finally getting to use the theoretical concepts we were taught early on in our CS curriculum is very rewarding.

Having a great group and working with intelligent people definitely made this project go a lot smoother. Don't be afraid to ask for help from other group members if you don't understand what a particular part of their code is doing - communication is key when it comes down to gluing all the individual parts of

the compiler together. It definitely helped to have a central form of communication such as a group chat application in case you need to send out a quick message, because email would have definitely been too slow/inconvenient.

7.4 Alden Quimby

Everyone says it, but start as early as possible. After figuring out what you want your language to do, dive in to syntax and get some example programs down so you have a clear goal in mind. At the same time, try to imagine what those programs might look like in your target language.

Don't try to add a handful of "cool" features to your language, and don't think you need to have all of the features of modern programming languages. Figure out why your language is interesting, and stay focused on that, everything else is just cruft.

Mandate weekly group meetings. Even if you have a week with no updates, use the meeting as a time to work on the project. If possible, try to do your project work in the same time/place as other team members so that as issues come up, you can address them immediately and don't lose time emailing back and forth, or waiting for the next meeting.

Team productivity depends on your ability to parallelize work, so spend a good chunk of time defining interfaces between modules (meaning the AST, SAST, etc.) with the whole team, so that team members can work on modules separately.

Lastly, use github! If you don't know git, learn it. If you haven't used github, use it. These days, every startup and many larger corporations use github. Get on board and you'll be able to contribute in the real world much faster when you get a job.

Chapter 8

Appendix

This appendix contains the code listing for pubCrawl. Counts are not included for test programs or demos.

File Name	Lines of Code
analyzer.ml	681
ast.ml	116
compile.sh	8
javagen.ml	353
Makefile	54
parser.mly	180
pc.ml	30
run.sh	15
sast.ml	218
scanner.mll	70
slave.sh	5
testall.sh	112
Compute.java	8
DistributeClient.java	104
DistributeServer.java	41
FileServer.java	66
IO.java	194
IPCFunction.java	4
List.java	149
PCList.java	79
PCObject.java	94

8.1 Scanner

```
{
  open Parser
  open Scanf

  let unescaped s =
    Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }
| "/*"                 { comment lexbuf }
| "//"                 { singlelinecom lexbuf}
```

```
| '('                   { LPAREN }
| ')'                   { RPAREN }
| '['                   { LBRACK }
| ']'                   { RBRACK }
| '{'                   { LBRACE }
| '}'                   { RBRACE }
| ';'                   { SEMI }
| ':'                   { COLON }
| "->"                  { ARROW }
| ','                   { COMMA }
| '.'                   { ACCESS }
| '+'                   { PLUS }
| '-'                   { MINUS }
| '*'                   { TIMES }
| '/'                   { DIVIDE }
| '%'                   { MOD }
| '='                   { ASSIGN }
| '^'                   { CONCAT }
| "=="                  { EQ }
| "!="                  { NEQ }
| '<'                   { LT }
| "<="                  { LEQ }
| ">"                   { GT }
| ">="                  { GEQ }
| "&&"                  { AND }
| "||"                  { OR }
| '!'                   { NOT }
| "if"                  { IF }
| "elif"                { ELIF }
| "else"                { ELSE }
| "for"                 { FOR }
| "while"               { WHILE }
| "return"              { RETURN }
| "distribute"          { DISTRIBUTE }
| "true"                { BOOLEAN_LIT(true) }
| "false"               { BOOLEAN_LIT(false) }
| ('\'' ([' '-'&' '('-'[' ']'-'~'] as c) '\'')
                        { CHAR_LIT(c) }
| ("'\\\\'" | "'\\''" | "'\\n'" | "'\\r'" | "'\\t'") as s
                        { CHAR_LIT((unescaped s).[0]) }
| ('0' | ['1'-'9']+['0'-'9']*)(['.']['0'-'9']+)? as lxm
                        { NUM_LIT(float_of_string lxm) }
| '"' (([' '-'!' '#'-'[' ']'-'~'] | '\\' ['\\' '"' 'n' 'r' 't'])* as s) '"'
                        { STRING_LIT(s) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm
                        { ID(lxm) }
| eof                   { EOF }
| _ as c                { raise (Failure("illegal character " ^ Char.escaped c)) }

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

and singlelinecom = parse
  "\n" { token lexbuf }
| eof  { EOF }
| _    { singlelinecom lexbuf}
```

8.2 Parser

```
%{ open Ast

  (* convert string into char list *)
  let explode s =
    let rec exp i l =
      if i < 0 then l else exp (i - 1) (s.[i] :: l) in
    exp (String.length s - 1) []

%}

%token LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK
%token SEMI COMMA ASSIGN COLON ARROW CONCAT ACCESS
%token PLUS MINUS TIMES DIVIDE MOD
%token EQ NEQ LT LEQ GT GEQ AND OR NOT
%token RETURN IF ELIF ELSE FOR WHILE DISTRIBUTE
%token <string> ID
%token <float> NUM_LIT
%token <bool> BOOLEAN_LIT
%token <string> STRING_LIT
%token <char> CHAR_LIT
%token EOF

%nonassoc ASSIGN COLON
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS CONCAT
%left TIMES DIVIDE MOD
%right NOT
%left ACCESS
%left LBRACK RBRACK
%left LPAREN RPAREN

%start program
%type <Ast.program> program

%%

program:
  stmt_list { List.rev $1 }

/***************
    STATEMENTS
***************/

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

body:
   LBRACE stmt_list RBRACE { List.rev $2 }
```

```
stmt:
    assignment SEMI { $1 }
  | func_call SEMI  { FuncCallStmt(fst $1, snd $1) }
  | FOR LPAREN assign_opt SEMI expr_opt SEMI assign_opt RPAREN body
      { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN body
      { While($3, $5) }
  | IF LPAREN expr RPAREN body elifs else_opt
      { If({condition=$3;body=$5} :: $6, $7) }
  | RETURN expr SEMI { Return($2) }

assign_opt:
    /* nothing */ { None }
  | assignment    { match $1 with Assign(e1, e2) -> Some(e1, e2) | _ -> None }

assignment:
    ID ASSIGN expr     { Assign(Id($1), $3) }
  | access ASSIGN expr { Assign($1, $3) }

/***************
    EXPRESSIONS
***************/

expr_opt:
    /* nothing */ { None }
  | expr          { Some($1) }

expr:
    NUM_LIT            { NumLit($1) }
  | BOOLEAN_LIT        { BoolLit($1) }
  | CHAR_LIT           { CharLit($1) }
  | STRING_LIT         { ListCreate(List.map (fun x -> CharLit(x)) (explode $1)) }
  | ID                 { Id($1) }
  | func_create        { $1 }
  | func_call          { FuncCallExpr(fst $1, snd $1) }
  | access             { $1 }
  | list_create        { $1 }
  | obj_create         { $1 }
  | LPAREN expr RPAREN { $2 }
  | expr CONCAT expr   { Binop($1, Concat,  $3) }
  | 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 MOD    expr   { Binop($1, Mod,     $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) }
  | NOT expr           { Not($2) }

elifs:
    /* nothing */                      { [] }
  | ELIF LPAREN expr RPAREN body elifs { {condition=$3;body=$5} :: $6 }
```

```
else_opt:
    /* nothing */ { None }
  | ELSE body     { Some($2) }

/***************
    FUNCTIONS
***************/

/*
  1. () -> body
  2. (x) -> body
  3. (x,y,z) -> body
  4. x -> body
*/
func_create:
    LPAREN RPAREN ARROW body              { FuncCreate([], $4) }
  | LPAREN expr RPAREN ARROW body         { match $2 with
                                              Id(x) -> FuncCreate([x], $5)
                                            | _ -> failwith "Invalid function creation."
                                          }
  | LPAREN mult_formals RPAREN ARROW body { FuncCreate($2, $5) }
  | ID ARROW body                         { FuncCreate([$1], $3) }

mult_formals:
  formal_list COMMA ID { List.rev ($3 :: $1) }

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

func_call:
  expr LPAREN actuals_opt RPAREN { ($1, $3) }
  | DISTRIBUTE LPAREN actuals_opt RPAREN { (Id("distribute"), $3)}

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

8.3 AST

```
(* Abstract Syntax Tree
    - created by parser
    - consumed by semantic analyzer
*)

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

type expr =
    NumLit of float
  | BoolLit of bool
  | CharLit of char
  | Id of string
  | FuncCreate of string list * stmt list
```

```
  | FuncCallExpr of expr * expr list
  | ObjAccess of expr * string
  | ListAccess of expr * expr
  | ListCreate of expr list
  | Sublist of expr * expr option * expr option
  | ObjCreate of (string * expr) list
  | Binop of expr * op * expr
  | Not of expr

and stmt =
    Return of expr
  | If of conditional list * stmt list option
  | For of (expr * expr) option * expr option * (expr * expr) option * stmt list
  | While of expr * stmt list
  | Assign of (expr * expr)
  | FuncCallStmt of expr * expr list

and conditional = {
  condition : expr;
  body : stmt list;
}

type program = stmt list


(*************************
**** PRINT AST **********
*************************)

let string_of_opt string_of = function
    Some(x) -> string_of x
  | None -> ""

let rec string_of_expr = function
    NumLit(n) -> string_of_float n
  | BoolLit(b) -> string_of_bool b
  | CharLit(c) -> "'" ^ Char.escaped c ^ "'"
  | Id(s) -> s
  | Not(e) ->  "!" ^ string_of_expr e
  | Binop(e1, op, e2) ->
      string_of_expr e1 ^ (match op with
        Add -> " + "    | Sub -> " - "     | Mult -> " * "
      | Div -> " / "    | Mod -> " % "
      | Equal -> " == " | Neq -> " != "    | Less -> " < "
      | Leq -> " <= "   | Greater -> " > " | Geq -> " >= "
      | Concat -> " ^ " | And -> " && "    | Or -> " || ") ^
      string_of_expr e2
  | FuncCallExpr(e, el) ->
      string_of_expr e ^ "(" ^
      String.concat ", " (List.map string_of_expr el) ^ ")"
  | FuncCreate(formals, body) ->
      "(" ^ String.concat ", " formals ^ ") -> {\n" ^
      String.concat "" (List.map string_of_stmt body) ^ "\n}"
  | ListCreate(exprs) ->
      "[" ^ String.concat ", " (List.map string_of_expr exprs) ^ "]"
  | Sublist(e, eleft, eright) ->
      string_of_expr e ^ "[" ^
      string_of_opt string_of_expr eleft ^ ":" ^
```

```
      string_of_opt string_of_expr eright ^ "]"
  | ListAccess(e1, e2) ->
      string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
  | ObjCreate(props) ->
      "{\n" ^ String.concat ",\n" (List.map
          (fun(prop) -> fst prop ^ ": " ^ string_of_expr (snd prop))
      props) ^ "\n}"
  | ObjAccess(e, s) ->
      string_of_expr e ^ "." ^ s

and string_of_stmt = function
    Return(expr) -> "return " ^ string_of_expr expr ^ ";";
  | If(conds, elsebody) ->
      "if" ^ string_of_cond (List.hd conds) ^ String.concat ""
        (List.map (fun(x) -> "\nelif" ^ string_of_cond x) (List.tl conds)) ^
        string_of_opt (fun(x) -> "\nelse {\n" ^ string_of_stmts x ^ "}") elsebody
  | For(a1, e, a2, s) ->
      "for (" ^ string_of_opt string_of_assign a1 ^ "; " ^
                string_of_opt string_of_expr e ^ "; " ^
                string_of_opt string_of_assign a2 ^ ") {\n" ^
      string_of_stmts s ^ "}"
  | While(e, s) ->
      "while (" ^ string_of_expr e ^ ") {\n" ^
      string_of_stmts s ^ "}"
  | Assign(a) -> string_of_assign a ^ ";"
  | FuncCallStmt(e, el) ->
      string_of_expr e ^ "(" ^
      String.concat ", " (List.map string_of_expr el) ^ ");"

and string_of_stmts stmts =
  String.concat "\n" (List.map string_of_stmt stmts) ^ "\n"

and string_of_cond cond =
  " (" ^ string_of_expr cond.condition ^ ") {\n" ^
  string_of_stmts cond.body ^ "}"

and string_of_assign ((e1, e2)) =
  string_of_expr e1 ^ " = " ^ string_of_expr e2

let string_of_prog prog =
  string_of_stmts prog
```

8.4 Analyzer

```
(* Semantic analysis
  - input: abstract syntax tree
  - build symbol table
  - run type inference algorithm
  - output: semantic tree
*)

open Ast
open Sast

(*************************
****** UTILITIES *********
```

```
************************)

type symbol_table = {
  parent : symbol_table option;
  mutable variables : (string * Sast.t) list;
}

type environment = {
  mutable func_return_type : Sast.t option; (* Function's return type *)
  scope : symbol_table;        (* symbol table for vars *)
}

let str_eq a b = ((Pervasives.compare a b) = 0)

let rec find_variable (scope : symbol_table) (name : string) : Sast.t option =
  try
    let (_, typ) = List.find (fun (s, _) -> s = name) scope.variables in
    Some(typ)
  with Not_found ->
    match scope.parent with
    | Some(p) -> find_variable p name
    | _ -> None

let find_prop (prop : string * Sast.t) (props : (string * Sast.t) list) : (string * Sast.t) option =
  try
    let found = List.find (fun prop2 -> (fst prop) = (fst prop2)) props in
    Some(found)
  with Not_found ->
    None

let code1 = ref (Char.code 'A')
let code2 = ref (Char.code 'A')

let next_type_var() : Sast.t =
  let c1 = !code1 in
  let c2 = !code2 in
    if c2 = Char.code 'Z'
    then code2 := Char.code 'a'
    else incr code2;
    if c2 = Char.code 'z'
    then (incr code1; code2 := Char.code 'A')
    else ();
    if c1 = Char.code 'Z'
    then code1 := Char.code 'a'
    else ();
    let name = (Char.escaped (Char.chr c1)) ^ (Char.escaped (Char.chr c2)) in
    TVar(name)

let type_of (ae : Sast.aExpr) : Sast.t =
  match ae with
  | ANumLit(_, t) -> t
  | ABoolLit(_, t) -> t
  | ACharLit(_, t) -> t
  | AId(_, _, t) -> t
  | AFuncCreate(_, _, t) -> t
  | AFuncCallExpr(_, _, t) -> t
  | AObjAccess(_, _, t) -> t
  | AListAccess(_, _, t) -> t
```

