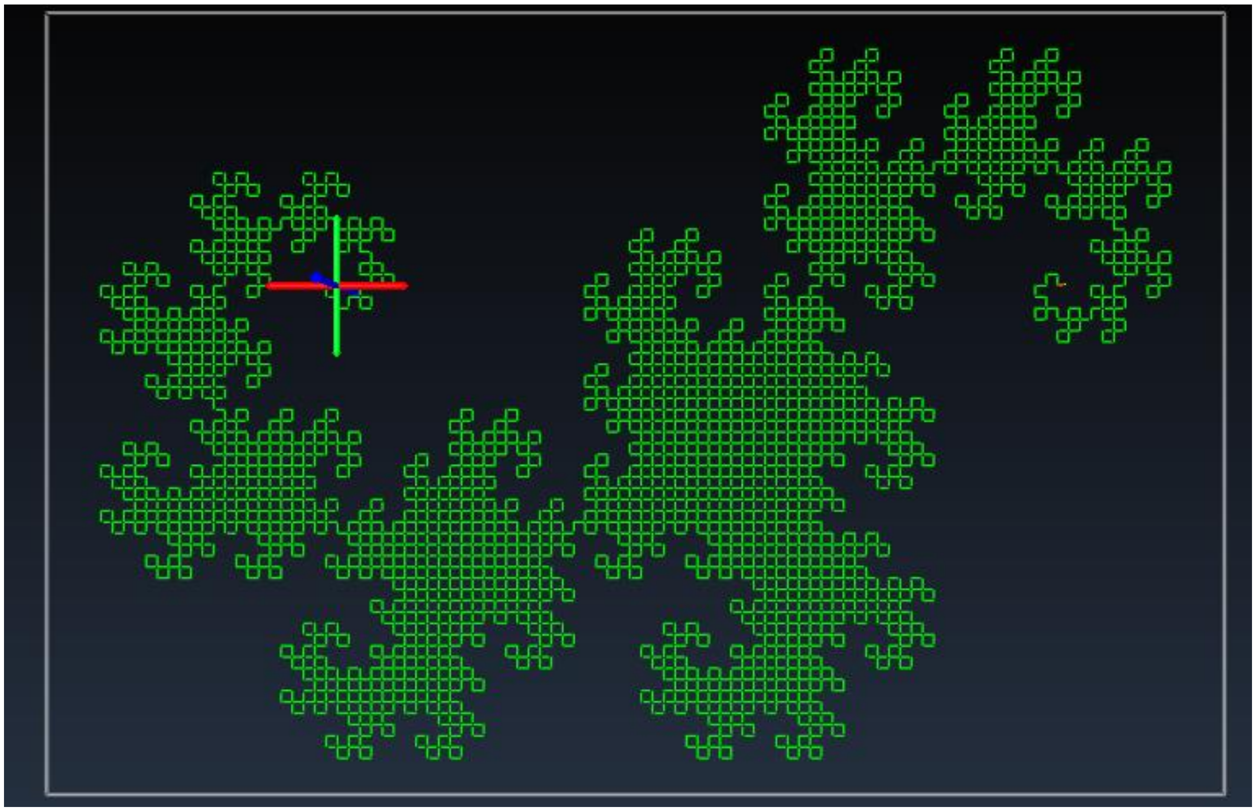
GeoCode Final Report



NC Code produced by Geocode
NC Code rendering compliments of Camotics: http://camotics.org/

Contents

1 Introduction

1.1 Motivation

With the advent of 3d machining and the availability of affordable routers, CNC machines have become quite common. The driving force behind a CNC machine is G-Code, an old language that is very cumbersome and verbose. The G-Code language focuses on the mechanics of the machine and this verbosity obscures the meaning of the final program.

Geocode is a language that focuses on allowing the user to describe the program through the definition of the geometrical entities that should be machined. By allowing the user to focus on higher logic, the program will be less obscured by the details of the machining process. The user will create the geometric entities that compose the target motions and Geocode will produce the output in a G-Code file that can be run on a CNC machine.

1.2 Key Features

1.2.1 Classes

Geocode allows for the creation of user defined classes. Classes contain a collection of user defined functions and variables.

1.2.2 Standard Library

The standard library defines basic functions and classes that are needed to created NC programs.

1.2.2.1 Local Coordinate System

All geometric entities contain a reference system of the type LCS. The LCS contains the following members:

- Origin: 2-d point to describe the center point of the reference system

- Axes: 2-d U and V orientation vectors for the reference system

The LCS contains the following transformation functions:

- Rotate(radians): Rotate the reference system by radians
- Translate(u,v): Translate the coordinate system by the vector <u,v>

1.2.2.2 Configuration

A configuration class contains all the information needed to properly perform a motion for a geometric entity. A configuration contains the following members:

- LCS: the coordinate system
- Retract Height: the height to retract between entities
- Cutting Height: the height that cutting takes place
- Feed Rate: The speed at which the cutting takes place

1.2.2.3 Geometric Entities

Geometric entities are included as part of the standard library. All entities contain a configuration and a description (for NC code commenting).

- Line(point): Start point at origin and end point at the point.
- Polyline(list[point]): First point is at the origin and moves to each point in the list.
- Circle(radius): center is the origin with the passed in radius.

Geometric entities define the following functions:

- Translate(u, v)
- Rotate(u, v)
- Print(): Output the NC Code for the geometric entity.

1.2.3 Indentation

Geocode uses tab indentation to identify grouping of logical statements. All lines must begin on the same indentation level as the previous line, unless a new logical block is defined. A new block is defined with the previous line ending in a colon.

An example of this is as follows:

```
Test(int check):
    int x

    x = 42
    if (check > 42):
        x = x + check

    return x
```

2 Language Tutorial

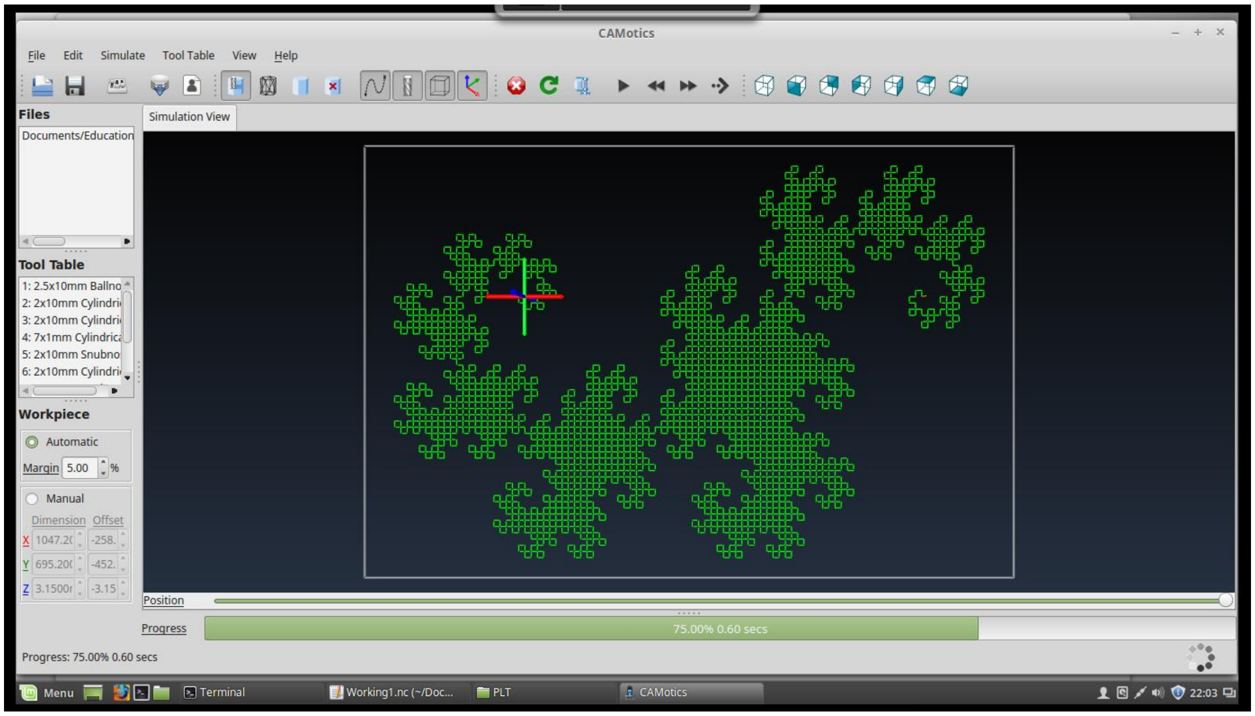
The best way to learn Geocode is to dive straight into the code. So let us begin with a simple Geocode program.

2.1 Sample Program: Dragon Curve

```
/* Dragon Curve */
include "../Standard/standard.geo"

main():
        class Math math
        class Config config
        class Polyline a, b
        int x

        /* Initialize origin and cutting/retract planes */
        config = Config([retract_height:5.0, cutting_height:-1.0, feed_rate:100.0])
        config.lcs.Identity()

        /* Create the first line for the dragon curve */
        a = Polyline([config:config, desc:"(Dragon Curve)"])
        a.AddPoint(Point([x:-10.0, y:0.0]), a.config.lcs)

        /* dragon curve iterations */
        for (x = 0; x < 12; x = x + 1):
                /* copy the previous geometries */
                b = a

                /* Create a new copy rotated 90 degrees around the end point */
                b.Reverse()
                b.Rotate(math.ToRad(-90.0))

                /* Append new copy to the original polyline */
                a.Append(b)

        /* Output initilization NC code */
        GCodeInit(config)

        /* Output NC code for dragon curve */
        a.Print()
```

2.1.1 Dragon Curve Rendering

NC Code rendering compliments of Camotics: http://camotics.org/



Documents/Education

1: 2.5x10mm Ballno
2: 2x10mm Cylindri
3: 2x10mm Cylindri
4: 7x1mm Cylindrica
5: 2x10mm Snubno
6: 2x10mm Cylindri

Automatic
Margin 5.00 %

Manual
Dimension Offset
X 1047.2(-258.
Y 695.20(-452.
Z 3.1500(-3.15

75.00% 0.60 secs

Progress: 75.00% 0.60 secs

2.1.2 Sample Program Explanation

Let us go through the sample program line by line and discuss the properties of the Geocode language.

```
/* Dragon Curve */
```

The first line is a comment, which will be ignored by the compiler. Comments are surrounded with the /* and */ symbols.

```
include "../Standard/standard.geo"
```

The include statement takes in a relative path and includes the source code of that file in this file. Here we are including the standard libraries in our program.

```
main():
```

This is a function declaration for the main function. All function execution begins in the main function, which must defined.

```
	class Math math
```

This line is declaring a class instance of the Math class, which is found in the standard library. Declaration of class variables must begin with the keyword class.