```
  | AListCreate(_, t) -> t
  | ASublist(_, _, _, t) -> t
  | AObjCreate(_, t) -> t
  | ABinop(_, _, _, t) -> t
  | ANot(_, t) -> t

(* inside a for loop or if block, nest scope *)
let nest_scope (env : environment) : environment =
  let s = { variables = []; parent = Some(env.scope) } in
  { env with scope = s; }

(* inside a function, nest entire env *)
let new_env() : environment =
  let print_type = next_type_var() in
  let print_file_type = next_type_var() in
  let dist_type = next_type_var() in
  let dist_return_type = next_type_var() in
  let add_type = next_type_var() in
  let remove_type = next_type_var() in
  let where_type = next_type_var() in
  let mapping_type = next_type_var() in
  let mapped_type = next_type_var() in
  let find_type = next_type_var() in
  let split_type = next_type_var() in
  let pop_type = next_type_var() in
  let core = [
    ("print", TFunc([print_type],
                    print_type));
    ("read", TFunc([],
                   TList(TChar)));
    ("printFile", TFunc([print_file_type; TList(TChar)],
                        print_file_type));
    ("readFile", TFunc([TList(TChar)],
                       TList(TList(TChar))));
    ("download", TFunc([TList(TChar)],
                       TList(TChar)));
    ("distribute", TFunc([TList(dist_type); TFunc([dist_type],dist_return_type)],
                         TList(dist_return_type)));
    ("numToString", TFunc([TNum],
                          TList(TChar)));
    ("numFromString", TFunc([TList(TChar)],
                            TNum))
  ] in
  let list_util = TObj([
    ("add", TFunc([TList(add_type); add_type],
                  TList(add_type)));
    ("remove", TFunc([TList(remove_type); TNum],
                  TList(remove_type)));
    ("where", TFunc([TList(where_type); TFunc([where_type], TBool)],
                  TList(where_type)));
    ("map", TFunc([TList(mapping_type); TFunc([mapping_type], mapped_type)],
                  TList(mapped_type)));
    ("find", TFunc([TList(find_type); TList(find_type)],
                   TNum));
    ("split", TFunc([TList(split_type); split_type],
                  TList(TList(split_type))));
    ("range", TFunc([TNum; TNum],
                  TList(TNum)));
```

```
    ("populate", TFunc([pop_type; TNum],
                       TList(pop_type)));
    ("length", TFunc([TList(next_type_var())],
                     TNum))
  ]) in
  let s = { variables = ("List", list_util) :: core; parent = None } in
  { scope = s; func_return_type = None; }

let is_keyword (name : string) : bool =
  let rec helper (name : string) (words : string list) : bool =
    match words with
    | [] -> false
    | h::t -> name = h || helper name t
  in
  helper name ["rec";"print";"read";"printFile";"readFile";"download";"distribute";"numToString";"numFromString"]




(*************************
****** ANNOTATE **********
*************************)

let rec annotate_expr (e : Ast.expr) (env : environment) : Sast.aExpr =
  match e with
  | NumLit(n) -> ANumLit(n, TNum)
  | BoolLit(b) -> ABoolLit(b, TBool)
  | CharLit(c) -> ACharLit(c, TChar)
  | Id(s) ->
      let typ = find_variable env.scope s in
      (match typ with
      | Some(x) -> AId(s, false, x)
      | None -> failwith ("Unrecognized identifier " ^ s ^ "."))
  | FuncCreate(formals, body) ->
      if List.exists is_keyword formals
      then failwith "Function parameter cannot be keyword."
      else
      let new_type = next_type_var() in
      let new_env = new_env() in
      let formals_with_type = List.map (fun x -> (x, next_type_var())) formals in
      let func_type = TFunc(List.map snd formals_with_type, new_type) in
      new_env.func_return_type <- Some(new_type);
      new_env.scope.variables <- formals_with_type @ [("rec", func_type)] @ new_env.scope.variables;
      let aBody = annotate_stmts body new_env in
      AFuncCreate(formals_with_type, aBody, func_type)
  | FuncCallExpr(e, elist) ->
      let ae = annotate_expr e env in
      let aelist = List.map (fun x -> annotate_expr x env) elist in
      let new_type = next_type_var() in
      AFuncCallExpr(ae, aelist, new_type)
  | ObjAccess(e, s) ->
      let ae = annotate_expr e env in
      let new_type = next_type_var() in
      AObjAccess(ae, s, new_type)
  | ListAccess(e1, e2) ->
      let ae1 = annotate_expr e1 env in
      let ae2 = annotate_expr e2 env in
```

```
      let new_type = next_type_var() in
      AListAccess(ae1, ae2, new_type)
  | ListCreate(exprs) ->
      let aExprs = List.map (fun x -> annotate_expr x env) exprs in
      let new_type = next_type_var() in
      AListCreate(aExprs, TList(new_type))
  | Sublist(e, eleft, eright) ->
      let ae = annotate_expr e env in
      let aeleft = match eleft with
      | Some(x) -> Some(annotate_expr x env)
      | None -> None in
      let aeright = match eright with
      | Some(x) -> Some(annotate_expr x env)
      | None -> None in
      let new_type = next_type_var() in
      ASublist(ae, aeleft, aeright, TList(new_type))
  | ObjCreate(props) ->
      let rec check_dups = function
          [] -> false
        | (h::t) -> if List.mem h t then true else check_dups t
      in
      let prop_names = List.map fst props in
      if check_dups prop_names
      then failwith "Duplicate property names on object."
      else if List.exists is_keyword prop_names
      then failwith "Ojbect property cannot be keyword."
      else
      let aProps = List.map (fun (name, e) -> (name, annotate_expr e env)) props in
      let types = List.map (fun (name, ae) -> (name, type_of ae)) aProps in
      AObjCreate(aProps, TObj(types))
  | Binop(e1, op, e2) ->
      let ae1 = annotate_expr e1 env in
      let ae2 = annotate_expr e2 env in
      let new_type = next_type_var() in
      ABinop(ae1, op, ae2, new_type)
  | Not(e) ->
      let ae = annotate_expr e env in
      ANot(ae, TBool)

and annotate_assign (e1 : Ast.expr) (e2 : Ast.expr) (env : environment) : Sast.aExpr * Sast.aExpr =
  let ae2 = annotate_expr e2 env in
  match e1 with
  | Id(x) ->
      if is_keyword x
      then failwith "Cannot assign keyword."
      else
      let typ = find_variable env.scope x in
      (match typ with
      | Some(t) ->
          (AId(x, false, t), ae2)
      | None ->
          let new_type = next_type_var() in
          env.scope.variables <- (x, new_type) :: env.scope.variables;
          (AId(x, true, new_type), ae2))
  | ObjAccess(_, _) ->
      let ae1 = annotate_expr e1 env in
      (ae1, ae2)
  | ListAccess(_, _) ->
```

```
      let ae1 = annotate_expr e1 env in
      (ae1, ae2)
  | _ -> failwith "Invalid assignment."

and annotate_stmt (s : Ast.stmt) (env : environment) : Sast.aStmt =
  match s with
  | Return(expr) ->
    (match env.func_return_type with
    | None -> failwith "Invalid return statement."
    | Some(x) ->
        let ae = annotate_expr expr env in
        AReturn(ae, x))
  | Assign(e1, e2) ->
      let (ae1, ae2) = annotate_assign e1 e2 env in
      AAssign(ae1, ae2)
  | If(conds, elsebody) ->
      let aIfFunc = fun cond ->
        let ae = annotate_expr cond.condition env in
        let scoped_env = nest_scope env in
        let aBody = annotate_stmts cond.body scoped_env in
        (ae, aBody)
      in
      (match elsebody with
      | Some(x) ->
        let scoped_env = nest_scope env in
        let aElse = annotate_stmts x scoped_env in
        AIf(List.map aIfFunc conds, Some(aElse))
      | None -> AIf(List.map aIfFunc conds, None))
  | For(a1, e, a2, body) ->
    let scoped_env = nest_scope env in
    let aa1 =
      match a1 with
      | Some(e1, e2) -> Some(annotate_assign e1 e2 scoped_env)
      | None -> None
    in
    let ae =
      match e with
      | Some(x) -> Some(annotate_expr x scoped_env)
      | None -> None
    in
    let aa2 =
      match a2 with
      | Some(e1, e2) ->
        let assign = annotate_assign e1 e2 scoped_env in
        (match assign with
        | (AId(_, true, _), _) -> failwith "Cannot create variable at end of for loop."
        | _ -> Some(assign))
      | None -> None
    in
    let aBody = annotate_stmts body scoped_env in
    AFor(aa1, ae, aa2, aBody)
  | While(e, body) ->
      let ae = annotate_expr e env in
      let scoped_env = nest_scope env in
      let aBody = annotate_stmts body scoped_env in
      AWhile(ae, aBody)
  | FuncCallStmt(e, elist) ->
      let ae = annotate_expr e env in
```

```
      let aelist = List.map (fun x -> annotate_expr x env) elist in
      AFuncCallStmt(ae, aelist)

and annotate_stmts (stmts : Ast.stmt list) (env : environment) : Sast.aStmt list =
  List.map (fun x -> annotate_stmt x env) stmts

let annotate_prog (p : Ast.program) : Sast.aProgram =
  let env = new_env() in
  annotate_stmts p env




(*************************
****** COLLECT ***********
*************************)

let rec collect_expr (e : Sast.aExpr) : (Sast.t * Sast.t) list =
  match e with
    | ANumLit(num, ty) -> []
    | ABoolLit(boo, ty) -> []
    | ACharLit(c, ty) ->   []
    | AId(name, seenBefore, ty) -> []
    | AFuncCreate(params, body, ty) ->
        collect_stmts body
    | AFuncCallExpr(fExpr, params, ty) ->
        let ftype = type_of fExpr in
        let myCreatedType = TFunc(List.map (fun p -> type_of p) params, ty) in
        let param_constraints = (List.fold_left (fun l p -> l @ collect_expr p) [] params) in
        param_constraints @ collect_expr fExpr @ [(myCreatedType, ftype)]
    | AObjAccess(oExpr, name, ty) ->
        let oType = type_of oExpr in
        (match oType with
        | TVar(s) -> (oType, TObjAccess([(name, ty)], s)) :: collect_expr oExpr
        | _ -> (oType, TObjAccess([(name, ty)], (string_of_type oType))) :: collect_expr oExpr)
    | AListAccess(lExpr, iExpr, return_type) ->
        let idx_type = type_of iExpr in
        let list_type = type_of lExpr in
        [(list_type, TList(return_type));(idx_type,TNum)] @ collect_expr lExpr @ collect_expr iExpr
    | AListCreate(members, ty) ->
        let constraints =
          (match ty with
          | TList(x) -> List.map (fun m -> (type_of m, x)) members
          | _ -> failwith "Internal error, should never happen: list not a list.")
        in
        constraints @ (List.fold_left (fun l m -> l @ collect_expr m) [] members)
    | ASublist(mylist, e1, e2, ty) ->
        let e_constraint = fun e_opt ->
          match e_opt with
            | Some(x) -> (type_of x, TNum) :: collect_expr x
            | None -> []
        in
        let list_type = type_of mylist in
        [(list_type, ty)] @ e_constraint e1 @ e_constraint e2
    | AObjCreate(props, ty) ->
        let prop_exprs = List.map snd props in
        (List.fold_left (fun l p -> l @ collect_expr p) [] prop_exprs)
```

```
    | ABinop(e1, op, e2, ty) ->
        let e1t = type_of e1 in
        let e2t = type_of e2 in
        let opc =
          (match op with
          | Add | Sub | Mult | Div | Mod ->
              [(e1t, TNum); (e2t, TNum); (ty, TNum)]
          | Equal | Neq ->
              [(e1t,e2t); (ty, TBool)]
          | Less | Leq | Greater | Geq ->
              [(e1t, TNum); (e2t, TNum); (ty, TBool)]
          | Concat ->
              let new_type = next_type_var() in
              [(e1t, e2t); (ty, e1t); (ty, TList(new_type))]
          | And | Or ->
              [(e1t, TBool); (e2t, TBool); (ty, TBool)])
        in
        (collect_expr e1) @ (collect_expr e2) @ opc
    | ANot(e, ty) ->
        let notc = [(type_of e, TBool); (ty, TBool)] in
        (collect_expr e) @ notc

and collect_stmt (s : aStmt) : (Sast.t * Sast.t) list =
  match s with
    | AReturn(expr, func_return_t) ->
        let constraints_from_expr = collect_expr expr in
        (type_of expr,func_return_t) :: (constraints_from_expr)
    | AIf(conds, the_else) ->
        let cond_func = fun list_so_far (ex, stlist) ->
          let expr_list = (type_of ex, TBool) :: collect_expr ex in
          let my_list = expr_list @ (collect_stmts stlist) in
          list_so_far @ my_list
        in
        let list_after_ifs = List.fold_left cond_func [] conds in
        let list_after_cond = (match the_else with
          | Some(x) -> list_after_ifs @ collect_stmts x
          | None -> [])
        in
        list_after_cond @ list_after_ifs
    | AFor(assign1, e, assign2, stmts) ->
      let list_from_a1 =
        (match assign1 with
          | None -> []
          | Some(e1, e2) ->
            let aa = AAssign(e1, e2) in
            collect_stmt aa)
        in
      let list_from_a2 =
        (match assign2 with
          | None -> []
          | Some(e1, e2) ->
            let aa = AAssign(e1, e2) in
            collect_stmt aa)
        in
      let list_from_e =
        match e with
            None -> []
          | Some(ex) -> (type_of ex, TBool) :: collect_expr ex
```

```
        in
      let list_from_topbit = list_from_a1 @ list_from_a2 @ list_from_e in
      list_from_topbit @ (collect_stmts stmts)
    | AWhile(expr, stmts) ->
      let list_from_expr = (type_of expr, TBool) :: collect_expr expr in
      list_from_expr @ collect_stmts stmts
    | AAssign(lhs, rhs) -> (collect_expr lhs @ collect_expr rhs) @ [(type_of lhs, type_of rhs)]
    | AFuncCallStmt (fExpr, params) ->
      let ftype = type_of fExpr in
      let myCreatedType = TFunc(List.map (fun p-> type_of p) params, next_type_var()) in
      let param_constraints = (List.fold_left (fun l p -> l @ collect_expr p) [] params) in
      [(myCreatedType, ftype)] @ param_constraints @ collect_expr fExpr

and collect_stmts (stmts : Sast.aStmt list) : (Sast.t * Sast.t) list =
 List.fold_left (fun l s -> l @ (collect_stmt s)) [] stmts

let collect_prog (cprog : Sast.aProgram) : (Sast.t * Sast.t) list =
  collect_stmts cprog




(*************************
****** UNIFY *************
*************************)

type substitution = (string * Sast.t) list

(* substitute term s for all occurrences of var x in term t *)
let rec subst (s : Sast.t) (x : string) (typ : Sast.t) : Sast.t =
  match typ with
  | TVar(name) -> if x = name then s else typ
  | TFunc(params, y) -> TFunc(List.map (fun param -> subst s x param) params, subst s x y)
  | TList(y) -> TList(subst s x y)
  | TObj(props) -> TObj(List.map (fun prop -> (fst prop, subst s x (snd prop))) props)
  | TObjAccess(props, key) ->
    if (str_eq key x)
    then TObjAccess(List.map (fun prop -> (fst prop, subst s x (snd prop))) props, key)
    else TObjAccess(List.map (fun prop -> (fst prop, subst s x (snd prop))) props, key)
  | TNum -> typ
  | TChar -> typ
  | TBool -> typ



(* apply a substitution to t right to left *)
let apply (s : substitution) (typ : Sast.t) : Sast.t =
  List.fold_right (fun (x, e) -> subst e x) s typ

let rec copy_type (typ : Sast.t) (table : symbol_table) : Sast.t =
  match typ with
  | TVar(name) ->
      let typ = find_variable table name in
      (match typ with
      | Some(x) -> x
      | None ->
          let new_type = next_type_var() in
```

```
          table.variables <- (name, new_type) :: table.variables;
          new_type)
  | TFunc(params, y) -> TFunc(List.map (fun p -> copy_type p table) params, copy_type y table)
  | TList(y) -> TList(copy_type y table)
  | TObj(props) -> TObj(List.map (fun prop -> (fst prop, copy_type (snd prop) table)) props)
  | TObjAccess(props, key) -> TObjAccess(List.map (fun prop -> (fst prop, copy_type (snd prop) table)) props, key
  | TNum -> typ
  | TChar -> typ
  | TBool -> typ

let unify_failed (a : Sast.t) (b : Sast.t) (msg : string) =
  let msg = "\n\n*************\nType mismatch\n*************\n" ^ msg ^
            Sast.string_of_type a ^ " <> " ^ Sast.string_of_type b ^ "\n\n" in
  failwith msg

(* unify one pair *)
let rec unify_one (a : Sast.t) (b : Sast.t) : substitution  =
  match (a, b) with
  | (TVar(x), TVar(y)) ->
      if x = y then [] else [(x, b)]
  | (TFunc(params1, x), (TFunc(params2, y) as z)) ->
      let copy = copy_type z { variables = []; parent = None } in
      (match copy with
      | TFunc(new_params2, new_y) ->
          (try
            let pairs = List.map2 (fun u v -> (u,v)) new_params2 params1 in
            unify ((x,new_y)::pairs)
          with Invalid_argument(_) ->
            unify_failed a b "Function has wrong # of parameters.\n")
      | _ -> failwith "Internal error, should never happen: copying TFunc failed.")
  | (TObj(props1), TObj(props2)) ->
      let mapper = fun prop1 props2 ->
        match (find_prop prop1 props2) with
        | Some(found) -> ((snd prop1), (snd found))
        | None -> unify_failed a b ("Object missing property '" ^ fst prop1 ^ "'.\n")
      in
      let _ = (List.map (fun prop2 -> mapper prop2 props1) props2) in
      unify (List.map (fun prop1 -> mapper prop1 props2) props1)
  | (TList(x), TList(y)) ->
      unify_one x y
  | (TObjAccess(props1, key1), TObjAccess(props2, key2)) ->

      (* context = props * constraints *)
      let mapper = fun context prop1 allProps2  ->
        match find_prop prop1 allProps2 with
        | Some(found) -> (fst context, ((snd prop1), (snd found)) :: (snd context))
        | None -> (prop1 :: (fst context), snd context)
      in

      let (new_props, constraints) = List.fold_left (fun ctx prop -> mapper ctx prop props1) ([],[]) props2 in
      let full_props = new_props @ props1 in
      let always_sub = (key2, TObjAccess(full_props, key2)) in
      let obj_subs =
        if str_eq key1 key2
        then
          [always_sub]
        else
          [(key1, TVar(key2)); always_sub]
```

```
      in
      let subs = unify constraints in
      subs @ obj_subs
  | (TNum, TNum) | (TChar, TChar) | (TBool, TBool) ->
      []
  | (TVar(x), z) | (z, TVar(x)) ->
      [(x, z)]
  | (TObj(props), TObjAccess(accessProps, _)) | (TObjAccess(accessProps, _), TObj(props)) ->
      let mapper = fun prop1 props2 ->
        match find_prop prop1 props2 with
        | Some(found) -> ((snd prop1), (snd found))
        | None -> unify_failed a b ("Object missing property '" ^ fst prop1 ^ "'.\n")
      in
      unify (List.map (fun accessProp -> mapper accessProp props) accessProps)
  | (_, _) -> unify_failed a b ""

(* unify a list of pairs *)
and unify (s : (Sast.t * Sast.t) list) : substitution =
  match s with
  | [] -> []
  | (x, y) :: tl ->
      let t2 = unify tl in
      let t1 = unify_one (apply t2 x) (apply t2 y) in
      (* if t1 just returned like "heyo bitch" be like aight*)
      fixObjAccess t2 t1

and fixObjAccess (oldSubs: substitution) (newSubs : substitution) : substitution =
  let checkIfObjAccess = fun (str, ty) -> match ty with
    | TObjAccess(props, key) -> true
    | _ -> false
  in
  let listOfObjAccessSubs = List.filter checkIfObjAccess newSubs in
  let flagBad = fun (str, _) ->
      (List.length (List.filter (fun (st,ty) -> match ty with
        | TObjAccess(props, key) -> ((Pervasives.compare key str) = 0)
        | _ -> false
      ) listOfObjAccessSubs )) = 0
  in
  let newOldSubs = List.filter flagBad oldSubs in

  let oaTypes = List.map (fun (str, ty) -> (match ty with TObjAccess(p,k) -> (p,k) | _ -> ([],"")) ) listOfObjAcc
  let newnewOldSubs = List.fold_left (fun subsList oaType -> List.map (fun sub -> (fst sub, (replaceOA oaType (sn
  newSubs @ newnewOldSubs

and replaceOA (props, key)  (ty: Sast.t) : Sast.t =
  match ty with
  | TVar(name) -> ty
  | TFunc(params, y) -> TFunc(List.map (fun param -> replaceOA (props,key) param) params, replaceOA (props,key) y
  | TList(y) -> TList(replaceOA (props,key) y)
  | TObj(tprops) -> TObj(List.map (fun prop -> (fst prop, replaceOA (props,key) (snd prop))) tprops)
  | TObjAccess(tprops, tkey) ->

    if (str_eq key tkey)
    then (
      TObjAccess(props,key)
    )
    else (
      TObjAccess(List.map (fun prop -> (fst prop, replaceOA (props,key) (snd prop))) tprops, tkey)
```

```
    )
  | TNum -> ty
  | TChar -> ty
  | TBool -> ty

(************************
****** INFER *************
************************)

let rec apply_expr (ae : Sast.aExpr) (subs : substitution) : Sast.aExpr =
  match ae with
    | ANumLit(n, ty) ->
        ANumLit(n, apply subs ty)
    | ABoolLit(b, ty) ->
        ABoolLit(b, apply subs ty)
    | ACharLit(c, ty) ->
        ACharLit(c, apply subs ty)
    | AId(name, seenBefore, ty) ->
        AId(name, seenBefore, apply subs ty)
    | AFuncCreate(params, body, ty) ->
        let new_params = List.map (fun (name, typ) -> (name, apply subs typ)) params in
        AFuncCreate(new_params, apply_stmts body subs, apply subs ty)
    | AFuncCallExpr(fExpr, params, ty) ->
        let new_params = List.map (fun e -> apply_expr e subs) params in
        AFuncCallExpr(apply_expr fExpr subs, new_params, apply subs ty)
    | AObjAccess(oExpr, name, ty) ->
        AObjAccess(apply_expr oExpr subs, name, apply subs ty)
    | AListAccess(lExpr, iExpr, ty) ->
        AListAccess(apply_expr lExpr subs, apply_expr iExpr subs, apply subs ty)
    | AListCreate(members, ty) ->
        let new_members = List.map (fun m -> apply_expr m subs) members in
        AListCreate(new_members, apply subs ty)
    | ASublist(lExpr, e1, e2, ty) ->
        let e_opt_func = fun e_opt ->
          match e_opt with
          | Some(e) -> Some(apply_expr e subs)
          | None -> None
        in
        ASublist(apply_expr lExpr subs, e_opt_func e1, e_opt_func e2, apply subs ty)
    | AObjCreate(props, ty) ->
        let new_props = List.map (fun (name, e) -> (name, apply_expr e subs)) props in
        AObjCreate(new_props, apply subs ty)
    | ABinop(e1, op, e2, ty) ->
        ABinop(apply_expr e1 subs, op, apply_expr e2 subs, apply subs ty)
    | ANot(e, ty) ->
        ANot(apply_expr e subs, apply subs ty)

and apply_stmt (s : aStmt) (subs : substitution) : Sast.aStmt =
  match s with
    | AReturn(expr, func_return_t) ->
        AReturn(apply_expr expr subs, apply subs func_return_t)
    | AIf(conds, the_else) ->
        let cond_func = fun (expr, body) ->
          (apply_expr expr subs, apply_stmts body subs)
        in
        let else_func = fun e ->
          (match e with
          | Some(body) -> Some(apply_stmts body subs)
```