Indentation is important in Geocode as it defines which statements are logically grouped together. Here we are setting the indentation level of the main function to 1 tab.

```
	class Config config
	class Polyline a, b
	int x
```

More variables are being declared. All variables must be declared at the beginning of a function. These variables will be local to the function, which means that all other functions will not be able to access them unless they are explicitly passed to that function.

```
	/* Initialize origin and cutting/retract planes */
	config = Config([retract_height:5.0, cutting_height:-1.0, feed_rate:100.0])
```

Here we are creating an instance of the configuration class, which is found in the standard library. The class creation takes a list as input. A list is always surrounded by square brackets and each item is separated with a comma. Class creation takes in a special list that has key:value pairs as items. This line is creating a configuration instance with a retract height of 5.0, a cutting height of -1.0, and a feed rate of 100.0.

```
	config.lcs.Identity()
```

Here we are calling a function on the lcs member (which is also a class) of the configuration class. This is setting the local coordinate system to the identity matrix.

```
	/* Create the first line for the dragon curve */
	a = Polyline([config:config, desc:"(Dragon Curve)"])
	a.AddPoint(Point([x:-10.0, y:0.0]), a.config.lcs)
```

Once again, we are creating a class instance, this time of the type Polyline. After creating the class, we add the first point to the Polyline.

```
/* dragon curve iterations */
for (x = 0; x < 12; x = x + 1):
```

Here we are starting a loop that will run 12 times. The code that follows will be executed in each iteration.

```
        /* copy the previous geometries */
        b = a

        /* Create a new copy rotated 90 degrees around the end point */
        b.Reverse()
        b.Rotate(math.ToRad(-90.0))

        /* Append new copy to the original polyline */
        a.Append(b)
```

This code is at a new indentation level, which signifies that it is a block of code that should be run for each iteration of the for loop.

The code copies the original polyline in a to the variable b. We are reversing the Polyline found in b, rotating it, and then appending it to the end of the original Polyline a.

```
/* Output initilization NC code */
GCodeInit(config)
```

After the for loop finishes, we are calling a function in the standard library to output the initialization NC Code.

```
/* Output NC code for dragon curve */
a.Print()
```

Finally, we output the NC code for the Polyline a.

2.1.3 Sample Program Compilation

The toolset needed is:

- Download Geocode source file.
- Install OCaml version 4.01.0.
- Install Python 2.7.6.

With the toolset in place, use ocaml to compile the Geocode executable:

```
~/Documents/Education/Geocode $ make
ocamlc -c ast.ml
ocamlc -c utility.ml
ocamlc -c class.ml
ocamlc -c environment.ml
ocamlc -c lists.ml
ocamlyacc parser.mly
ocamlc -c parser.mli
ocamlc -c parser.ml
ocamllex scanner.mll
167 states, 4781 transitions, table size 20126 bytes
ocamlc -c scanner.ml
ocamlc -c ncgenerator.ml
ocamlc -c sast.ml
ocamlc -c geocode.ml
ocamlc -o geocode ast.cmo utility.cmo class.cmo environment.cmo lists.cmo parser.cmo
scanner.cmo ncgenerator.cmo sast.cmo geocode.cmo
```

Finally, run the Geocode compiler with the –nc option

```
~/Documents/Education/Geocode $ ./geocode -nc “Working/Curve.geo” > Curve.nc
```

The output of the compiler will be a G-code file which can be analyzed by any CAM product. I have used Camotics to render and analyze the output.

3 Language Manual

3.1 Introduction

With the advent of 3d machining and the availability of affordable routers, CNC machines have become quite common. The driving force behind a CNC machine is G-Code, an old language that is very cumbersome and verbose. The G-Code language focuses on the mechanics of the machine and this verbosity obscures the meaning of the final program.

GeoCode is a language that focuses on allowing the user to describe the program through the definition of the geometrical entities that should be machined. By allowing the user to focus on higher logic, the program will be less obscured by the details of the machining process. The user will create the geometric entities that compose the target motions and Geocode will produce the output in a G-Code file that can be run on a CNC machine.

3.2 Lexical Elements

3.2.1 Identifiers

Identifiers are case sensitive strings used for naming variables, functions, and objects. All identifiers must start with an upper or lower case letter. Valid characters for and identifier are:

- Letters (upper and lower case)
- Digits (0-9)
- Underscore

3.2.2 Keywords

Special identifiers are reserved by the Geocode language. These identifiers are:

- `bool`
- `class`

- `double`
- `else`
- `false`
- `for`
- `if`
- `include`
- `int`
- `list`
- `return`
- `string`
- `true`
- `while`

3.2.3 Literals

Literals are notations for constant values of primitive types.

1) Integer Constant: A sequence of digits (0-9). Example: 42
2) Double Constant: A sequence of digits (0-9) that contain the decimal(.) character. The decimal may be the first character in the sequence as long as digits follow the decimal. Example: 13.2 or .17
3) String Constant: A sequence of zero or more characters, digits, and escape sequences surrounded by quotation marks. Invalid characters for a string are the newline and double quotation mark. Example: "test@"
4) List Constant: A sequence of the same data type surrounded in square brackets and separated with a comma. Any of the above 4 data types can be used in a list. Example: [1, 2, 3, 4, 5]
5) Key Value Pair List: A special type of list that contains key value pairs. A key value pair is signified with an `identifier:expression`. All pairs are separated by a colon. Example: [answer:12, name:"Todd"]

3.2.4 Line Structure

A Geocode program is divided into a group of lines called statements that are terminated by the Unix form using ASCII LF (linefeed). A statement must be on a single line (cannot cross multiple lines).

3.2.5 Indentation

Leading tabs at the beginning of a statement are used to signify the indentation level. The indentation level for a group of statements must be the same, which forms a block of execution.

3.2.6 Blocks

A block is a logical grouping of statements that are all on the same indentation level. A block must have an indentation level that is greater than the identifier that started the block. A new block must be started in the following cases:

- Function definition
- Class definition
- If
- Else
- While
- For

A block is evaluated from the beginning to end. The first statement to follow one of the above identifiers defines the indentation level for the block. The following is an example of a function definition. The block following the function definition will be executed upon calling the function.

```
PrintAnswer():
	int x
	x = 42
	print(x)
```

3.2.7 Whitespace

Spaces and tabs (except for leading tabs) are ignored.

3.2.8 Comments

Comments begin with the character sequence /* and end with character sequence */. Everything in a comment is ignored by the compiler.

3.3 Data Types

3.3.1 Primitive Data Types

There are 4 primitive types in the Geocode language: integer, double, bool, and string.

3.3.1.1 Integer Types

Integer values are numbers from -2^{30} to $2^{30}-1$. Uninitialized integers are set to the value 0.

3.3.1.2 Double Types

Double values are numbers in floating-point representation. The current implementation uses double-precision floating-point numbers conforming to the IEEE 754 standard, with 53 bits of mantissa and an exponent ranging from −1022 to 1023. Uninitialized doubles are set to the value 0.0.

3.3.1.3 Bool Types

Boolean values are the truth values true or false. Uninitialized bools are set to the value false.

3.3.1.4 String Types

String values are a sequences of characters. All characters are valid except for the character ” and a newline. Uninitialized strings are set to the value “”.

3.3.2 Class

A class is a user-defined data type made up of variables of primitive types and other classes. The inclusion of class types are allowed with two exceptions:

- A class may not include itself (recursion).
- Two classes may not mutually include each other (mutual recursion).

3.3.2.1 Defining Classes

A class is defined with the keyword class followed by the name and then a colon. The next non-empty line defines the indentation level for the class member declarations. This indentation level must be greater than the previous line that defined the class.

The member variables (if any) must be defined first. A member variable is declared the same as a regular variable. Following the member variable declaration is the member function declaration. A member function declaration is the same as a regular function declaration. A class must have either at least one member variable or one member function.

The end of the class definition is signified by the end of the indentation level for the class member declarations. Here is an example of defining a simple object for a person:

```
class Person:
   string name
   int age
   Print():
      print(name)
```

This defines a class named Person which has two member variables (name and age) and one member function (Print).

3.3.2.1 Declaring Class Variables

Variables of a class are declared the same as primitive variables, however they must be preceded by the keyword class. The line must include the class keyword, the name of the class, and the instance name.

```
class Person bill, ted
```

This example declares two variables of type Person named bill and ted.

3.3.2.2 Initializing Class Instances

A class instance is created with all the primitives set to the default values. To construct a class that contains non-default values, use the class constructor syntax:

```
classname()
classname([identifier:expression, identifier:expression])
```

The class constructor with no parameters returns a class instance with all default values. A class constructor that is passed a list of key value pairs will initialize the class with the following constraints:

- The key must be a name of a member variable.
- The key is an identifier, which cannot contain the access operator.

- The value must be the same type as the member variable.
- The value is an expression which will be evaluated.
- Multiple key value pairs must be separated with a comma.

The class constructor will set the members matching the key to the value returned by the expression.

```
Person bill
bill = Person([name:”Bill”, age:Answer()])
```

With this syntax it is possible to initialize a member that is a class, however, it is not possible to initialize a member of a member of a class. The following line is NOT valid:

```
a = Circle([lcs.u.x:1.0, lcs.u.y:0.0])
```

3.3.2.3 Accessing Class Members

Class members are accessed using the member access operator, which is the period (.) symbol. To access a member, type the name of the class instance followed by the access operator followed by the member name. The access operator is recursive, therefore it is possible to access the members of a class within a class.

```
bill.name = “Bill”
bill.Print()
```

The access operator is only defined for classes and lists, all other uses of the access operator will result in a syntax failure.

3.3.3 List

A list is a container type that holds a collections of primitive types or classes. All elements of a list must be the same data type. A list of list is not supported at this time.

3.3.3.1 Declaring Lists

List variables are declared similar to primitive variables, except the type that the list contains must be enclosed in square brackets.

```
list[int] numbers
```

3.3.3.2 Initializing Arrays

List variables can be initialized by enclosing the values in square brackets. All values in the initialization list must be of the same type as the declared list type. Any mixing of types in the initialization list or in the assignment will result in an error.

```
list[int] numbers = [ 0, 1, 2, 3, 4 ]
```

3.3.3.3 List Functions

List functions are called just like class functions, using the access operator. Indices in list are zero-based. The list type only defines the following functions:

- at(): Return the value at the zero-based index provided.
- count(): Return the number of elements in the list.
- insert(index, element): Insert element into the list at zero-based index and return the inserted item.
- pop(): Remove and return the last element on the list.
- push(element): Add element to the end of the list.
- remove(index): Remove and return the element at zero-based index.
- reverse(): Reverse the order of the list.