```
          | None -> None)
        in
        AIf(List.map cond_func conds, else_func the_else)
    | AFor(assign1, expr, assign2, stmts) ->
        let assign_func = fun a ->
          (match a with
          | Some(e1, e2) -> Some(apply_expr e1 subs, apply_expr e2 subs)
          | None -> None)
        in
        let expr_func = fun e ->
          (match e with
          | Some(e1) -> Some(apply_expr e1 subs)
          | None -> None)
        in
        AFor(assign_func assign1, expr_func expr, assign_func assign2, apply_stmts stmts subs)
    | AWhile(expr, stmts) ->
        AWhile(apply_expr expr subs, apply_stmts stmts subs)
    | AAssign(lhs, rhs) ->
        AAssign(apply_expr lhs subs, apply_expr rhs subs)
    | AFuncCallStmt(fExpr, params) ->
        let new_params = List.map (fun e -> apply_expr e subs) params in
        AFuncCallStmt(apply_expr fExpr subs, new_params)

and apply_stmts (stmts : Sast.aStmt list) (subs : substitution) : Sast.aStmt list =
  List.map (fun s -> apply_stmt s subs) stmts

let infer_prog (p : Ast.program) : Sast.aProgram =
  let ap = annotate_prog p in
  let constraints = collect_prog ap in
  let subs = unify (List.rev constraints) in
  apply_stmts ap subs
```

8.5 SAST

```
include Ast

(*************************
**** SAST ****************
*************************)

(* types for inference *)
type t =
    TVar of string
  | TFunc of t list * t
  | TList of t
  | TObj of (string * t) list
  | TObjAccess of string * t
  | TNum
  | TChar
  | TBool

(* annotated expression *)
type aExpr =
    ANumLit of float * t
  | ABoolLit of bool * t
  | ACharLit of char * t
```

```
  | AId of string * bool * t
  | AFuncCreate of (string * t) list * aStmt list * t
  | AFuncCallExpr of aExpr * aExpr list * t
  | AObjAccess of aExpr * string * t
  | AListAccess of aExpr * aExpr * t
  | AListCreate of aExpr list * t
  | ASublist of aExpr * aExpr option * aExpr option * t
  | AObjCreate of (string * aExpr) list * t
  | ABinop of aExpr * Ast.op * aExpr * t
  | ANot of aExpr * t

(* annotated statment *)
and aStmt =
    AReturn of aExpr * t
  | AIf of (aExpr * aStmt list) list * aStmt list option
  | AFor of (aExpr * aExpr) option * aExpr option * (aExpr * aExpr) option * aStmt list
  | AWhile of aExpr * aStmt list
  | AAssign of (aExpr * aExpr)
  | AFuncCallStmt of aExpr * aExpr list

(* annotated program *)
type aProgram = aStmt list


(*************************
**** PRINT SAST **********
*************************)

let string_of_opt string_of = function
    Some(x) -> string_of x
  | None -> ""

let rec string_of_type = function
    TVar(s) -> "'" ^ s
  | TFunc(tlist, t) -> "TFunc(" ^ String.concat "," (List.map string_of_type tlist)
  ^ " -> " ^ string_of_type t ^ ")"
  | TList(t) -> "TList(" ^ string_of_type t ^ ")"
  | TObj(props) -> "TObj(" ^ String.concat ", " (List.map (fun (s, t) ->
  s ^ ":" ^ string_of_type t) props) ^ ")"
  | TObjAccess(s, t) -> "TObjAccess(" ^ s ^ ":" ^ string_of_type t ^ ")"
  | TNum -> "TNum"
  | TChar -> "TChar"
  | TBool -> "TBool"

let sot typ =
  " [" ^ string_of_type typ ^ "]\n"

let rec string_of_expr = function
    ANumLit(n, t) -> string_of_float n ^ sot t
  | ABoolLit(b, t) -> string_of_bool b ^ sot t
  | ACharLit(c, t) -> "'" ^ Char.escaped c ^ "'" ^ sot t
  | AId(s, b, t) ->
    if b
    then s ^ " [NEW " ^ string_of_type t ^ "]\n"
    else s ^ sot t
  | AFuncCreate(formals, body, t) ->
      "(" ^ String.concat ", " (List.map (fun x -> fst x ^ sot (snd x)) formals) ^ ") -> {\n" ^
      String.concat "" (List.map string_of_stmt body) ^ "}" ^ sot t
```

```
  | AFuncCallExpr(ae, ael, t) ->
      string_of_expr ae ^ "(" ^
      String.concat ", " (List.map string_of_expr ael) ^ ")" ^ sot t
  | AObjAccess(ae, s, t)  ->
      string_of_expr ae ^ "." ^ s ^ sot t
  | AListAccess(ae1, ae2, t) ->
      string_of_expr ae1 ^ "[" ^ string_of_expr ae2 ^ "]" ^ sot t
  | AListCreate(ael, t) ->
      "[" ^ String.concat ", " (List.map string_of_expr ael) ^ "]" ^ sot t
  | ASublist(ae, aeleft, aeright, t) ->
      string_of_expr ae ^ "[" ^
      string_of_opt string_of_expr aeleft ^ ":" ^
      string_of_opt string_of_expr aeright ^ "]" ^ sot t
  | AObjCreate(props, t) ->
      "{\n" ^ String.concat ",\n" (List.map (fun prop ->
      fst prop ^ ": " ^ string_of_expr (snd prop)) props) ^
      "}" ^ sot t
  | ABinop(ae1, op, ae2, t) ->
      string_of_expr ae1 ^ (match op with
        Add -> " + "    | Sub -> " - "     | Mult -> " * "
      | Div -> " / "    | Mod -> " % "
      | Equal -> " == " | Neq -> " != "    | Less -> " < "
      | Leq -> " <= "   | Greater -> " > " | Geq -> " >= "
      | Concat -> " ^ " | And -> " && "    | Or -> " || ") ^
      string_of_expr ae2 ^ sot t
  | ANot(ae, t) ->
      "!" ^ string_of_expr ae ^ sot t

and string_of_stmt = function
    AReturn(ae, t) -> "return " ^ string_of_expr ae ^ ";" ^ sot t
  | AIf(conds, elsebody) ->
      "if" ^ string_of_cond (List.hd conds) ^ String.concat ""
        (List.map (fun x -> "\nelif" ^ string_of_cond x) (List.tl conds)) ^
        string_of_opt (fun x -> "\nelse {\n" ^ string_of_stmts x ^ "}") elsebody
  | AFor(a1, ae, a2, asl) ->
      "for (" ^ string_of_opt string_of_assign a1 ^ "; " ^
                string_of_opt string_of_expr ae ^ "; " ^
                string_of_opt string_of_assign a2 ^ ") {\n" ^
      string_of_stmts asl ^ "}"
  | AWhile(ae, asl) ->
      "while (" ^ string_of_expr ae ^ ") {\n" ^
      string_of_stmts asl ^ "}"
  | AAssign(a) -> string_of_assign a ^ ";"
  | AFuncCallStmt(ae, ael) ->
      string_of_expr ae ^ "(" ^
      String.concat ", " (List.map string_of_expr ael) ^ ");"

and string_of_stmts stmts =
  String.concat "\n" (List.map string_of_stmt stmts)

and string_of_cond (ae, asl) =
  " (" ^ string_of_expr ae ^ ") {\n" ^
  string_of_stmts asl ^ "}"

and string_of_assign (e1, e2) =
  string_of_expr e1 ^ " = " ^ string_of_expr e2

let string_of_prog prog =
```

```
  string_of_stmts prog ^ "\n"


(*************************
**** PRINT CONSTRAINTS ***
*************************)

let string_of_constraints (l : (t * t) list) =
  String.concat "\n" (List.map (fun (t1, t2) ->
  string_of_type t1 ^ "  " ^ string_of_type t2) l) ^ "\n"


(*************************
**** PRINT SUBSTITUTIONS *
*************************)

let string_of_subs (s : (string * t) list) =
  String.concat "\n" (List.map (fun (g, t) -> g ^ " " ^ string_of_type t) s) ^ "\n"


(*************************
**** PRINT INFERENCES ****
*************************)

let rec string_of_inferred_expr = function
    ANumLit(_, _) -> ""
  | ABoolLit(_, _) -> ""
  | ACharLit(_, _) -> ""
  | AId(s, b, t) ->
    if b
    then s ^ " ==> " ^ string_of_type t ^ "\n"
    else ""
  | AFuncCreate(_, body, _) ->
      String.concat "" (List.map string_of_inferred_stmt body)
  | AFuncCallExpr(ae, ael, _) ->
      string_of_inferred_expr ae ^
      String.concat "" (List.map string_of_inferred_expr ael)
  | AObjAccess(ae, _, _)  ->
      string_of_inferred_expr ae
  | AListAccess(ae1, ae2, _) ->
      string_of_inferred_expr ae1 ^
      string_of_inferred_expr ae2
  | AListCreate(ael, _) ->
      String.concat "" (List.map string_of_inferred_expr ael)
  | ASublist(ae, aeleft, aeright, _) ->
      string_of_inferred_expr ae ^
      string_of_opt string_of_inferred_expr aeleft ^
      string_of_opt string_of_inferred_expr aeright
  | AObjCreate(props, _) ->
      String.concat "" (List.map (fun (_, ae) -> string_of_inferred_expr ae) props)
  | ABinop(ae1, _, ae2, _) ->
      string_of_inferred_expr ae1 ^
      string_of_inferred_expr ae2
  | ANot(ae, _) ->
      string_of_inferred_expr ae

and string_of_inferred_stmt = function
    AReturn(ae, _) -> string_of_inferred_expr ae
```

```
  | AIf(conds, elsebody) ->
      String.concat "" (List.map (fun (ae, asl) ->
      string_of_inferred_expr ae ^ string_of_inferred_stmts asl) conds) ^
      string_of_opt string_of_inferred_stmts elsebody
  | AFor(a1, ae, a2, asl) ->
      string_of_opt string_of_inferred_assign a1 ^
      string_of_opt string_of_inferred_expr ae ^
      string_of_opt string_of_inferred_assign a2 ^
      string_of_inferred_stmts asl
  | AWhile(ae, asl) ->
      string_of_inferred_expr ae ^
      string_of_inferred_stmts asl
  | AAssign(a) ->
      string_of_inferred_assign a
  | AFuncCallStmt(ae, ael) ->
      string_of_inferred_expr ae ^
      String.concat "" (List.map string_of_inferred_expr ael)

and string_of_inferred_stmts stmts =
  String.concat "" (List.map string_of_inferred_stmt stmts)

and string_of_inferred_assign (e1, e2) =
  string_of_inferred_expr e1 ^ string_of_inferred_expr e2

let string_of_inferred_prog prog =
  string_of_inferred_stmts prog ^ "\n"
```

8.6 Code Generation

```
open Sast
open Printf
open Random

(************
  HELPERS
************)

let type_of (ae : Sast.aExpr) : Sast.t =
  match ae with
  | ANumLit(_, t) -> t
  | ABoolLit(_, t) -> t
  | ACharLit(_, t) -> t
  | AId(_, _, t) -> t
  | AFuncCreate(_, _, t) -> t
  | AFuncCallExpr(_, _, t) -> t
  | AObjAccess(_, _, t) -> t
  | AListAccess(_, _, t) -> t
  | AListCreate(_, t) -> t
  | ASublist(_, _, _, t) -> t
  | AObjCreate(_, t) -> t
  | ABinop(_, _, _, t) -> t
  | ANot(_, t) -> t

let java_from_type (ty: Sast.t) : string =
    match ty with
```

```
      | TFunc(a,b) -> "IPCFunction"
      | TList(a) -> "PCList"
      | _ ->  "PCObject"



(*****************************************************************************
  Expression and statement evaluation
*****************************************************************************)

let rec writeToFile fileName progString =
  let file = open_out ("java/" ^ fileName ^ ".java") in
    fprintf file "%s"  progString

and gen_program fileName prog = (*have a writetofile*)
  let stmtString = writeStmtList prog in
  let out = sprintf "
  public class %s
  {
      public static void main(String[] args)
      {
          DistributeClient.setSlaves(args);
%s
      }
  }
  " fileName stmtString in
  writeToFile fileName out;
  out

and writeStmtList stmtList =
  let outStr = List.fold_left (fun a b -> a ^ (gen_stmt b)) "" stmtList in
  sprintf "%s" outStr

and gen_stmt = function
    AReturn(exp, _) -> writeReturnStmt exp
  | AIf(condTupleList, elseStmt) -> writeIfStmt condTupleList elseStmt
  | AFor(asnTuple, cond, incrTuple, stmtList) ->
      writeForLoop asnTuple cond incrTuple stmtList
  | AWhile(cond, stmtList) -> writeWhileLoop cond stmtList
  | AAssign((expr1 , expr2)) -> writeAssign expr1 expr2
  | AFuncCallStmt(funcNameExpr, paramsListExpr) ->
      writeFuncCallStmt funcNameExpr paramsListExpr

and gen_expr = function
    ANumLit(flt, _)   -> writeNumLit flt
  | ABoolLit(boolLit, _) -> writeBoolLit boolLit
  | ACharLit(charLit, _) -> writeCharLit charLit
  | AId(name, typed, typ) -> writeID  name typed
  | AFuncCreate(params, body, _) -> writeFunc params body
  | AFuncCallExpr(exp, params, ty) -> writeFuncCall exp params (Some(ty))
  | AObjAccess(objName, fieldName, ty)-> writeObjectAccess objName fieldName ty
  | AListAccess(listName, idx, ty) -> writeListAccess listName idx ty
  | AListCreate(contentList,ty) -> writeListCreate contentList ty
  | ASublist(listName, leftIdx, rightIdx, _) ->
      writeSubList listName leftIdx rightIdx
  | AObjCreate(nVTplList, _) ->  writeObjCreate nVTplList
  | ABinop(ope1, op, ope2, _) -> writeBinop ope1 op ope2
  | ANot(exp, _) -> writeUnaryNot exp
```