List functions are called the same as class functions:

```
print(numbers.count())
```

3.3.3.5 Concatenating Lists

Lists of the same type can be concatenated with the + operator. The order of newly created list1 + list2 will be list1 followed by list2. Any concatenation of lists of different types will result in an error.

```
all_numbers = [ 0, 1, 2, 3, 4 ] + [ 1000, 1001, 1002, 1003, 1004 ]
```

3.4 Expressions and Operators

3.4.1 Expressions

An expression is a literal, an identifier, a binary operator, an assignment, a function call, of a grouping of expressions surrounded by parentheses. Here are some examples:

```
     42
     21 * 2
     print(42)
```

3.4.2 Assignment Operators

Assignment operators store values in variables. The assignment operator = stores the value of its right operand in the variable specified by its left operand.

```
left-operand = right-operand
```

The left-operand must be a variable. A class member can be set by using the access operator. The right-operand can be an expression that returns a value. The type of the left-operand must be the same as the right-operand, or this will result in a semantic error.

The result of the assignment operator is the value right-operand.

```
     String test = “Hello World”
```

3.4.3 Arithmetic Operators

Addition +, subtraction -, multiplication *, and division / are called with the use of the operators identified with infix notation. Arithmetic operators only operate on variables of the same type.

Arithmetic operators are fully supported for integers and doubles:

```
     x = 5 + 3
     x = 101.0 / 2.0
```

The addition operator is defined for the string and list type.

```
    x = "Hello " + "World"
```

3.4.4 Comparison Operators

Two variables of the same type can be compared through comparison operators. The comparison operators are equality ==, not equal !=, less than <, less than or equal <=, greater than >, and greater than or equal >=.

All primitive data types can use the equality and not equal operators.

```
    if ("Hello " != "World")
```

The rest of the comparison operators are only supported for double and integers.

```
    if (2 < 3)
```

3.4.5 Type Casts

The syntax for a type cast is as follows:

```
<type>(expression)
```

The expression's return value will be cast into the type specified. Currently, the only supported type cast are the following conversions:

- Integer to String
- Double To String

All non-supported casts will result in a run-time error.

3.4.6 Operator Precedence

The operators are listed in the order of precedence:

None	Negation	-
Left Associative	Access Operator	* \

Left Associative	multiplication and division	* \
Left Associative	addition and subtraction	+ -
Left Associative	less than and greater than	< <= > >=
Left Associative	equality and not equal	== !=
Right Associative	assignment	=

3.5 Statements

Statements are expressions or special actions that change the control flow within the program.

3.5.1 Block

A block is a group of statements that are treated by the compiler as a single statement. All statements in a block must have the same tab indentation level.

Blocks can only be nested inside other blocks when starting a new statement type that can contain a block. These statements are the following:

- if
- else
- while
- for

Any other change in indentation level will result in a compilation error.

3.5.2 if Statement

The if statement executes a group of code when the predicate is true. The predicate of the if statement must be an expression that can be evaluated as true or false.

The form of the if statement is the keyword if followed by a predicate surrounded by parentheses, ending with a colon. There may be an else associated with the if statement, which will be executed if the predicate is not true.

```
if (expression-predicate):
  block
else:
  block
```

A new block must be started after the if statement with an indentation level greater than the if statement. All statements in the block will be execute if the predicate is true.

One or no else statements can included after an if statement. The indentation level is used to find the corresponding else. The block defined inside the else statement will be executed if the predicate is false.

The if and else statements must have at least one statement defined in their block.

```
x = 5 + 3
if (x == 8):
   x = x + 5
   print(“Thirteen is great!”)
else:
   x = 8
   print(“Now x is great!”)
```

3.5.3 while Statement

The while statement evaluates the predicate expression before beginning execution. If the predicate is true, then the block will execute once. After this execution, the predicate will be evaluated again, and if true, the block will execute.

```
while (expression-predicate):
  block
```

3.5.4 for Statement

The for statement is similar to the while statement in that it will continually loop as long as the predicate is true. The for statement has 3 parameters:

1. Initialization expression: The expression will be executed once before entering the for loop body.
2. Predicate expression: The for loop will continue as long as the predicate is true.

3. Step expression: At the end of each iteration of the loop, this expression will be called.

```
for (initialize; test; step)
  statement
```

3.5.5 return Statement

The return statement is used inside a function to stop the execution of that function and return the value to the caller.

```
WhatTheCatSay():
  return “Meow”

main():
  string answer
  answer = WhatTheCatSay()
```

If a function returns a value, then all return statements in that function must return the same value. That is, the same function cannot return values with different types.

3.6 Functions

Functions are used to group logical code into a unit that can be called later in the program. A function must be declared and defined before it is used in the program.

Every program requires at least one function, called main. That is where the program's execution begins.

3.6.1 Function Declarations

A function is declared with a function name, and a parameter list surrounded by parentheses. Here is the general form:

```
function-name (parameter-list):
```

A parameter is a variable type followed by a variable name. A parameter list is a sequence of parameters separated with commas. The parameter list can be empty. Here is an example of a function declaration with two parameters:

```
int add(int x, int y):
```

3.6.2 Function Definitions

A function must be defined right after the declaration. The function definition is a block following the function declaration that has a greater indentation than the function declaration.

```
int add(int x, int y):
   return x + y
```

3.6.3 Function Return Types

A function does not to specify the return type, it will be evaluated by the compiler. A function's return type must be consistent, which is evaluated with the following rules:

- A function with no return statements is consistent.
- A function with a return statement must also return at the end of the function block.
- All function return statements must return the same type.

3.6.3 Calling Functions

A function is called by its name along with any parameters that are required. The same number of parameters that were defined in the function definition must be given, otherwise this will result in an error. The result of a function is based upon the return type or if no return type is signified, then it is the integer value 0.

3.6.4 The main Function

Every program must have one function, called 'main'. This is where the program begins executing. The main function has no return type and no parameter list.

```
main ():
   print("Hello World!")
```

3.6.5 print Function

The print function directs the parameters to the final output of the program. The following program would print the text "Hello World" upon completion of the program:

```
main ():
   print(“Hello World!”)
```

The print function takes only one parameter and will output a string representation of this data type.

3.7 Program Structure and Scope

A program is a file that contains the main function. This program may contain any number of other functions that assist in the program's execution.

3.7.1 Include

A program may include another file by using the include command. The include command inserts the specified file in the current file as if the file was embedded. The path to the include file must be a relative path.

Geocode is able to include files that have includes in them. The include path is relative to the location of the current source file. If there are source files that recursively include each other, then the compiler will terminate and return an include error.

```
include "standard.geo"

main ():
  Circle circle(radius:5)

  print(circle)
```

3.7.2 Scope

Scope refers to what parts of the program can "see" a declared object. Any line in the program has access to:

1. All variables in the current statement block.
2. All variables declared in the previous lines of the same function that are of a lesser indentation level.
3. All function parameters of the enclosing function.
4. All global variables.

The current statement block is defined as all lines that are of the same indentation level.

3.8 Standard Library

A standard library standard.geo is included with the compiler. This file defines the basic geometry classes and some utility classes.

3.8.1 Point

The Point class represents a 2-dimensional point. The class contains 2 double members: x, y. The data members can be accessed with the class access operator:

```
Point start

start.x = 4.2
```

3.8.2 Local Coordinate System

All geometric entities contain a reference system of the type LCS. The LCS contains the following members:

- Origin: 2-d point to describe the center point of the reference system
- Axes: 2-d U and V orientation vectors for the reference system

The LCS contains the following transformation functions:

- Rotate(radians): Rotate the reference system by radians
- Translate(u,v): Translate the coordinate system by the vector <u,v>

3.8.3 Configuration

A configuration class contains all the information needed to properly perform a motion for a geometric entity. A configuration contains the following members:

- LCS: the coordinate system
- Retract Height: the height to retract between entities
- Cutting Height: the height that cutting takes place
- Feed Rate: The speed at which the cutting takes place

3.8.4 Geometric Entities

Geometric entities are included as part of the standard library. All entities contain a configuration and a description (for NC code commenting).

- Line(point): Start point at origin and end point at the point.
- Polyline(list[point]): First point is at the origin and moves to each point in the list.
- Circle(radius): center is the origin with the passed in radius.

Geometric entities define the following functions:

- Translate(u, v)
- Rotate(u, v)
- Print(): Output the NC Code for the geometric entity.

4 Project Plan

4.1 Overall Plan

The overall plan was broken up into 7 phases. The main objective was to get to the golden rule: “Only commit code when all tests pass.” I was not able to accomplish this goal until phase 4 (most previous stages were planning/preparation).

Once passed phase 4, there was a strong push to guarantee that I added more tests with each new feature added. I ended up with 112 unique tests.

4.1.1 Phase 1: Specification

The first objective was to define what compiler I would like to build. This involved combining the things I like to work on (NC Code) with the languages I enjoy most (c++ and python).

The goal of this phase was a Geocode whitepaper describing the compiler that would produce G-code.

4.1.2 Phase 2: Language Grammar

The next objective was to verify the Geocode syntax. This involved working extensionally with ocamlyacc to verify that my grammar would not be ambiguous. In many cases, I was forced to make syntactic compromises from the original vision listed in the whitepaper.

The goal of this phase was to produce a grammar parser that contained no ambiguities. From this, I was able to write the Geocode Language Reference Manual.

4.1.3 Phase 3: Formatting

The next objective was to just get something simple working. I tackled the pre parser which would define the python like indentation style. I did not add any types at this time, just kept working with what was included in microc.

The goal of this phase was compiling a correctly formatted program that would output hello world.

4.1.4 Phase 4: Basic Tests

With a simple working file, now came the time to ensure the tests pass. I reformatted all the test cases that were include with microc. At this point I made sure all existing cases pass. I did some minor tweaks on the regression test script.

The goal of this phase was to get the tests pass. Now I will implement the rule, no commits unless all tests pass.

4.1.5 Phase 5: Types

Once the tests were in order, I added all the types that were required for the Geocode project. The goal of this phase was to have all types implemented that were reference in the LRM.

4.1.6 Phase 6: Semantic Checks

Finally, with everything in place, it was time to get the semantic error checking working.

4.1.7 Phase 7: Fun

Now that all the pieces were in place, I spent some time producing some cool results in my language. The result of this was the dragon curve sample program that you see at the beginning of this report.

4.2 Style Guideline

- Use tabs for indentation
- Group similar logical statements with the same indentation level.
- Variable names should be clear.
- Source files over 300 lines should be broken up into multiple source files.
- Break big blocks (10+ lines) into functions as much as possible.

4.3 Software Tools

- OCaml version 4.01.0.
- Python 2.7.6.

4.4 Commit Log

```
commit 2c83667450df93872205be495743315a6971775f
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 21 18:46:05 2015 -0500

    Removed unused files.
    Updated tests to be sematically correct.
    Added more semantic tests.

commit 8a31410ee000e9dc737b34263bac66132164f6f4
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 21 16:47:27 2015 -0500

    Renamed folder with correctly spelled Semantics.

commit 30973eaf2a0159e8b94befa7faf96fc49000cbac
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 21 15:14:52 2015 -0500

    Semantics check passed the first test!
    Added first semantics test.

commit 06d80ac5ed24d70e6ebb715f2c022f0fe1408451
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 21 11:51:57 2015 -0500

    Few changes with control loops for semantics.
    Pre check-in before updates to the return syntax in semantics check.

commit 81ae95181bfa00af1b424e9cfac4355445fe7dbc
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 21 11:05:58 2015 -0500

    First working copy of sematic analyzer.

commit 5c28ec2095a0d4e828c1d14455f9524ddff890f8
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 19:56:50 2015 -0500

    Renamed interpret file to reflect nc code generation ability.
```

```
commit 8fffe2e4acb8b58b28fd3159a7a9e2a1ac83d369
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 19:27:21 2015 -0500

    Dragon Curve example created.

commit d9d31d558dfe319cd181f76c6720ee95883ac8f3
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 16:47:29 2015 -0500

    Polyline first implementation finished.
    Added tests for polyline.

commit abeb93e5c5c5d78d4419c6d445a997451f7c00d5
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 15:53:35 2015 -0500

    Moved list code into separate file.

commit a8ca123f794384112d85fdccdf628b7d48e9a93a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 15:48:52 2015 -0500

    List functionality completed.
    More list tests added.

commit 17f1c1ed5e86eb51a86c8ff253dc1965d0384200
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 11:55:25 2015 -0500

    Lists are now types.
    Added functionality for List At.
    Tests added for lists.

commit 2590263196248e195dfc6f200f33a8260ff81806
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 10:02:11 2015 -0500

    Move shared function to Utility.
    Fixed bug with embedded class function calls.

commit 4b785d36cb2b12ca6d8eaf950b8b986a384f032c
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 20 09:23:09 2015 -0500

    First implementation of Line rotation.
    Tests added for line creation and rotation.

commit e408829365adcc9280f838b413e79236c9755aa8
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Dec 19 13:36:55 2015 -0500

    Updated preparser to handle relative include directories.
    Reorganized standard library.
```

```
commit 17b33ec3b172c90eb9257387df4247c95e6025bc
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Dec 19 12:56:16 2015 -0500

    Fixed minor bugs with standard library.
    Added more cases for circles.
    Updated LCS to be 2-dimensional.
    Created first working test of olympic rings.

commit b81bc96aca9861ad2d297cf61623ed1a7df4a774
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Fri Dec 18 19:55:33 2015 -0500

    Bool type added. New tests added for boolean type.

commit 8df2eaa8ea5e1a1b692c48f81529b3f50b95791b
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Fri Dec 18 16:28:59 2015 -0500

    Fix error message bug with class construction and function calling.
    Added new function calling tests.

commit 3cb6e3583aef965cbd6c7d23422efb62300d5f0c
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Fri Dec 18 15:17:28 2015 -0500

    Renamed test files to have .geo extension.

commit b096bfe9c9536e4e0423924c8c5f3ed598fabfef
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Fri Dec 18 14:43:46 2015 -0500

    Renamed to Geocode.

commit 92d97bcbe7295f6579e9983ac0fed13b098824ab
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Fri Dec 18 14:39:31 2015 -0500

    Formatting updated.

commit 624b0f0cbecbc7699f8745600c77273ce97f8622
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Thu Dec 17 20:53:01 2015 -0500

    Reorganized code, added comments, removed unused code.

commit 92204193eaa26cf57a1e93cef006f03976bf78a7
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Dec 15 20:52:52 2015 -0500

    Fixed bug with string parsing.  Test added to catch failure.

commit b0e4cd53814d15cd127e6a71919fac393351dcd1
Author: Carroll Loy <crl2131@columbia.edu>
```

```
Date:   Tue Dec 15 20:00:14 2015 -0500

    Tests added for syntax errors.

commit 523f0f48b4e09078f8aa1666fe4c70669bd0f61f
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Dec 15 18:45:40 2015 -0500

    More test coverage for class constructors.

commit 83d26784f1ec127dfce0ed75c4aac759544a8cf3
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Dec 8 21:14:34 2015 -0500

    Class constructors now activated.
    Key value pair lists implemented to allow class constructor parameter passing.

commit 4e2ba80c7a6606744f6aafa26f0939fbb261975a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Dec 8 13:04:09 2015 -0500

    Fixed shift/reduce conflicts with casting operator.

commit 79524b2f7da73f5de6036425983a14ff0a4d9611
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Dec 8 09:11:37 2015 -0500

    Updated error messages for easier debugging.

commit 7c87c3735b684e01750c801d70c1456b3772acd4
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 7 21:22:12 2015 -0500

    String casting added.

commit 786d6fbe5074a9ed5868b6f53c8a35575a2f348b
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Mon Dec 7 20:45:27 2015 -0500

    Fixed bug with calling functions of objects within objects.

commit 4b476d62d89763c79e901ae622555a1d8e2af282
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 6 16:35:36 2015 -0500

    Added some compiler syntax error diagnostic messages.

commit 6bc5f9ec7d0c5a3cf0d88d0c4478e9b3b15d568a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Dec 6 08:33:42 2015 -0500

    Tests added for double and string binary operations.

commit 882a2bcdbc763640abd6ba818d1c346ba7e45f1c
Author: Carroll Loy <crl2131@columbia.edu>
```

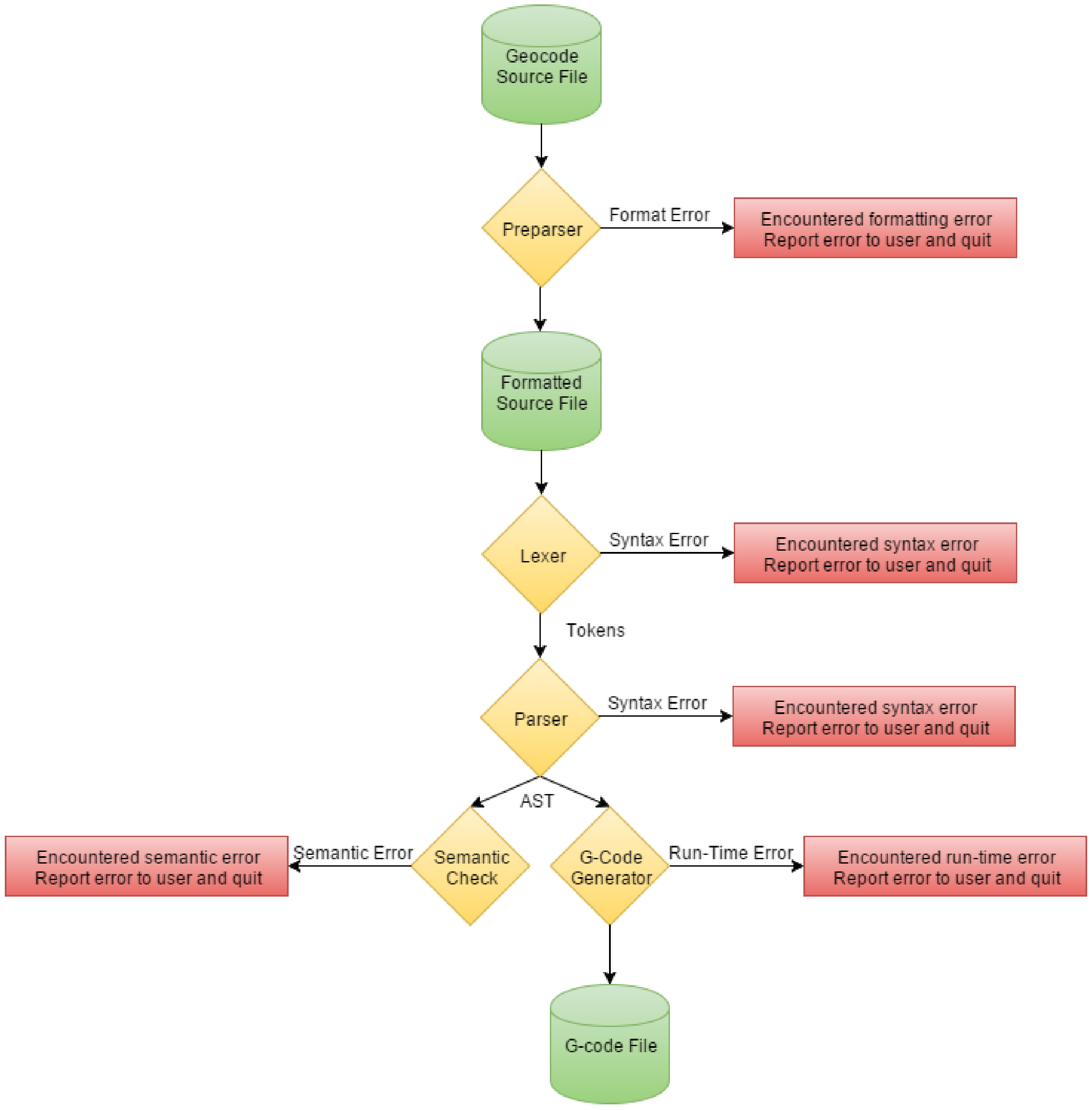
```
Date:   Sat Dec 5 21:57:40 2015 -0500

    String and double binary operations and comparisons added.

commit dd0f11fa812afa974c2bc2b8ac887afa6923f4fd
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Dec 5 19:27:24 2015 -0500

    string and double stand alone boolean check added.

commit 77f97b02d68eaed9c5cab70f90417ce10ce8288a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Dec 5 15:06:26 2015 -0500

    Double type added.

commit 4cff003c85aa7f903c7c3cc6f84cf5795d15441c
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 22 17:28:17 2015 -0500

    Preparser is now handling include command.

commit bb9b59f8227e1a0cd9ca39dad118260db32306ad
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 22 16:05:47 2015 -0500

    Pre_Parser promoted!
    Added lots of tests for pre-parser.

commit b4fb6217672fb746a5485a5100e36fd36cffb879
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 22 12:32:29 2015 -0500

    Updated microc to take in the source file and call python pre_parser script.
    Working on acceptable script pre_parser script.

commit b36d122b43a3b95005666147815cf4e3971965ca
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Nov 21 18:34:24 2015 -0500

    an now call class functions within class functions.

commit 838c5d38e019550dc93035a8cc79cb9d98d85ebd
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Nov 21 16:05:20 2015 -0500

    Fixed bug with updating envirnoment after class function call.

commit adb1380285e27bed0a1036d19e7d724556036290
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Nov 21 11:47:11 2015 -0500

    Rewrote the way the envirnoment is passed to work for class function side
effects.
    Reworked the access to class member variables.
    Preliminary class tests all pass.

commit 18019545f47ddd5c75710d05af266ed9dd8667ab
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Thu Nov 19 18:58:05 2015 -0500

    Can now assign and access class member variables.
    Working on accessing these values within class functions.

commit 1ae712da78d16fb390570ded9a098c41ab8bc3b5
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Nov 17 17:48:47 2015 -0500

    Class access kind of works.
    Disabled compiler, now only supporting interpreter.

commit 0115b9f7e7f06a5c61aaec1c04b2a75436a4e422
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 15 16:32:45 2015 -0500

    Updated test script to handle test folders for organization.
    Made Classes foler for class tests.

commit 6bf52b76593f48a9f271cfffeee14f37339a5a14
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 15 16:13:23 2015 -0500

    Little clean up work.

commit dd7427c463a68da2ae1f286abf7d42a8145a1649
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 15 16:03:41 2015 -0500

    All tests pass!  Class data members are recognized in class function.

commit 8d5c8c7b594cedd700c2e976af9dce475c859f37
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Wed Nov 11 21:06:23 2015 -0500

    Class functions are now in the function maps for compile and interpretted mode.
    Bug exists in compiled mode when referencing class member variables.

commit 75b0e2812b863059ef50b5c0eb21b030b23f85f9
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Nov 10 20:49:02 2015 -0500

    Class function kind of works in compiler mode.

commit 82f2b6e4875e96f8229af71c75397994238c7433
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 8 17:40:01 2015 -0500

    Test added to ensure global variable multiple variables on same line work.

commit 26f9dbe9d630e288c295ec2fc87ed2ca869ec69c
```

```
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 8 17:34:29 2015 -0500

    Cleaned up variables names.

commit 60b79624256d33d0158c1cb4a45c2a55d1db098a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 8 17:28:45 2015 -0500

    Fixed problem with the parsing definition of my variables.

commit bd02fb3a14ae3afebc876dc2f4d4e802a416ac15
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 8 16:59:48 2015 -0500

    Multiple varaiable declaration on same line now possible for classes.

commit e9d694d6a0a73bb650b2a6a72088bef8b8dd3c86
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 8 16:54:24 2015 -0500

    Class declarations added with class keyword.
    Multiple varaiable definition on same line for functions only.

commit d5f98986b202c0daf04c3299af5e8d747b222f14
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Nov 3 19:34:25 2015 -0500

    Added the ability to define multiple variables on the same line

commit 6994ed5f2acabb210b27efb08eb047062f53dc86
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Nov 3 14:51:51 2015 -0500

    The first string test is working!

commit 85c769eee192ac3583b57756cbab2244a5ac7d74
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Tue Nov 3 09:39:47 2015 -0500

    Added string variable type, but not tested.
    All tests pass except for the one using the class instance.

commit 826af954140ec28cb47329f8efeba90764869113
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sun Nov 1 16:07:21 2015 -0500

    Updated tests file formation to work with pythonic indentation.

commit eeb78d63c27f95e8c93ee4e48fcb16f512cae2ed
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Oct 31 19:51:10 2015 -0400

    Updated test script to display number of failures.
```

```
commit ce4336415042c3981a0a504a213193e67629eb9a
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Oct 31 09:36:01 2015 -0400

    Adding some files

commit 5a9f4893eb5e8bae1baed870b213f5786af397d7
Author: Carroll Loy <crl2131@columbia.edu>
Date:   Sat Oct 31 08:54:34 2015 -0400

    initial branch from microC
```