```
(*****************************************************************************
  Specific Statement evaluation
*****************************************************************************)

and writeReturnStmt exp =
  let expStr = (gen_expr exp) in
    sprintf "return %s;" expStr

and writeIfStmt condTupleList elseStmtList =
  let string_of_tuple (condExpr, stmtList) =
    let body = writeStmtList stmtList
    and cond = gen_expr condExpr in
    sprintf "
    if (%s.<Boolean>getBase())
    {
        %s\t}" cond body
  in let ifString =
    let rec string_of_tupleList = function
        [] -> ""
      | a::[] -> string_of_tuple a ^ "\n"
      | a::tl -> (string_of_tuple a) ^ " else " ^ string_of_tupleList tl
    in string_of_tupleList condTupleList
  and elseString =
    let checkForNone str = match str with
        Some(str) -> "else{\n" ^ writeStmtList str ^ "\n}"
      | None -> ""
    in checkForNone elseStmtList
in sprintf "%s\n%s" ifString elseString

and writeForLoop asnTuple cond incrTuple stmtList =
  let asn =
    let matchTuple = function
        Some(asnTup) ->   gen_expr (fst asnTup) ^ "=" ^ gen_expr (snd asnTup)
      | None -> ""
    in matchTuple asnTuple
  and stmtString = writeStmtList stmtList
  and cond =
    let matchCond = function
        Some(cond) -> gen_expr cond
      | None -> ""
    in matchCond cond
  and incrString =
    let matchTuple = function
        Some(incrTup) ->   gen_expr (fst incrTup) ^ "=" ^ gen_expr (snd incrTup)
      | None -> ""
    in matchTuple incrTuple
  in
  sprintf "for (%s;%s.<Boolean>getBase();%s)\n{\n%s\n}\n" asn cond incrString stmtString

and writeWhileLoop cond stmtList =
  let condString = gen_expr cond
  and stmtString = writeStmtList stmtList in
    sprintf "while (%s.<Boolean>getBase())\n{\n%s\n}\n" condString stmtString

(*ASSIGNING IS SPECIAL SO WE HANDWROTE THESE WITH LOVE*)
```

```
and writeAssign expr1 expr2 =
    let lhs_type = java_from_type (type_of expr2) in
    let e2string = gen_expr expr2 in
      match expr1 with
        | AId(name, typed, typ) ->
          if typed
          then
            sprintf "%s %s = (%s)(%s);\n" lhs_type name lhs_type e2string
          else
            sprintf "%s = (%s)(%s);\n" name lhs_type e2string
        | AListAccess(listName, idx, _) ->
          let listNamestring = gen_expr listName and idxstring = gen_expr idx in
          sprintf "%s.set(%s, %s);\n" listNamestring idxstring e2string
        | AObjAccess(objName, fieldName, _)->
          let objNamestring = gen_expr objName in
          sprintf "%s.set(\"%s\", %s);\n" objNamestring fieldName e2string
        | _ -> failwith "How'd we get all the way to java with this!!!! Not a valid LHS"

and writeFuncCallStmt fNameExpr paramsListExpr =
  (writeFuncCall fNameExpr paramsListExpr None) ^ ";\n"



(*****************************************************************************
    Function handling - helper functions
*****************************************************************************)

and writeFunc params stmtList =
  let fName = function_name_gen() in
  let fileName = "java/" ^ fName ^ ".java" in
  let file = open_out fileName in
  let paramSetting = snd (List.fold_left (
                            fun a p ->
                              let count = fst a in
                              let sofar = snd a in
                              let typeName = java_from_type (snd p) in
                              let newString = typeName ^" " ^ (fst p) ^
                              " = (" ^ typeName ^ ")args[" ^ string_of_int count ^ "];\n"
                              in
                              (count +1, sofar ^ newString)
                           ) (0, "") params)
  and body = writeStmtList stmtList in
  fprintf file "
  import java.io.Serializable;
  public class %s extends IPCFunction implements Serializable
  {
      public %s() {}

      public PCObject call(PCObject... args)
      {
          %s
          %s
      }
  }
  " fName fName paramSetting body;
  sprintf "new %s()" fName

and writeFuncCall toCallExp paramsExp optCast =
```

```
  let toCall = (gen_expr toCallExp) and params = (params_to_string paramsExp) in
  match optCast with
  | Some(ty) -> sprintf "((%s)%s.call(%s))" (java_from_type ty) toCall params
  | None -> sprintf "%s.call(%s)" toCall params

and params_to_string paramsList=
  let paramsStringList = List.map gen_expr paramsList in
    let rec paramsListtoString = function
        [] -> ""
      | [a] -> sprintf("%s") a
      | hd::tl -> (sprintf("%s,") hd)^paramsListtoString tl
    in paramsListtoString paramsStringList

and pNames_to_string paramTupleList =
  let pNameList = List.map (fun (a,b) -> a) paramTupleList in
  List.fold_left (fun a b -> a^b) "" pNameList




(*****************************************************************************
    List and Object handling - helper functions
*****************************************************************************)

and writeObjectAccess objNameExpr fieldName ty=
  let objName = gen_expr objNameExpr in
  let access_type_string = java_from_type ty in
  sprintf "%s.<%s>get(\"%s\")" objName access_type_string fieldName

and writeListAccess listNameExpr idxExpr ty=
  let listName = gen_expr listNameExpr and idx = gen_expr idxExpr in
  let access_type_string = java_from_type ty in
  sprintf "%s.<%s>get(%s)" listName access_type_string idx

and writeListCreate exprList ty =
  match ty with
    | TList(t) ->
      (match t with
        | TChar ->
            let implode l =
              let res = String.create (List.length l) in
              let rec imp i = function
                | [] -> res
                | c :: l -> res.[i] <- c; imp (i + 1) l
              in
              imp 0 l
            in
            let char_from_expr = function
              | ACharLit(c, _) -> c
              | _ -> failwith "NEVER: list of characters has non-character"
            in
            let word = implode (List.map char_from_expr exprList) in
            sprintf "new PCList(\"%s\")" word
        | _ ->
            let concatAdds = (fun a b -> a^(sprintf(".add(%s)") b))
            and list_of_strings = List.map gen_expr exprList in
            List.fold_left concatAdds "new PCList()" list_of_strings
      )
```

```
    | _ -> failwith "NEVER: trying to generate a list from a non list type"

and writeSubList listNameExpr startIdxExpr endIdxExpr =
  let listName = gen_expr listNameExpr in
  let startIdx =
    let det = function
        Some(x) -> gen_expr x
      | None -> "new PCObject(0)"
    in det startIdxExpr
  and endIdx =
    let det = function
        Some(x) -> gen_expr x
      | None -> sprintf "new PCObject(%s.size()-1)" listName
    in det endIdxExpr
  in sprintf "%s.subList(%s,%s)" listName startIdx endIdx

and writeObjCreate kvt_list =
  let string_of_tuple (k , vExpr)  =
    let v = gen_expr vExpr in
    sprintf ".set(\"%s\", %s)" k v
  in let stringList = List.map string_of_tuple kvt_list; in
    List.fold_left (fun a b -> a^b) "new PCObject()" stringList



(*****************************************************************************
    Binop and Not Handling - helper functions
*****************************************************************************)

and writeUnaryNot boolExpr =
  let boolObj = gen_expr boolExpr in
  sprintf "!%s" boolObj

and writeBinop expr1 op expr2 =
  let e1 = gen_expr expr1 and e2 = gen_expr expr2 in
    let writeBinopHelper e1 op e2 = match op with
        Add  -> sprintf "new PCObject(%s.<Double>getBase() + %s.<Double>getBase())" e1 e2
      | Sub  -> sprintf "new PCObject(%s.<Double>getBase() - %s.<Double>getBase())" e1 e2
      | Mult -> sprintf "new PCObject(%s.<Double>getBase() * %s.<Double>getBase())" e1 e2
      | Div  -> sprintf "new PCObject(%s.<Double>getBase() / %s.<Double>getBase())" e1 e2
      | Mod -> sprintf "new PCObject(%s.<Double>getBase() %% %s.<Double>getBase())" e1 e2
      | Less -> sprintf "new PCObject(%s.<Double>getBase() < %s.<Double>getBase())" e1 e2
      | Leq -> sprintf "new PCObject(%s.<Double>getBase() <= %s.<Double>getBase())" e1 e2
      | Greater -> sprintf "new PCObject(%s.<Double>getBase() > %s.<Double>getBase())" e1 e2
      | Geq -> sprintf "new PCObject(%s.<Double>getBase() >= %s.<Double>getBase())" e1 e2
      | And -> sprintf "new PCObject(%s.<Boolean>getBase() && %s.<Boolean>getBase())" e1 e2
      | Or -> sprintf "new PCObject(%s.<Boolean>getBase() || %s.<Boolean>getBase())" e1 e2
      | Equal -> sprintf "new PCObject(%s.equals(%s))" e1 e2
      | Neq -> sprintf "new PCObject(!(%s.equals(%s)))" e1 e2
      | Concat -> sprintf "new PCList(%s,%s)" e1 e2
    in writeBinopHelper e1 op e2


(*****************************************************************************
    Id handling - helper function
*****************************************************************************)

and writeID idName ty =
```

```
  let newName = (match idName with
  | "rec" -> "this"
  | "print" | "printFile" | "read" | "readFile" | "download" | "numToString" | "numFromString"
  -> sprintf "IO.get(\"%s\")" idName
  | "distribute" -> "DistributeClient.distribute"
  | _ -> idName) in
  match ty with
    true -> sprintf "PCObject %s" newName
  | false -> sprintf "%s" newName


(*****************************************************************************
    Literal expression handling - helper functions
*****************************************************************************)

and writeNumLit numLit =
  if floor(numLit) = numLit
  then sprintf "new PCObject(%d)" (int_of_float numLit)
  else sprintf "new PCObject(%f)" numLit

and writeBoolLit boolLit =
  sprintf "new PCObject(%b)" boolLit

and writeCharLit charLit =
  let fChar = Char.escaped charLit in
  sprintf "new PCObject('%s')" fChar


(*****************************************************************************
    Function Name Generation - helper functions
*****************************************************************************)

and function_name_gen() =
  Random.self_init();
  let x = (Random.int 100000) in
  sprintf "function_%d" x
```

8.7 pc.ml

```
(* pubCrawl compiler
  1. give stdin to scanner, get tokens
  2. give tokens to parser, get AST
  3. give AST to analyzer, get semantic tree
  4. give semantic tree to java converter, get java tree
  5. give java tree to java code generator, get java code
  6. give java code to java compiler, get executable
  7. run java executable
*)

type action = Ast | Sast | Java

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast); ("-s", Sast); ("-j", Java);]
  else Java in
```

```
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  match action with
  | Ast ->
      print_string (Ast.string_of_prog program)
  | Sast ->
      let ap = Analyzer.infer_prog program in
      print_string ("\n" ^ Sast.string_of_inferred_prog ap)
  | Java ->
      let ap = Analyzer.infer_prog program in
      let _ = Javagen.gen_program "output" ap in
      print_string "Success! Compiled to java/output.java\n"
```

8.8 PCObject.java

```
import java.io.Serializable;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class PCObject implements Serializable
{
    private final HashMap<String, Object> _props = new HashMap<String, Object>();
    private final static String _base = "050FCAC0-610C-4CF6-9CC2-5EA5A40C3155";


    // Constructors
    public PCObject() {
        set(_base, this);
    }
    public PCObject(double num) {
        set(_base, num);
    }
    public PCObject(boolean b) {
        set(_base, b);
    }

    public PCObject(char c) {
        set(_base, c);
    }

    // Check if contains key
    public boolean contains(String key) {
        return _props.containsKey(key);
    }

    // Set value and key
    public PCObject set(String key, Object value) {
        _props.put(key, value);
        return this;
    }

    private Object getObj(String key) {
        return _props.containsKey(key) ? _props.get(key) : null;
    }
```

```
    @SuppressWarnings("unchecked")
    public <T> T get(String key) {
        // this is an unchecked cast and that's ok, it might fail at run time
        return (T) getObj(key);
    }

    // Get whatever is stored in the PCObject
    public <T> T getBase() {
        return get(_base);
    }