5 Architectural Design

5.1 Major Components



Geocode Source File
Preparser
Format Error
Encountered formatting error Report error to user and quit
Formatted Source File
Lexer
Syntax Error
Encountered syntax error Report error to user and quit
Tokens
Parser
Syntax Error
Encountered syntax error Report error to user and quit
AST
Encountered semantic error Report error to user and quit
Semantic Error
Semantic Check
G-Code Generator
Run-Time Error
Encountered run-time error Report error to user and quit
G-code File

5.2 Preparser

The preparser is responsible for taking the source file and creating a source file that can be parsed by the lexer. The responsibilities of the preparser are:

- Embed files that are referenced through include statements.
- Look for invalid include loops (recursion).
- Ensure all blocks are on the same indentation level.
- Ensure statements that require blocks have one defined.
- Ensure indentation levels are not increased/decreased at improper points in the program.

The preparser was written in python and is found in the file pre_parser.py.

5.3 Lexer

The lexer takes the formatted output of the preparser and breaks it into tokens. At this point, invalid characters will generate an error. The lexer is found in the file scanner.mll and compiled with ocamllex.

5.4 Parser

The parser generates an abstract syntax tree from the tokens generated by the lexer. The parser will identify any invalid sequence of token as defined by the grammar. The parser is found in the file parser.mly compiled with ocamlyacc.

5.5 Semantic Check

The Semantic Check evaluates the AST to ensure that it is semantically correct. The sematic check validates the following properties of the program:

- all return statements in a functions return same type
- assignments are of correct type
- all list items are same type
- all functions have the correct number of parameters

The code is found in sast.ml and is compiled with ocaml.

5.5 G-Code Generator

The G-Code Generator creates the final G-Code file. The source code is divided up into the following source files:

- utility.ml: Source file to contain the string functions and general utility functions.
- class.ml: Source file to contain all functions specific to class data object and all class data types.
- environment.ml: Source file to contain all environment functions and data types.
- lists.ml: Source file to contain all code related to list functions and data types.

The source code is found in file ncgenerator.ml and compiled with ocaml.

6 Test Plan

The test plan began with the mantra: “Only commit code when all tests pass.” Therefore, once I had a simple program working, I wanted to ensure that the tests always passed with each modification to the code base.

In addition, upon each promotion of new functionality to the repository, I would add at least one new test case. For this reason, the test suite contains 112 unique test files.

6.1 Test Harness

The test harness that was included in microc was also used in Geocode. Only slight modifications were made to this program. Therefore, the running of all the tests could be accomplished with the command:

```
./testall.sh
```

This shell script iterates through all the files in the tests folder, comparing the output generated by the test file to the reference file.

6.2 Test Cases

All of the tests are integration tests. The test file is a source file written in the Geocode language. This test file is compiled and then the output is validated against the output.

6.2.1 Test Division

The test cases are separated into the major components of the Geocode compiler. A separate folder was created for each component and test files were created under that folder. The test folders are:

- Classes: Test cases for all class related functionality.
- Functions: Test cases for all function related functionality.
- Includes: Test cases for the include directive.
- Lists: Test cases for all list related functionality.

- Preparser: Test cases to test the preparser.
- Semantics: Test cases to validate the sematic checker.
- Standard: Test cases for the standard library.
- Syntax: Test cases to guarantee the syntax of the language.
- Types: Test cases for all type related functionality.
- Working: Test cases to prove full working programs.

6.2.1 Success vs. Failure Tests

Geocode always produce an output, even on failure. I tried to provide descriptive error message that could be used to validate test cases. Therefore, all success and failure test cases have the exact same logic, compile the program and observe the output.

I tried to include 1 failure test case for about every 4 good test cases. In some cases, like the semantic checking, the test cases are almost exclusively failure test cases.

6.3 Test Samples

The following test samples are found in the tests/Working folder. These are full programs.

6.3.1 Olympic Rings (Working/test-1.geo)

```
/* Olympic Rings */
include "../Standard/standard.geo"

main():
       class Circle a, b, c, d, e
       class Config config

       /* Initialize origin and cutting/retract planes */
       config = Config([retract_height:5.0, cutting_height:-1.0, feed_rate:100.0])
       config.lcs.Translate(10.0, 20.0)

       /* Top row of olympic circles */
       a = Circle([config:config, radius:9.0, desc:"(Circle 1)"])

       b = a
       b.desc = "(Circle 2)"
       b.Translate(20.0, 0.0)

       c = b
```

```
        c.desc = "(Circle 3)"
        c.Translate(20.0, 0.0)

        /* Bottom row of olympic circles */
        d = a
        d.desc = "(Circle 4)"
        d.Translate(10.0, -10.0)

        e = d
        e.desc = "(Circle 5)"
        e.Translate(20.0, 0.0)

        /* Output initilization NC code */
        GCodeInit(config)

        /* Output NC code for each circle */
        a.Print()
        b.Print()
        c.Print()
        d.Print()
        e.Print()
```

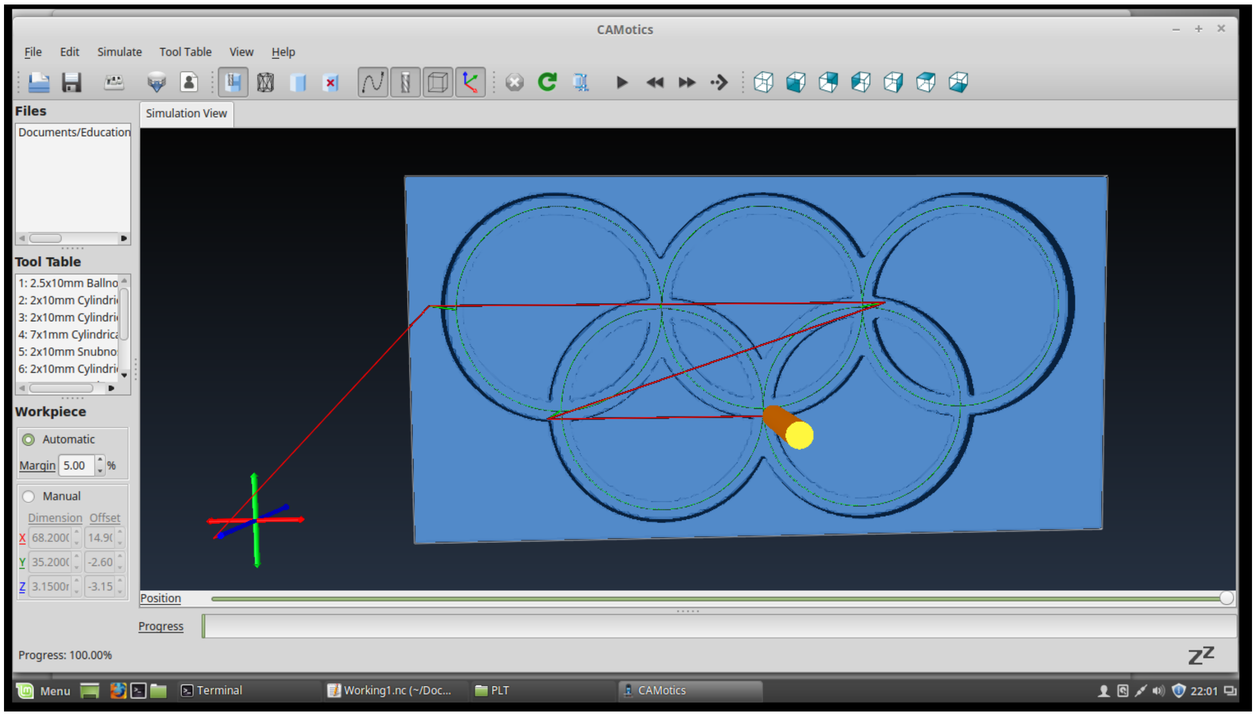
6.3.2 Olympic Rings Output

```
G21
T1
G0 Z5.
G21
(Circle 1)
G0 X19. Y20.
G1 Z-1. F100.
G2 X19. Y20. I10 J0 F100.
G0 Z5.
(Circle 2)
G0 X39. Y20.
G1 Z-1. F100.
G2 X39. Y20. I10 J0 F100.
G0 Z5.
(Circle 3)
G0 X59. Y20.
G1 Z-1. F100.
G2 X59. Y20. I10 J0 F100.
G0 Z5.
(Circle 4)
G0 X29. Y10.
G1 Z-1. F100.
G2 X29. Y10. I10 J0 F100.
G0 Z5.
(Circle 5)
G0 X49. Y10.
G1 Z-1. F100.
```

```
G2 X49. Y10. I10 J0 F100.
G0 Z5.
```