    //we know from type inference these must be of the same type
    //so we only ever have to check one
    public boolean equals(PCObject other) {
        //first let's check if these are primitives
        if(_props.containsKey(_base)){
            if(_props.get(_base) instanceof Double){
                double EPSILON = 0.0000001;
                return Math.abs(this.<Double>getBase() - other.<Double>getBase()) <= EPSILON;
            }
            else if (_props.get(_base) instanceof Boolean){
                return this.<Boolean>getBase() == other.<Boolean>getBase();
            }
            else if (_props.get(_base) instanceof Character){
                return this.<Character>getBase() == other.<Character>getBase();
            }
            else return false; //hopefully never see this guy
        }
        else{
            //else let's check each guy
            Iterator it = _props.entrySet().iterator();
            while (it.hasNext())
            {
                Map.Entry entry = (Map.Entry) it.next();
                String key = (String)entry.getKey();
                PCObject val = (PCObject)entry.getValue();
                if(!other.contains(key)){
                    return false;
                }
                else{
                    PCObject mine =val;
                    PCObject his = other.<PCObject>get(key);
                    if(!mine.equals(his)){
                        return false;
                    }
                }
            }
            return true;
        }
    }

}
```

8.9 PCList.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;

public class PCList extends PCObject implements Iterable<PCObject>
{
    private final ArrayList<PCObject> _list = new ArrayList<PCObject>();

    public PCList() {

    }

    // Constructors
    public PCList(List<PCObject> items) {
        for (PCObject item : items) {
            _list.add(item);
        }
    }

    public PCList(String mystring) {
        char[] myarray = mystring.toCharArray();
        for(int i = 0; i < myarray.length; i++){
            _list.add(new PCObject(myarray[i]));
        }
    }

    public PCList(PCObject obj1, PCObject obj2)
    {
        PCList a = (PCList) obj1;
        PCList b = (PCList) obj2;

        for (PCObject o : a) {
            _list.add(o);
        }
        for (PCObject o : b) {
            _list.add(o);
        }
    }

    // Return size of list
    public int size() {
        return _list.size();
    }

    @SuppressWarnings("unchecked")
    public <T> T get(int idx) {
        return (T) _list.get(idx);
    }

    @SuppressWarnings("unchecked")
    public <T> T get(PCObject idx) {
        return get((int)idx.<Double>getBase().doubleValue());
    }

    public void set(int idx, PCObject val) {
        _list.set(idx, val);
    }
```

```
    // Add to the list
    public PCList add(PCObject val) {
        _list.add(val);
        return this;
    }

    public void removeAt(int idx) {
        _list.remove(idx);
    }

    // Sublist with specified starting and ending index
    public PCList subList(PCObject start, PCObject end) {
        return new PCList(_list.subList(start.<Double>getBase().intValue(),
        end.<Double>getBase().intValue() + 1));
    }

    // Iterate the list
    public Iterator<PCObject> iterator() {
        return _list.iterator();
    }
}
```

8.10 IPCFunction

```
public abstract class IPCFunction extends PCObject
{
    public abstract PCObject call(PCObject... args);
}
```

8.11 DistributeClient.java

```
import java.lang.Exception;
import java.lang.SecurityManager;
import java.lang.String;
import java.lang.System;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class DistributeClient {

    private static FileServer server = new FileServer();
    private static ArrayList<Compute> slaves;
    private static String[] slaveList;
```

```
    private static boolean initialized = false;

    public static void setSlaves(String[] args){
        slaveList = args;
    }

    private static void getRegistries(String... hosts){
        slaves = new ArrayList<Compute>();
        System.setProperty("java.security.policy", "client.policy");
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        Registry registry;
        Compute comp;
        try{
            for(int i=0; i< hosts.length-1; i = i+2) {
               registry = LocateRegistry.getRegistry(hosts[i], Integer.parseInt(hosts[i+1]));
               comp = (Compute) registry.lookup("Compute");
               slaves.add(comp);
            }
        }
        catch(Exception e) {
            System.out.println("Failed to connect\n\n");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static IPCFunction distribute = new IPCFunction() {
        @Override
        public PCObject call(PCObject... args) {
            PCList toProcess = (PCList)args[0];
            final IPCFunction function = (IPCFunction)args[1];

            if(!initialized) {
                getRegistries(slaveList);
                server.start();
                initialized = true;
            }

            ArrayList<PCObject> output = new ArrayList<PCObject>();
            Iterator<Compute> slave_it = slaves.iterator();

            try {
                ExecutorService exec = Executors.newFixedThreadPool(toProcess.size());

                ArrayList<Future<PCObject>> futures = new ArrayList<Future<PCObject>>();
                for(final PCObject param: toProcess){

                    if(!slave_it.hasNext()){
                        slave_it = slaves.iterator();
                    }

                    final Compute slave = slave_it.next();
                    Future<PCObject> future = exec.submit(new Callable<PCObject>() {
                        @Override
                        public PCObject call() throws RemoteException {
```

```
                            return slave.callFunction(function, param);
                        }
                    });
                    futures.add(future);
                }

                for (Future<PCObject> future : futures) {
                    output.add(future.get());
                }

                exec.shutdown();

            }
            catch(Exception e){
                e.printStackTrace();
                System.exit(1);
            }
            finally {
                server.stop();
            }

            return new PCList(output);
        }
    };
}
```

8.12 DistributeServer.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class DistributeServer implements Compute {

    public DistributeServer(){
        super();
    }

    public PCObject callFunction(IPCFunction function, PCObject param){
        return function.call(param);
    }

    public static void main(String[] args){
        System.setProperty("java.security.policy", "server.policy");
        System.setProperty("java.rmi.server.useCodebaseOnly","false");
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            String name = "Compute";

            Compute engine = new DistributeServer();
            Compute stub =
```

```
                    (Compute) UnicastRemoteObject.exportObject(engine, 0);

            Registry registry = LocateRegistry.createRegistry(Integer.parseInt(args[0]));

            registry.rebind(name, stub);

            System.out.println("Server started");
        } catch (Exception e) {
            System.err.println("Server exception:");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

8.13 Compute.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote
{
    PCObject callFunction(IPCFunction function, PCObject param) throws RemoteException;
}
```

8.14 FileServer.java

```
import java.io.*;
import java.net.InetSocketAddress;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class FileServer {

    private HttpServer server;

    public void start() {
        try {
            server = HttpServer.create(new InetSocketAddress(8782), 0);
            server.createContext("/", new MyHandler());
            server.setExecutor(null); // creates a default executor
            server.start();
        }
```

```
        catch(IOException e) {
            e.printStackTrace();
        }
    }

    public void stop() {
        server.stop(0);
    }

    public static void main(String[] args) throws Exception {
        new FileServer().start();
    }

    static class MyHandler implements HttpHandler {
        public void handle(HttpExchange t) throws IOException {
            String response = t.getRequestURI().toString();
            String filePath = response.substring(1,response.length());
            byte[] fileInfo = readSmallBinaryFile(filePath);
            t.sendResponseHeaders(200, fileInfo.length);
            OutputStream os = t.getResponseBody();
            os.write(fileInfo);
            os.close();
        }
    }

    private static byte[] readSmallBinaryFile(String aFileName) throws IOException {
        Path path = Paths.get(aFileName);
        return Files.readAllBytes(path);
    }

    private static String readFileAsString(String filePath) throws IOException {
        StringBuffer fileData = new StringBuffer();
        BufferedReader reader = new BufferedReader(
                new FileReader(filePath));
        char[] buf = new char[1024];
        int numRead=0;
        while((numRead=reader.read(buf)) != -1){
            String readData = String.valueOf(buf, 0, numRead);
            fileData.append(readData);
        }
        reader.close();
        return fileData.toString();
    }
}
```

8.15 List.java

```
public class List
{
  private static PCObject _obj = new PCObject();

  static {

    _obj.set("length", new IPCFunction(){
      @Override
```

```
    public PCObject call(PCObject... args){
      PCList list = (PCList)args[0];
      return new PCObject(list.size());

    }
  });

  _obj.set("add", new IPCFunction(){
    @Override
    public PCObject call(PCObject... args){
      PCList list = (PCList)args[0];
      list.add(args[1]);
      return list;
    }
  });


  _obj.set("remove", new IPCFunction(){
    @Override
    public PCObject call(PCObject... args){
      PCList list = (PCList)args[0];
      int idx = args[1].<Double>getBase().intValue();
      list.removeAt(idx);
      return list;
    }
  });

  _obj.set("map", new IPCFunction() {
    @Override
    public PCObject call(PCObject... args) {
      PCList list = (PCList)args[0];
      IPCFunction mapper = (IPCFunction)args[1];
      PCList newList = new PCList();
      for (PCObject o : list) {
        newList.add(mapper.call(o));
      }
      return newList;
    }
  });

  _obj.set("where", new IPCFunction() {
    @Override
    public PCObject call(PCObject... args) {
      PCList list = (PCList)args[0];
      IPCFunction filter = (IPCFunction)args[1];
      PCList newList = new PCList();
      for (PCObject o : list) {
        if (filter.call(o).<Boolean>getBase()) {
          newList.add(o);
        }
      }
      return newList;
    }
  });

  _obj.set("find", new IPCFunction(){
    @Override
    //adapted from algs4.cs.princeton.edu/53substrings
```

```
    public PCObject call(PCObject... args){
      PCList txt = (PCList)args[0];
      PCList pat = (PCList)args[1];

          int N = txt.size();
      int M = pat.size();

          for (int i = 0; i < N; i++) {
            int srcIdx = i;
            int j;
              for (j = 0; j < M; j++) {
                  if (!txt.<PCObject>get(srcIdx).equals(pat.<PCObject>get(j))) {
                      break;
                  }
                  if (j == M - 1) {
                    return new PCObject(i);
                  }
                  if (++srcIdx == N) {
                    break;
                  }
              }
          }

          return new PCObject(-1); // not found
      }
    });

    _obj.set("split", new IPCFunction(){
      @Override
      public PCObject call(PCObject... args){
        PCList list = (PCList)args[0];
        PCObject wedge = args[1];
        PCList output = new PCList();
        int lBound = 0;
        int uBound = 0;

        for(PCObject element : list){
          if(element.equals(wedge)){
            output.add(list.subList(new PCObject(lBound),new PCObject(uBound-1)));
            lBound = uBound+1;
          }
          uBound++;
        }

        if (uBound > lBound) {
          output.add(list.subList(new PCObject(lBound),new PCObject(uBound-1)));
        }

        return output;
      }
    });

    _obj.set("range", new IPCFunction(){
      @Override
      public PCObject call(PCObject... args){
        PCList output = new PCList();
        int lLimit = args[0].<Double>getBase().intValue();
        int uLimit = args[1].<Double>getBase().intValue();
```

```
        for(;lLimit<=uLimit;lLimit ++){
          output.add(new PCObject(lLimit));
        }
        return output;
      }
    });

    _obj.set("populate", new IPCFunction(){
      @Override
      public PCObject call(PCObject... args){
        PCList output = new PCList();
        PCObject item = args[0];
        int num = args[1].<Double>getBase().intValue();
        for(int i = 0; i < num; i++) {
          output.add(item);
        }
        return output;
      }
    });
  }

    public static <T> T get(String key) {
      return _obj.<T>get(key);
    }
}
```

8.16 IO.java

```
import java.io.*;
import java.util.Iterator;
import java.net.URL;

public class IO
{

  // Converts a number to a string and return that string
    private static String getStringOfObj(PCObject obj){
        Object myobject = obj.<Object>getBase();
        if(myobject instanceof Double){
            Double mynum = obj.<Double>getBase();
            if(Math.floor(mynum) == mynum){
                return (new Integer(mynum.intValue())).toString();
            }
            else {
                return mynum.toString();
            }
        }
        else {
            return myobject.toString();
        }
    }

  private static PCObject _obj = new PCObject();

  static {
```

```
// Read from stdin
_obj.set("read", new IPCFunction(){
  @Override
  public PCObject call(PCObject... args){
        try {
          BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
            return new PCList(inStream.readLine());
          }
        catch(IOException e) {
            e.printStackTrace();
        }
        return new PCList();
  }
});

// Read from a file
_obj.set("readFile", new IPCFunction(){
  @Override
  public PCObject call(PCObject... args){
    PCList listOfChars = (PCList)args[0];

        PCList toReturn = new PCList();
        String fileName = new String();

        for (PCObject element : listOfChars) {
            fileName += element.<Character>getBase();
        }
        BufferedReader br;
        try {
            br = new BufferedReader(new FileReader(fileName));
            String line = br.readLine();
            while (line != null) {
                toReturn.add(new PCList(line));
                line = br.readLine();
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        return toReturn;
  }
});

// Download given url
_obj.set("download", new IPCFunction() {
  @Override
  public PCObject call(PCObject... args){
        String fileName = new String();
        PCList listOfChars = (PCList)args[0];
        for (PCObject element : listOfChars) {
            fileName += element.<Character>getBase();
        }

        String result = "";
        BufferedReader in = null;
        try {
            URL myURL = new URL(fileName);
```

```
                in = new BufferedReader(new InputStreamReader(myURL.openStream()));
                String line = in.readLine();
                while (line != null) {
                    result += line + "\n";
                    line = in.readLine();
                }
            }
            catch (IOException e) {}
            finally {
                if (in != null) {
                    try {
                        in.close();
                    }
                    catch(IOException ex) {}
                }
            }

            return new PCList(result);
      }
    });