6.3.3 Olympic Rings Rendering

Using the program Camotics found at http://camotics.org/ I have rendered the NC Code:



6.4.1 Dragon Curve (Working/test-2.geo)

```
/* Dragon Curve */
include "../Standard/standard.geo"

main():
        class Math math
        class Config config
        class Polyline a, b
        int x

        /* Initialize origin and cutting/retract planes */
        config = Config([retract_height:5.0, cutting_height:-1.0, feed_rate:100.0])
        config.lcs.Identity()

        /* Create the first line for the dragon curve */
        a = Polyline([config:config, desc:"(Dragon Curve)"])
        a.AddPoint(Point([x:-10.0, y:0.0]), a.config.lcs)

        /* dragon curve iterations */
        for (x = 0; x < 12; x = x + 1):
               /* copy the previous geometries */
               b = a

               /* Create a new copy rotated 90 degrees around the end point */
               b.Reverse()
               b.Rotate(math.ToRad(-90.0))

               /* Append new copy to the original polyline */
               a.Append(b)

        /* Output initilization NC code */
        GCodeInit(config)

        /* Output NC code for dragon curve */
        a.Print()
```

6.4.2 Dragon Curve Output

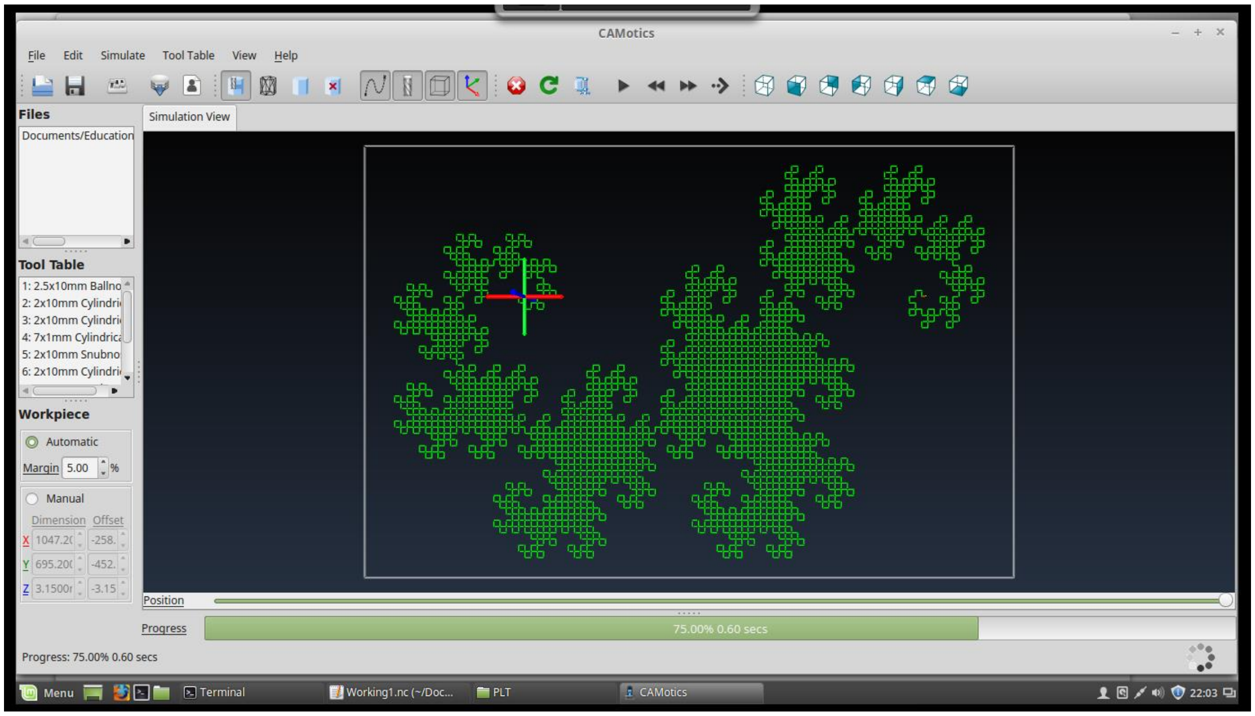
I have included the first few lines of the output, however, the entire output is over 4,000 lines.

```
G21
T1
G0 Z5.
G21
(Dragon Curve)
G0 X0. Y0.
G1 Z-1. F100.
G1 X-10. Y0. F100.
G1 X-10. Y-10. F100.
G1 X0. Y-10. F100.
G1 X0. Y-20. F100.
G1 X10. Y-20. F100.
G1 X10. Y-10. F100.
G1 X20. Y-10. F100.
G1 X20. Y-20. F100.
G1 X30. Y-20. F100.
G1 X30. Y-10. F100.
G1 X20. Y-10. F100.
G1 X20. Y0. F100.
G1 X30. Y0. F100.
G1 X30. Y10. F100.
G1 X40. Y10. F100.
G1 X40. Y0. F100.
G1 X50. Y0. F100.
```

```
G1 X50. Y10. F100.
G1 X40. Y10. F100.
...
```

6.4.3 Dragon Curve Rendering

Using the program Camotics found at http://camotics.org/ I have rendered the NC Code:



1: 2.5x10mm Ballno
2: 2x10mm Cylindri
3: 2x10mm Cylindri
4: 7x1mm Cylindric
5: 2x10mm Snubno
6: 2x10mm Cylindri

Margin 5.00 %

X 1047.2(-258.
Y 695.20(-452.
Z 3.1500r -3.15

75.00% 0.60 secs

Progress: 75.00% 0.60 secs

7 Lessons Learned

I learned the following lessons while completing this project:

- You can never start too early.
- Spend as much time as you can on this project from the beginning, you will regret leaving out fun features.
- Ocamlyacc and Ocamllex are amazing programs. While designing the grammar, I would frequently have ah-ha moments where everything seemed so easy. Almost too easy …
- Ocamlyacc and Ocamllex are frustrating in the beginning. The learning curve was quite sharp for me.
- The LRM is crucial. I am thankful that Professor Edwards made this a requirement early on, I would have not done it until the end if I was left to my own devices.

8 Appendix

8.1 Geocode Source Files

8.1.1 ast.ml

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

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

type var =
    Int
  | String
  | Bool
  | List of var
  | Double
  | ClassType of string

type var_decl = {
    vtype : var;
    vname : string;
  }

type literal =
    IntLiteral of int
  | BoolLiteral of bool
  | StringLiteral of string
  | DoubleLiteral of float
  | ClassLiteral of literal NameMap.t
  | ListLiteral of literal list
  | KVPLiteral of string * literal (* key value pair *)
  | LhsLiteral of string * literal (* Left hand side value *)

type expr =
    Literal of literal
  | Id of string
  | Binop of expr * op * expr
  | Assign of expr * expr
  | Call of string * expr list
  | ClassCall of expr * string * expr list
  | Access of expr * string
  | Noexpr
  | Negate of expr
  | Cast of var * expr
  | ListItems of list_item list
and
```

```
 list_item = {
    lkey : string;
    lvalue : expr;
}

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

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

type class_decl = {
    class_name : string;
    data_members : var_decl list;
    function_members : func_decl list;
  }

type program = var_decl list * func_decl list * class_decl list
```

8.1.2 scanner.mll

```
{
      open Utility
      open Parser

      (* Used to output error for unknown syntax *)
      exception Syntax_error of string
      let syntax_error msg = raise (Syntax_error (msg ^ " on line: " ^
(string_of_int !current_parse_line)))
}


(* code used to parse a double *)
let digit = ['0'-'9']
let sign = ['+' '-']
let exponent = ('e'(sign?)digit+)
let fpn =                          (* Floating point number *)
(digit*)'.'(digit+)(exponent?)    (* optional integer, required decimal, optional
exponent *)
| (digit+)'.'(digit*)(exponent?) (* required integer, optional decimal, optional
exponent *)
| (digit+)exponent                (* required integer, no decimal, required exponent
*)
```

```
let non_string_char = [^ '"' '\n' ]

rule token = parse
  [' ' '\t' '\r'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LSQBRACKET }
| ']'      { RSQBRACKET }
| ';'      { SEMI }
| ':'      { COLON }
| '@'      { AT }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "string" { STRING }
| "double" { DOUBLE }
| "bool"   { BOOL }
| "list"   { LIST }
| "class"  { CLASS }
| "true"   { TRUE }
| "false"  { FALSE }
| '.'      { ACCESS }
| "__BEGIN_OF_INCLUDE__" ([^ '\n']* as file_path) "!@<" { current_include_parse_file
:= file_path; incr current_parse_line; token lexbuf }
| "__START_MAIN_FILE__" ([^ '\n']* as file_path) "!@<" { main_parse_file :=
file_path; token lexbuf }
| "__END_OF_INCLUDE__" { current_include_parse_file := ""; current_parse_line := 3;
token lexbuf }
| ['0'-'9']+ as lxm { INTLITERAL(int_of_string lxm) }
| '"' ([^ '"' '\n' ]* as lxm) '"' { STRINGLITERAL(lxm) }
(* | '"' (['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm) '"' { STRINGLITERAL(lxm) } *)
| fpn as lit { DOUBLELITERAL(float_of_string lit) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
(* | ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as classname { CLASSTYPE(classname) } *)
| '\n' { incr current_parse_line; token lexbuf }
| eof { EOF }
(* | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) } *)
```