    // Print to stdout
    _obj.set("print", new IPCFunction(){
      @Override
      public PCObject call(PCObject... args){
        PCObject toPrint = (PCObject)args[0];
        if(toPrint instanceof PCList) {
              PCList mylist = (PCList) toPrint;
                for(PCObject obj : mylist) {
                    System.out.print(getStringOfObj(obj));
                }
            }
            else{
                System.out.print(getStringOfObj(toPrint));
            }
            System.out.println();
            return toPrint;

          }
    });

    // Print to a file
    _obj.set("printFile", new IPCFunction(){
      @Override
      public PCObject call(PCObject... args){
        PCObject toPrint  = (PCObject)args[0];
        PCList listOfChars = (PCList)args[1];
        String fileName = new String();

        for(PCObject element : listOfChars) {
            fileName += element.<Character>getBase();
        }

        try {
            File f = new File(fileName);
            if (!f.exists()) {
                f.createNewFile();
            }
```

```
            PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(fileName)));

            if(toPrint instanceof PCList) {
                PCList mylist = (PCList) toPrint;
                for(PCObject obj : mylist) {
                    out.print(getStringOfObj(obj));
                }
            }
            else{
                out.print(getStringOfObj(toPrint));
            }
            out.close();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        return toPrint;


      }
    });

    // Convert a number to a string and return a PCObject
    _obj.set("numToString", new IPCFunction() {
      @Override
      public PCObject call(PCObject... args){
        PCObject origNum = args[0];
            String result = getStringOfObj(origNum);
            return new PCList(result);
      }
    });

    // Convert a string to a number and return a PCObject
    _obj.set("numFromString", new IPCFunction() {
      @Override
      public PCObject call(PCObject... args){
        String s = new String();
        for(PCObject element : (PCList)args[0]) {
            s += element.<Character>getBase();
        }
        return new PCObject(Double.parseDouble(s));
      }
    });

  }

    public static IPCFunction get(String key) {
      return _obj.<IPCFunction>get(key);
    }
}
```

8.17 Makefile

```
OBJS = ast.cmo sast.cmo parser.cmo scanner.cmo analyzer.cmo javagen.cmo pc.cmo

SRCS = $(wildcard *.*)
```

```
JAVA_SRCS = $(shell find java -name '*')
TESTS := $(shell find tests -name '*.pc' -o -name "*.out")

TARFILES = Makefile $(SRCS) $(JAVA_SRCS) $(TESTS)

pc : $(OBJS)
  ocamlc -g -o pc $(OBJS)

scanner.ml : scanner.mll
  ocamllex scanner.mll

parser.ml parser.mli : parser.mly
  ocamlyacc parser.mly

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

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

pubCrawl.tar.gz : $(TARFILES)
  tar czf pubCrawl.tar.gz $(TARFILES)

.PHONY : cleanJava
cleanJava :
  rm -f java/function_* java/output.java java/*.class

.PHONY : clean
clean : cleanJava
  rm -f pc parser.ml parser.mli parser.output scanner.ml \
  *.cmo *.cmi *.out *.diff *.log *.gz

.PHONY : all
all : clean pc


# Generated by ocamldep *.ml *.mli
analyzer.cmo : sast.cmo ast.cmo
analyzer.cmx : sast.cmx ast.cmx
javagen.cmo : sast.cmo
javagen.cmx : sast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
pc.cmo : scanner.cmo sast.cmo parser.cmi javagen.cmo ast.cmo analyzer.cmo
pc.cmx : scanner.cmx sast.cmx parser.cmx javagen.cmx ast.cmx analyzer.cmx
sast.cmo : ast.cmo
sast.cmx : ast.cmx
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
parser.cmi : ast.cmo
```

8.18 compile.sh

```
#!/bin/bash

make cleanJava
make
```

```
./pc < $1
cd java
javac *.java
```

8.19 run.sh

```
#!/bin/bash

./compile.sh $1 >/dev/null
shift
cd java

# get IP address for mac or ubuntu
if [[ "$(ifconfig en0 2>/dev/null)" == *error* ]]; then
  MY_IP=$(ifconfig | awk -F':' '/inet addr/&&!/127.0.0.1/{split($2,_," ");print _[1]}')
else
  MY_IP=$(ifconfig en0 | grep inet | grep -v inet6 | awk '{print $2}')
fi

java -Djava.rmi.server.codebase=http://$MY_IP:8782/ output $@
```

8.20 slave.sh

```
#!/bin/bash

cd java
java -jar DistributeServer.jar $1
```

8.21 Tests

8.21.1 testall.sh

8.21.2 test-writefile1.pc

```
printFile("cool!", "tmp.class"); // name it .class so make clean picks it up :)
print(readFile("tmp.class")[0]);
```

8.21.3 test-writefile1.out

```
cool!
```

8.21.4 test-while1.pc

```
i = 5;
while (i > 0) {
  print(i);
  i = i - 1;
```

```
}

print(42);
```

8.21.5 test-while1.out

```
5
4
3
2
1
42
```

8.21.6 test-var1.pc

```
a = 42;
print(a);
```

8.21.7 test-var1.out

```
42
```

8.21.8 test-typeinference2.pc

```
x = (f,a) -> {return f(a);};

result1 = x(a->{return a+1;}, 5);

result2 = x(a->{return a^"second";}, "first");

print(result1);
print(result2);
```

8.21.9 test-typeinference2.out

```
6
firstsecond
```

8.21.10 test-typeinference.pc

```
a = (() -> { return [{a:1},{a:2},{a:3}]; }())[1].a;
x1 = x -> { return x; };
x2 = List.map([1,2,3], x1);
x3 = (o) -> { return List.map("hey", o.a.b[0]); };
obj = {
    i: x1,
        wrap: (x) -> { return { a: { b: [x] } }; }
};
x4 = x3(obj.wrap(obj.i));
```

```
print(x2);
print(x4);
```

8.21.11 test-typeinference.out

```
123
hey
```

8.21.12 test-tostring.pc

```
x = 5;
y = numToString(x);
print(y);
```

8.21.13 test-tostring.out

```
5
```

8.21.14 test-string1.pc

```
myString = "HELLOWORLD";

print(myString[3:5]);
```

8.21.15 test-string1.out

```
LOW
```

8.21.16 test-scanner1.pc

```
print("'");
print('\'');
print('"');
print("\"");
print("\n\ttabbed\\");
```

8.21.17 test-scanner1.out

```
'
'
"
"

        tabbed\
```

8.21.18 test-readfile1.pc

```
print(readFile("../tests/test-readfile1.pc")[0]);
```

8.21.19 test-readfile1.out

```
print(readFile("../tests/test-readfile1.pc")[0]);
```

8.21.20 test-ops1.pc

```
print(1 + 2);
print(1 - 2);
print(1 * 2);
print(100 / 2);
print(99);
print(1 == 2);
print(1 == 1);
print(99);
print(1 != 2);
print(1 != 1);
print(99);
print(1 < 2);
print(2 < 1);
print(99);
print(1 <= 2);
print(1 <= 1);
print(2 <= 1);
print(99);
print(1 > 2);
print(2 > 1);
print(99);
print(1 >= 2);
print(1 >= 1);
print(2 >= 1);
print(4 % 2);
print(5 % 2);
print("new" ^ "Word");
print(['a','b'] ^ ['c', 'd']);
```

8.21.21 test-ops1.out

```
3
-1
2
50
99
false
true
99
true
false
99
true
false
99
true
```

```
true
false
99
false
true
99
false
true
true
0
1
newWord
abcd
```

8.21.22 test-obj3.pc

```
myObj = {
        value1: "hey",
        value2: 9
};

print(myObj.value1);
print(myObj.value2);
```

8.21.23 test-obj3.out

```
hey
9
```

8.21.24 test-obj2.pc

```
myObj = {
        value1: 7,
        value2: {
                value3: {
                        value4: {
                                value5: "hey!"
                        }
                }
        }
};

print(myObj.value2.value3.value4.value5);

myObj.value2.value3 = {
        value4: {
                value5: "dude!"
        }
};

print(myObj.value2.value3.value4.value5);
```

8.21.25 test-obj2.out

```
hey!
dude!
```

8.21.26 test-obj1.pc

```
myObj = {
        value1: 7,
        value2: 9
};

print(myObj.value1);
print(myObj.value2);
```

8.21.27 test-obj1.out

```
7
9
```

8.21.28 test-mutual-rec.pc

```
//subtracts 1
func1 = (x,f2) -> {
        if(x < 1){
                return 1;
        }
        smaller = x - 1;
        return x + f2(smaller, rec);
};

//cuts it by 2
func2 = (y,f1) -> {
        if(y < 1){
                return 1;
        }
        smaller = y/2;
        return y + f1(smaller, rec);
};
result = func1(100,func2);

print(result);
```

8.21.29 test-mutual-rec.out

```
380.6875
```

8.21.30 test-if4.pc

```
if (false) {
  print(42);
}
elif (false) {
```

```
  print(9);
}
elif (true) {
  print(8);
}
else {
  print(88);
}

print(17);
```

8.21.31 test-if4.out

```
8
17
```

8.21.32 test-if3.pc

```
if (false) {
  print(42);
}

print(17);
```

8.21.33 test-if3.out

```
17
```

8.21.34 test-if2.pc

```
if (true) {
  print(42);
}
else {
  print(8);
}

print(17);
```

8.21.35 test-if2.out

```
42
17
```

8.21.36 test-if1.pc

```
if ((() -> { return false; })() || true) {
  print(42);
}
```

```
print(17);
```

8.21.37 test-if1.out

```
42
17
```

8.21.38 test-hello.pc

```
print(42);
print(71);
print(1);
```

8.21.39 test-hello.out

```
42
71
1
```

8.21.40 test-hello-world.pc

```
numToLetters = a -> {
        myLetters = "abcdefghijklmnopqrstuvwxyz ";
        return myLetters[a];
};


mynums = [7, 4, 11, 11, 14, 26, 22, 14, 17, 11, 3];

print(List.map(mynums, numToLetters) ^ "!");
```

8.21.41 test-hello-world.out

```
hello world!
```

8.21.42 test-hello-world-distributed.pc

```
numToLetters = a -> {
        myLetters = "abcdefghijklmnopqrstuvwxyz ";
        return myLetters[a];
};


mynums = [7, 4, 11, 11, 14, 26, 22, 14, 17, 11, 3];

print(distribute(mynums, numToLetters) ^ "!");
```

8.21.43 test-hello-world-distributed.out

```
hello world!
```

8.21.44 test-gcd.pc

```
gcd = (a, b) -> {
  while (a != b) {
    if (a > b) {
      a = a - b;
        }
        else {
      b = b - a;
        }
  }
  return a;
};

print(gcd(2,14));
print(gcd(3,15));
print(gcd(99,121));
```

8.21.45 test-gcd.out

```
2
3
11
```

8.21.46 test-func4.pc

```
fun = (x, y) -> {
  print(x+y);
  return 0;
};

i = 1;

fun(i, i+1);

print(i);
```

8.21.47 test-func4.out

```
3
1
```

8.21.48 test-func3.pc

```
printem = (a,b,c,d) -> {
  print(a);
  print(b);
  print(c);
```

```
  print(d);
  return true;
};

printem(42,17,192,8);
```

8.21.49 test-func3.out

```
42
17
192
8
```

8.21.50 test-func2.pc

```
/* Bug noticed by Pin-Chin Huang */

fun = (x, y) -> {
  return 0;
};

i = 1;

fun(i, i+1);

print(i);
```

8.21.51 test-func2.out

```
1
```

8.21.52 test-func1.pc

```
add = (a, b) -> {
  return a + b;
};

a = add(39, 3);
print(a);

b = 7;
print(b);
```

8.21.53 test-func1.out

```
42
7
```

8.21.54 test-for1.pc

```
for (i = 0 ; i < 5 ; i = i + 1) {
  print(i);
}
print(42);
```

8.21.55 test-for1.out

```
0
1
2
3
4
42
```

8.21.56 test-fib.pc

```
fib = (x) -> {
  if (x < 2) {
          return 1;
  }
  return rec(x-1) + rec(x-2);
};

print(fib(0));
print(fib(1));
print(fib(2));
print(fib(3));
print(fib(4));
print(fib(5));
```

8.21.57 test-fib.out

```
1
1
2
3
5
8
```

8.21.58 test-download1.pc

```
google = download("http://www.google.com");
idx = List.find(google, "doctype");
result = google[0:idx-1];
print(result);
```

8.21.59 test-download1.out

```
<!
```

8.21.60 test-doublerec.pc

```
outer = (x) -> {
  if (x < 2) {
          return 1;
  }

  y = rec(x - 1);

  inner = (x) -> {
          if (x < 1) {
                  return 0;
          }
          return rec(x - 1) + rec(x - 2);
  };

  return inner(y);
};

print(outer(9));
```

8.21.61 test-doublerec.out

```
0
```

8.21.62 test-distribute3.pc

```
myObj = {
  gcd: (list) -> {
    a = list[0];
    b = list[1];
    while (a != b) {
      if (a > b) {
        a = a - b;
             }
             else {
        b = b - a;
             }
    }
    return a;
  },

  listOflists : [[2, 14], [3, 15], [99, 121]]