```
 | _ { syntax_error "couldn't identify the token" }

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

8.1.3 parser.mly

```
%{
      open Ast
      open Utility

      (* Used to output error for unknown syntax *)
      let parse_error s = (* Called by the parser function on error *)
        print_endline ("***** " ^ s ^ " in file: " ^ (get_parse_file_name
!main_parse_file) ^ ", on line: " ^ (string_of_int (!current_parse_line)));
        flush stdout
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR WHILE INT
%token COLON LSQBRACKET RSQBRACKET CLASS INT STRING DOUBLE BOOL LIST TRUE FALSE
ACCESS AT
%token <bool> BOOLLITERAL
%token <int> INTLITERAL
%token <string> ID STRINGLITERAL CLASSTYPE
%token <float> DOUBLELITERAL
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left ACCESS
%nonassoc UMINUS

%start program
%type <Ast.program> program

%%

/* Program consists of variable, function, and class declarations */
program:
   /* nothing */ { [], [], [] }
 | program vdecl { let (variables, functions, classes) = $1 in
   ($2 @ variables), functions, classes }
 | program fdecl { let (variables, functions, classes) = $1 in
    variables, ($2 :: functions), classes }
 | program cdecl { let (variables, functions, classes) = $1 in
    variables, functions, ($2 :: classes) }

fdecl_list:
    /* nothing */     { [] }
  | fdecl_list fdecl { $2 :: $1 }

fdecl:
      ID LPAREN formals_opt RPAREN COLON LBRACE vdecl_list stmt_list RBRACE
             { { fname = $1; formals = $3; locals = List.rev $7; body = List.rev $8 }
}


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

formal_list:
    var ID                  { [{ vtype = $1;
                                 vname = $2 }] }
  | formal_list COMMA var ID { { vtype = $3;
                                vname = $4 } :: $1 }

vdecl:
    var ID { [{ vtype = $1; vname = $2 }] }
 |  vdecl COMMA ID { { vtype = ((List.hd $1).vtype); vname = $3 } :: $1 }
 |  vdecl SEMI { $1 }

vdecl_list:
     { [] }
  | vdecl_list vdecl { $2 @ $1 }


var:
    INT      { Int }
  | STRING   { String }
  | BOOL     { Bool }
  | LIST LSQBRACKET var RSQBRACKET { List($3) }
  | DOUBLE   { Double }
  | CLASS ID { ClassType($2) }

cdecl:
   CLASS ID COLON LBRACE vdecl_list fdecl_list RBRACE
     { { class_name = $2;
        data_members = List.rev $5;
        function_members = List.rev $6 } }

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

stmt:
    expr SEMI { Expr($1) }
  | RETURN expr SEMI { Return($2) }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF expr COLON stmt %prec NOELSE { If($2, $4, Block([])) }
  | IF expr COLON stmt ELSE COLON stmt    { If($2, $4, $7) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN COLON stmt
     { For($3, $5, $7, $10) }
  | WHILE LPAREN expr RPAREN COLON stmt { While($3, $6) }

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

expr:
    INTLITERAL       { Literal(IntLiteral($1)) }
  | BOOLLITERAL      { Literal(BoolLiteral($1)) }
  | TRUE             { Literal(BoolLiteral(true)) }
  | FALSE            { Literal(BoolLiteral(false)) }
  | STRINGLITERAL    { Literal(StringLiteral($1)) }
  | DOUBLELITERAL    { Literal(DoubleLiteral($1)) }
  | ID               { Id($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | expr ACCESS ID LPAREN actuals_opt RPAREN { ClassCall($1, $3, $5) }
  | expr ACCESS ID   { Access($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | MINUS expr %prec UMINUS { Negate($2) }
  | LT var GT LPAREN expr RPAREN { Cast($2, $5) }
  | LSQBRACKET list_opt RSQBRACKET { ListItems($2) }

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

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

list_opt:
    /* nothing */ { [] }
  | list_items  { $1 }
  | list_nvp_items  { $1 }

list_items:
  |  expr { [ { lkey = "_"; lvalue = $1 } ] }
  | list_items COMMA expr { { lkey = "_"; lvalue = $3 } :: $1 }

list_nvp_items:
    ID COLON expr { [{ lkey = $1; lvalue = $3 }] }
  | list_nvp_items COMMA ID COLON expr { { lkey = $3; lvalue = $5 } :: $1 }
```

8.1.4 sast.ml

```
open Ast
open Utility
open Class
open Environment
open Lists


type sast_environment = {
    sast_typed  : literal NameMap.t;
    sast_untyped: literal NameMap.t;
    call_list   : string list;
  }

(* compare types to see if they can be used interchangably.  Otherwise, this is a
semantic error *)
let rec types_compatible lit1 lit2 =
      match lit1, lit2 with
            IntLiteral(lit1), IntLiteral(lit2) -> true
            | BoolLiteral(lit1), BoolLiteral(lit2) -> true
            | StringLiteral(lit1), StringLiteral(lit2) -> true
            | DoubleLiteral(lit1), DoubleLiteral(lit2) -> true
            | ClassLiteral(cls_map1), ClassLiteral(cls_map2) ->
                  get_class_name (ClassLiteral(cls_map1)) = get_class_name
(ClassLiteral(cls_map2))
            | ListLiteral(items1), ListLiteral(items2) ->
                  if (List.length items1) == 0 then
                        true
                  else if (List.length items2) == 0 then
                        true
                  else
                        types_compatible (List.hd items1) (List.hd items2)
            | KVPLiteral(key1, lit1), KVPLiteral(key2, lit2) -> types_compatible
lit1 lit2
            | (LhsLiteral(var, lit), (_ as second)) -> types_compatible lit second
            | ((_ as first), LhsLiteral(var, lit)) -> types_compatible first lit
            | _, _ -> false

(* Typed functions must end in a return *)
let rec statment_end_in_return = function
    Block(stmts) ->
            if (List.length stmts > 0) then
                  statment_end_in_return (List.nth stmts ((List.length stmts) - 1))
            else
                  false
  | Return(expr) -> true
  | If(e, s1, s2) ->  (statment_end_in_return s1) && (statment_end_in_return s2)
  | _ -> false
```

```
let eval_actual_list (actuals, env, sast_env, eval) =
	List.fold_left
		(fun (actuals, env, sast_env) actual ->
			let v, (env, sast_env) = eval (env, sast_env) actual in
			v :: actuals, env, sast_env)
		([], env, sast_env) (List.rev actuals)

(* ensure function return types match up *)
let check_fdecl (check, fdecl, actuals, env, sast_env) =
	let sast_env_pre_call_list = sast_env.call_list in
	let sast_env = { sast_env with call_list = fdecl.fname::sast_env.call_list }
in
	let (env, sast_env) = (check fdecl actuals env sast_env) in
	let sast_env = {sast_env with call_list = sast_env_pre_call_list} in
	let env = { env with env_locals = NameMap.empty } in
	if NameMap.mem fdecl.fname sast_env.sast_typed then
(* let test = print_endline ("TYPED: " ^ fdecl.fname ^ ": " ^ (string_of_literal
(NameMap.find fdecl.fname sast_env.sast_typed))) in *)
		let statement_size = List.length fdecl.body in
		if statement_size > 0 then
			let last_statement = (List.nth fdecl.body (statement_size - 1))
in
(* let test = print_endline (string_of_stmt last_statement) in *)
			if (statment_end_in_return last_statement) then
				(NameMap.find fdecl.fname sast_env.sast_typed), (env,
sast_env)
			else
				raise(SemanticFailure("Function contains return
statements, but does not end in one: " ^ fdecl.fname))
		else
			(NameMap.find fdecl.fname sast_env.sast_typed), (env, sast_env)
	else if NameMap.mem fdecl.fname sast_env.sast_untyped then
(* let test = print_endline ("UNTYPED: " ^ fdecl.fname) in *)
		(NameMap.find fdecl.fname sast_env.sast_untyped), (env, sast_env)
	else
(* let test = print_endline ("UNKNOWN: " ^ fdecl.fname) in *)
		let sast_env = { sast_env with sast_untyped = NameMap.add (fdecl.fname)
(IntLiteral(0)) sast_env.sast_untyped } in
		IntLiteral(0), (env, sast_env)

let cls_check_func (check, env, func_decls, func_name, actuals, cls_lit, call_fdecl,
sast_env) =
	let fdecl = find_fdecl (func_name, func_decls) in
	let cls_context = ((get_lhs_varname cls_lit), (get_class_value_map cls_lit))
in
	let call_env = {env with env_locals = NameMap.empty; env_context =
cls_context} in
	let call_lit, (post_call_env, sast_env) = check_fdecl (check, fdecl, actuals,
call_env, sast_env) in
	let err_msg = "Unknown context: " ^ (fst call_env.env_context) in
	let return_env = update_env_class_call (env, post_call_env, err_msg) in
	call_lit, (return_env, sast_env)
```

```
(*
	Semantic Check:
	all return statements in a functions return same type
	assignments are of correct type
	all list items are same type
*)

let semantic_check (vars, funcs, classes) =
	let func_decls = map_funcs (funcs, classes) in
	(* Check by functions and return an updated global symbol table *)
	let rec check fdecl actuals call_env sast_env =
		(* Evaluate an expression and return (value, updated environment) *)
		let rec eval (env, sast_env) = function
			Literal(i) -> i, (env, sast_env)
			| Noexpr -> IntLiteral(1), (env, sast_env)
			| Id(var) ->
				let ret_lit, env = id_variable_lookup (var, env, "ID: 
undeclared identifier ") in
				ret_lit, (env, sast_env)
			| Cast(v_type, e1) ->
				let v1, (env, sast_env) = eval (env, sast_env) e1 in
				(cast_of_literal (v_type, v1)), (env, sast_env)
			| Negate(e1) ->
				let v1, (env, sast_env) = eval (env, sast_env) e1 in
				(negate_of_literal v1), (env, sast_env)
			| Binop(e1, op, e2) ->
				let v1, (env, sast_env) = eval (env, sast_env) e1 in
				let v2, (env, sast_env) = eval (env, sast_env) e2 in
				let bool_compare_exception = (fun x y -> raise 
(Failure("Boolean greater/less than operators not supported"))) in
				(match op with
					Add ->
						if ((is_list_literal v1) && (is_list_literal 
v2)) then
							list_concat (v1, v2)
						else
							do_literal_operation (+) (+.) (^) 
(v1,v2)
					| Sub -> do_literal_operation (-) (-.) (fun x y -> 
raise (Failure("String subtract not supported"))) (v1,v2)
					| Mult -> do_literal_operation ( * ) ( *. ) (fun x y 
-> raise (Failure("String multiply not supported"))) (v1,v2)
					| Div -> do_literal_operation ( / ) ( /. ) (fun x y 
-> raise (Failure("String divide not supported"))) (v1,v2)
					| Equal -> do_literal_compare (==) (=) (=) (==) 
(v1,v2)
					| Neq -> do_literal_compare (!=) (<>) (<>) (!=) 
(v1,v2)
					| Less -> do_literal_compare (<) (<) (<) 
(bool_compare_exception) (v1,v2)
					| Leq -> do_literal_compare (<=) (<=) (<=) 
(bool_compare_exception) (v1,v2)
					| Greater -> do_literal_compare (>) (>) (>) 
(bool_compare_exception) (v1,v2)
```

```
                                | Geq -> do_literal_compare (>=) (>=) (>=)
(bool_compare_exception) (v1,v2)), (env, sast_env)
                    | ListItems(items) ->
                          let check_list_literal item  =
                                let v1, (env, sast_env) = eval (env, sast_env)
item.lvalue in
                                if item.lkey = "_" then
                                      v1, (env, sast_env)
                                else
                                      KVPLiteral(item.lkey, v1), (env, sast_env)
                          in
                          if (List.length items) = 0 then
                                ListLiteral([]), (env, sast_env)
                          else
                                let ret_lit, (env, sast_env) = check_list_literal
(List.hd items) in
                                ListLiteral([ret_lit]), (env, sast_env)
                    | Assign(e_var, e_val) ->
                          let v_var, (env, sast_env) = eval (env, sast_env) e_var in
                          let v_val, (env, sast_env) = eval (env, sast_env) e_val in
                          if is_lhs_literal v_var then
                                let var_name = (get_lhs_varname v_var) in
                                if types_compatible v_var v_val then
                                      let ret_lit, env = assign_variable_lookup
(var_name, v_val, env, "Assign: undeclared identifier ") in
                                      ret_lit, (env, sast_env)
                                else
                                      raise(SemanticFailure("Cannot assign variable
to an incompatible type: " ^ var_name))
                          else
                                raise(SemanticFailure("Cannot assign to the
expression: " ^ (string_of_expr e_var)))
                    | Call("print", [e]) ->
                          let v, (env, sast_env) = eval (env, sast_env) e in
                          IntLiteral(0), (env, sast_env)
                    | Call("cos", [e]) ->
                          let v, (env, sast_env) = eval (env, sast_env) e in
                          DoubleLiteral(cos (double_of_literal v)), (env, sast_env)
                    | Call("sin", [e]) ->
                          let v, (env, sast_env) = eval (env, sast_env) e in
                          DoubleLiteral(sin (double_of_literal v)), (env, sast_env)
                    | Call("sqrt", [e]) ->
                          let v, (env, sast_env) = eval (env, sast_env) e in
                          DoubleLiteral(sqrt (double_of_literal v)), (env, sast_env)
                    | ClassCall (e, cls_func, actuals) ->
                          let cls_lit, (env, sast_env) = eval (env, sast_env) e in
                          let actuals, env, sast_env = eval_actual_list (actuals,
env, sast_env, eval) in
                          if (is_list_literal cls_lit) = true then
                                let ret_lit, env = check_list_func (cls_lit,
cls_func, actuals, env) in
                                ret_lit, (env, sast_env)
                          else
                                let class_name = (get_class_name cls_lit) in
                                let func_name = class_name ^ "." ^ cls_func in
                                cls_check_func (check, env, func_decls, func_name,
actuals, cls_lit, check_fdecl, sast_env)
                    | Access (e, member) ->
                          let cls_lit, (env, sast_env) = eval (env, sast_env) e in
                          if is_lhs_literal cls_lit then
                                id_cls_variable_lookup ((get_lhs_varname cls_lit),
member, env, "undeclared access identifier: "), (env, sast_env)
                          else
                                raise(SemanticFailure("Cannot use access operator on
expression: " ^ (string_of_expr e)))
                    | Call(func, actuals) ->
                          let get_context_cls_func =
                                if NameMap.cardinal (snd env.env_context) = 0 then
                                      ""
                                else
                                      let cls_name = (get_class_name
(ClassLiteral(snd env.env_context))) in
                                      let cls_func_name = cls_name ^ "." ^ func in
                                      if NameMap.mem cls_func_name func_decls then
                                            cls_func_name
                                      else
                                            "" in
                          let cls_func_name = get_context_cls_func in
                          if String.length cls_func_name > 0 then
                                let cls_lit = LhsLiteral((fst env.env_context),
ClassLiteral(snd env.env_context)) in
                                let actuals, env, sast_env = eval_actual_list
(actuals, env, sast_env, eval) in
                                cls_check_func (check, env, func_decls,
cls_func_name, actuals, cls_lit, check_fdecl, sast_env)
                          else
                                let actuals, env, sast_env = eval_actual_list
(actuals, env, sast_env, eval) in

                                try
                                      let fdecl = find_fdecl (func, func_decls) in
                                      let call_env = { env with env_locals =
NameMap.empty; env_context = get_def_env_context } in
                                      let call_lit, (post_call_env, sast_env) =
check_fdecl (check, fdecl, actuals, call_env, sast_env) in
                                      call_lit, ({env with env_globals =
post_call_env.env_globals}, sast_env)
                                with FunctionNotFoundException s ->
                                      try
                                            let ret_lit, env =
try_func_as_class_constructor (actuals, env, classes, func, init_var) in
                                            ret_lit, (env, sast_env)
                                      with Failure s1 ->
                                            raise (FunctionNotFoundException(s))
              in
              (* Execute a statement and return an updated environment *)
              let rec exec (env, sast_env) = function
                    Block(stmts) -> List.fold_left exec (env, sast_env) stmts
                    | Expr(e) -> let _, (env, sast_env) = eval (env, sast_env) e in
(env, sast_env)
```

```
                    | If(e, s1, s2) ->
                          let v, (env, sast_env) = eval (env, sast_env) e in
                          let (env, sast_env) = exec (env, sast_env) s1 in
                          exec (env, sast_env) s2
                    | While(e, s) ->
                          let rec loop env =
                                let v, (env, sast_env) = eval (env, sast_env) e in
                                if (bool_of_literal v) then
                                      (exec (env, sast_env) s)
                                else (env, sast_env)
                          in loop env
                    | For(e1, e2, e3, s) ->
                          let _, (env, sast_env) = eval (env, sast_env) e1 in
                          let rec loop env =
                                let v, (env, sast_env) = eval (env, sast_env) e2 in
                                if (bool_of_literal v) then
                                let (env, sast_env) = exec (env, sast_env) s in
                                let _, (env, sast_env) = eval (env, sast_env) e3 in
                                      (env, sast_env)
                                else
                                      (env, sast_env)
                          in loop env
                    | Return(e) ->
                          let current_func = (List.hd sast_env.call_list) in
                          let matches = List.find_all (fun name -> current_func =
name) sast_env.call_list in
                          if ((List.length matches) < 3) then
                                let v, (env, sast_env) = eval (env, sast_env) e in
                                if (NameMap.mem current_func sast_env.sast_typed)
then
                                      if types_compatible v (NameMap.find
current_func sast_env.sast_typed) then
                                            (env, sast_env)
                                      else
                                            raise (SemanticFailure("Incompatible
return types found for function: " ^current_func))
                                else
                                      let sast_env = {sast_env with sast_typed =
NameMap.add current_func v sast_env.sast_typed } in
                                      (env, sast_env)
                          else
                                (env, sast_env)
(*                        let v, env = eval env e in
                          raise (ReturnException(v, env)) *)
              in

              (* Enter the function: bind actual values to formal arguments *)
              let locals =
                    try List.fold_left2
                          (fun locals formal actual -> NameMap.add formal.vname
actual locals)
                          NameMap.empty fdecl.formals actuals
                    with Invalid_argument(_) -> raise (Failure ("wrong number of
arguments passed to " ^ fdecl.fname))
              in
```

```
            (* Initialize local variables to 0 *)
            let locals = List.fold_left
                  (fun accum local -> NameMap.add local.vname (init_var classes
local.vtype) accum)
                  locals fdecl.locals
            in
            let locals = NameMap.fold
                  (fun key value accum -> NameMap.add key value accum)
                  call_env.env_locals locals
            in
            (* Execute each statement in sequence, return updated global symbol
table *)
            (List.fold_left exec ({ call_env with env_locals = locals }, sast_env)
fdecl.body)

      (* Run a program: initialize global variables to 0, find and run "main" *)
      in let globals = List.fold_left
            (fun globals vdecl -> NameMap.add vdecl.vname (init_var classes
vdecl.vtype) globals)
            NameMap.empty vars
      in try
            check (NameMap.find "main" func_decls) []
                  { env_globals = globals;
                    env_locals = NameMap.empty;
                    env_context = get_def_env_context }
                  { sast_typed = NameMap.empty;
              sast_untyped = NameMap.empty;
                    call_list = [] }
      with Not_found -> raise (Failure ("did not find the main() function"))
```