};



results = distribute(myObj.listOflists, myObj.gcd);

print(results[0]);
print(results[1]);
print(results[2]);
```

8.21.63 test-distribute3.out

```
2
3
11
```

8.21.64 test-distribute2.pc

```
gcd = (list) -> {
  a = list[0];
  b = list[1];
  while (a != b) {
    if (a > b) {
      a = a - b;
        }
        else {
      b = b - a;
        }
  }
  return a;
};

listOflists = [[2, 14], [3, 15], [99, 121]];

results = distribute(listOflists, gcd);

print(results[0]);
print(results[1]);
print(results[2]);
```

8.21.65 test-distribute2.out

```
2
3
11
```

8.21.66 test-distribute1.pc

```
list1 = [1, 2, 3];

fun = (x) -> {
  return x+1;
};

results = distribute(list1, fun);

print(results[0]);
print(results[1]);
print(results[2]);
```

8.21.67 test-distribute1.out

```
2
3
```

```
4
```

8.21.68 test-crazy-obj-generics.pc

```
f = (o) -> {
        x = o.id.a;
        return numToString(o.id.b) ^ o.name;
};

//should work!
myObj = {
        id: {
                a: 5,
                b: 10
        },
        name: "heyo"
};

print(f(myObj));
```

8.21.69 test-crazy-obj-generics.out

```
10heyo
```

8.21.70 test-collection3.pc

```
list1 = ["list", "of", "strings"];

print(list1[0]);
print(list1[2]);
```

8.21.71 test-collection3.out

```
list
strings
```

8.21.72 test-collection2.pc

```
list1 = [1, 2, 3, 4, 5];

list2 = list1[:];
list3 = list2[0:2];

print(list2[4]);

print(list3[0]);
print(list3[1]);
print(list3[2]);
```

8.21.73 test-collection2.out

```
5
1
2
3
```

8.21.74 test-collection1.pc

```
list1 = [1, 2, 3, 4, 5];
list2 = [6, 7, 8];

print(list1[0]);

print(list2[2]);
```

8.21.75 test-collection1.out

```
1
8
```

8.21.76 test-bool3.pc

```
if (!true) {
        print("hey");
}
if (!false) {
        print("dude");
}
```

8.21.77 test-bool3.out

```
dude
```

8.21.78 test-bool2.pc

```
if (true || true) {
        print("hey");
}
if (true || false) {
        print("dude");
}
if (false || false) {
        print("whats up");
}
```

8.21.79 test-bool2.out

```
hey
dude
```

8.21.80 test-bool1.pc

```
if (true && true) {
        print("hey");
}

if (true && false) {
        print("dude");
}
```

8.21.81 test-bool1.out

```
hey
```

8.21.82 test-arith2.pc

```
print(1 + 2 * 3 + 4);
```

8.21.83 test-arith2.out

```
11
```

8.21.84 test-arith1.pc

```
print(39 + 3);
```

8.21.85 test-arith1.out

```
42
```

8.21.86 test-ListWhere2.pc

```
list1 = ["hey", "dude", "hi"];

result = List.where(list1, (x) -> { return x[0] == 'h'; });

print(result[0]);
print(result[1]);
```

8.21.87 test-ListWhere2.out

```
hey
hi
```

8.21.88 test-ListWhere1.pc

```
list1 = [1, 2, 3, 4, 5, 6];

fun = (x) -> { return x%2 == 0; };

result1 = List.where(list1, fun);

print(result1[0]);
print(result1[1]);
print(result1[2]);
```

8.21.89 test-ListWhere1.out

```
2
4
6
```

8.21.90 test-ListSplit2.pc

```
sentence = [1,2,3,1,4,5,6,1];

result = List.split(sentence, 1);

result1 = result[0];
result2 = result[1];
result3 = result[2];

print(result2[0]);
print(result2[1]);
print(result3[0]);
print(result3[1]);
print(result3[2]);
```

8.21.91 test-ListSplit2.out

```
2
3
4
5
6
```

8.21.92 test-ListSplit1.pc

```
sentence = "this is a sentence that has words";

result = List.split(sentence, ' ');

print(result[0]);
print(result[1]);
print(result[2]);
print(result[3]);
print(result[4]);
```

```
print(result[5]);
print(result[6]);
```

8.21.93 test-ListSplit1.out

```
this
is
a
sentence
that
has
words
```

8.21.94 test-ListRemove2.pc

```
list1 = [1, 2, 3, 4, 5, 6];

fun = x -> {
        if(x%2 == 0){
                return true;
        }
        return false;
};

result1 = List.where(list1,fun);

print(result1[0]);
print(result1[1]);
print(result1[2]);
```

8.21.95 test-ListRemove2.out

```
2
4
6
```

8.21.96 test-ListRemove1.pc

```
list1 = [1, 2, 3, 4, 5, 6];

list1 = List.remove(list1, 0);
list1 = List.remove(list1, 0);

print(list1[0]);
```

8.21.97 test-ListRemove1.out

```
3
```

8.21.98 test-ListRange2.pc

```
results = List.range(1, 1);

print(results[0]);
```

8.21.99 test-ListRange2.out

```
1
```

8.21.100 test-ListRange1.pc

```
results = List.range(4, 7);

print(results[0]);
print(results[1]);
print(results[2]);
print(results[3]);
```

8.21.101 test-ListRange1.out

```
4
5
6
7
```

8.21.102 test-ListMap2.pc

```
list1 = ["hey","i","talk"];

results1 = List.map(list1, (x) -> {
        return x[0];
});

print(results1);
```

8.21.103 test-ListMap2.out

```
hit
```

8.21.104 test-ListMap1.pc

```
list1 = [1, 2, 3];

fun = (x) -> {
  return x+1;
};

results1 = List.map(list1, fun);

print(results1[0]);
print(results1[1]);
print(results1[2]);
```

8.21.105 test-ListMap1.out

```
2
3
4
```

8.21.106 test-ListLength2.pc

```
list1 = [1, 2, 3, 4, 5, 6];

print(List.length(list1));

list1 = List.remove(list1 , 1);

print(List.length(list1));
```

8.21.107 test-ListLength2.out

```
6
5
```

8.21.108 test-ListLength1.pc

```
list1 = [1, 2, 3, 4, 5, 6];

print(List.length(list1));
```

8.21.109 test-ListLength1.out

```
6
```

8.21.110 test-ListFind2.pc

```
myList = [ 1, 2, 3, 4 ];

print(List.find(myList, [1, 2]));
print(List.find(myList, [3, 4]));
```

8.21.111 test-ListFind2.out

```
0
2
```

8.21.112 test-ListFind1.pc

```
myString = "hello";

print(List.find(myString, "dog"));
print(List.find(myString, "el"));
```

8.21.113 test-ListFind1.out

```
-1
1
```

8.21.114 test-ListAdd2.pc

```
list1 = [];

list1 = List.add(list1, 1);
list1 = List.add(list1, 2);
list1 = List.add(list1, 3);
list1 = List.add(list1, 4);
list1 = List.add(list1, 5);

print(list1[4]);
```

8.21.115 test-ListAdd2.out

```
5
```

8.21.116 test-ListAdd1.pc

```
list1 = [0, 1, 2, 3, 4, 5, 6];

list2 = List.add(list1, 7);

print(list2[7]);
```

8.21.117 test-ListAdd1.out

```
7
```

8.22 Demos

8.22.1 chat.pc

```
print("How many people are you chatting with?");
num = numFromString(read());
print("Go ahead and start chatting!");

chatFn = msg -> {
    print(msg);
    return read();
};

while(true) {
    msg = read();
    resp = distribute(List.populate(msg, num), chatFn);
    List.map(resp, print);
}
```

8.22.2 ddos.pc

```
print("What server would you like to DDOS?");
url = read();
print("How many machines do you want to use?");
machines = numFromString(read());

distribute(List.populate(url, machines), url -> {
        for(i = 0; i < 500; i = i+1) {
                download(url);
        }
        return true;
});
```

8.22.3 fib.pc

```
fib = (x) -> {
  if (x == 0) { return 0; }
  if (x == 1) { return 1; }
  return rec(x-1) + rec(x-2);
};

print(fib(7));
```

8.22.4 inference1.internal

```
TChar             = 'AN
'AO               = TObj(id:TNum, name:TList('AN))

TChar             = 'AP
TChar             = 'AQ
'AR               = TObj(id:TList('AP), email:TList('AQ))

'Ah               = 'AS
'Ag               = TObjAccess(id:'Ah)
'Ag               = TObjAccess(foo:'Ax)
'Ai               = TFunc('Ag -> 'AS)

TFunc('AO -> 'Aj) = 'Ai
'Ak               = 'Aj
TFunc('AR -> 'Al) = 'Ai
'Am               = 'Al

TFunc('Ak -> 'An) = TFunc('AA -> 'AA)
TFunc('Am -> 'Ao) = TFunc('AA -> 'AA)
```

```
SUBSTITUTIONS

Ao   => TList(TChar)
An   => TNum
Am   => TList(TChar)
Al   => TList(TChar)
Ak   => TNum
Aj   => TNum
Ai   => TFunc(TObjAccess(id:'AS) -> 'AS)
Ag   => TObjAccess(id:'AS)
Ah   => 'AS
AR   => TObj(id:TList(TChar), email:TList(TChar))
AQ   => TChar
AP   => TChar
AO   => TObj(id:TNum, name:TList(TChar))
AN   => TChar
```

8.22.5 inference1.pc

```
o1 = {
  id: 7,
  name: "dude"
};

o2 = {
  id: "my_unique_key",
  email: "alden@quimby.com"
};

unwrap = (o) -> { return o.id; };

numId = unwrap(o1);
strId = unwrap(o2);

print(numId);
print(strId);
```

8.22.6 inference1.types

```
o1     ==> TObj(id:TNum, name:TList(TChar))



o2     ==> TObj(id:TList(TChar), email:TList(TChar))



unwrap ==> TFunc(TObjAccess(id:'AS) -> 'AS)

numId  ==> TNum
```

```
strId  ==> TList(TChar)
```

8.22.7 inference2.pc

```
myFunc = (f, a) -> { return f(a); };

aNumber = myFunc(a -> { return a+1; }, 5);

aString = myFunc(a -> { return a^"snd"; }, "fst ");

print(aNumber);
print(aString);
```

8.22.8 inference2.types

```
myFunc  ==> TFunc(TFunc('Ac -> 'AN),'Ac -> 'AN)

aNumber ==> TNum

aString ==> TList(TChar)
```

8.22.9 inference3.pc

```
iGen = () -> {
        return () -> {
                return x -> { return x; };
        };
};

myNums = List.map([1,2,3], iGen()());

obj = {
    i: iGen,
    wrap: (f) -> {
            return { a: { b: [f()] } };
        }
};

unwrap = (o) -> {
        return o.a.b[0]();
};

mapper = unwrap(obj.wrap(obj.i));

wow = List.map("wow", mapper);

print(wow);
```

8.22.10 inference3.types

```
iGen ==> TFunc( -> TFunc( -> TFunc('Ap -> 'Ap)))




myNums ==> TList(TNum)

obj ==> TObj(
        i: TFunc( -> TFunc( -> TFunc('Ap -> 'Ap))),
        wrap: TFunc(
                TFunc( -> 'Bb) -> TObj(a:TObj(b:TList('Bb)))
        )
)

unwrap ==> TFunc(
        TObjAccess(a:TObjAccess(b:TList(TFunc( -> 'Bd)))) -> 'Bd
)

mapper ==> TFunc('CP -> 'CP)

wow ==> TList(TChar)
```

8.22.11 inferencefixedafterpresentation.pc

```
f = (o) -> {
       x = o.id.a();
       print(x ^ " awesome!");
       return numToString(o.id.b) ^ o.name;
};

//should work!
myObj = {
       id: {
               a: () -> { return "this is "; },
               b: 10
       },
       name: "heyo"
};



print(f(myObj));
```

8.22.12 inferencefixedafterpresentation.types

```
f ==> TFunc(TObjAccess(name:TList(TChar), id:TObjAccess(a:TFunc( -> TList(TChar)), b:TNum)) -> TList(TChar))
x ==> TList(TChar)
myObj ==> TObj(id:TObj(a:TFunc( -> TList(TChar)), b:TNum), name:TList(TChar))
```

8.22.13 reddit.pc

```
getRedditTitles = (redditUrl) -> {
        result = download(redditUrl);
        links = List.where(List.split(result, '<'), el -> {
                return List.find(el, "a class=") >= 0
                        && List.find(el, "title") >= 0;
        });
        titles = List.map(links, x -> {
                parts = List.split(x, '>');
                return parts[1];
        });
        return titles;
};

counts = List.map(List.range(0, 9), x -> { return x*25; });

urls = List.map(counts, x -> {
        return "http://www.reddit.com/?count=" ^ numToString(x);
});

allTitles = distribute(urls, getRedditTitles);

List.map(allTitles, titles -> {
        return List.map(titles, print);
});
```