8.1.5 utility.ml

```
open Ast

exception FunctionNotFoundException of string
exception ClassConstructorException of string
exception SemanticFailure of string


(* ****************** Syntax File Parsing variables ****************** *)
exception Syntax_error of string
let current_parse_line = ref 0
let current_include_parse_file = ref ""
let main_parse_file = ref ""
let get_parse_file_name = (fun x -> if !current_include_parse_file = "" then x else
!current_include_parse_file)

(* ****************** Literal String Functions ****************** *)

(* return the number of tab chars specified *)
let rec get_tabs count =
      if count = 0 then
            ""
```

```
	else
		"\t" ^ (get_tabs (count - 1))

let string_of_literal lit =
	let rec rec_string_of_literal embed_count = function
		IntLiteral(lit)    -> string_of_int lit
		| BoolLiteral(lit) -> string_of_bool lit
		| StringLiteral(lit) -> lit
		| DoubleLiteral(lit) -> string_of_float lit
		| ClassLiteral(cls_map) ->
			NameMap.fold (fun key value accum -> accum ^ " \n" ^ (get_tabs
(embed_count))
				^ key ^ " " ^ (rec_string_of_literal (embed_count + 1)
value)) cls_map ""
		| LhsLiteral(var, lit) -> (rec_string_of_literal embed_count lit)
		| ListLiteral(items) ->  "[" ^ (String.concat ", " (List.map
(rec_string_of_literal embed_count) items)) ^ "]"
		| KVPLiteral(key, lit) -> key ^ ": " ^ (rec_string_of_literal
embed_count lit)
	in
	(rec_string_of_literal 0 lit)

let rec is_list_literal = function
	ListLiteral(items) ->  true
	| LhsLiteral(var, lit) -> (is_list_literal lit)
	| _ -> false

let rec get_list_items = function
	ListLiteral(items) ->  items
	| LhsLiteral(var, lit) -> (get_list_items lit)
	| _ -> raise(Failure("Type is not a list"))


let string_of_var = function
    Int    -> "Int"
  | Bool -> "Bool"
  | List(var) -> "List"
  | String -> "String"
  | Double -> "Double"
  | ClassType(classname) -> classname

let string_of_var_decl vdecl =
    string_of_var vdecl.vtype ^ " " ^ vdecl.vname ^ ";\n"

let rec string_of_expr = function
    Literal(lit) -> string_of_literal lit
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^
      (match o with
	Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
      | Equal -> "==" | Neq -> "!="
      | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">=") ^ " " ^
      string_of_expr e2
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | ClassCall(class_instance, f, el) ->
      string_of_expr class_instance ^ " . " ^ f ^ "(" ^ String.concat ", " (List.map
string_of_expr el) ^ ")"
  | Access(e, member) -> string_of_expr e ^ " . " ^ member
  | Noexpr -> ""
  | Negate(e) -> "Negation of " ^ string_of_expr e
  | Cast(vari, e) -> "Cast of " ^ (string_of_expr e) ^ " to " ^ (string_of_var vari)
  | ListItems(items) -> "List Items: " ^ String.concat "\n"
                                        (List.map (fun item -> item.lkey ^ ": " ^
string_of_expr item.lvalue) items)


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

let string_of_fdecl fdecl =
  fdecl.fname ^ "(" ^
  String.concat "" (List.map string_of_var_decl fdecl.locals) ^ ")\n{\n" ^
  String.concat "" (List.map string_of_var_decl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_odecl odecl =
  odecl.class_name ^ "\n{\n" ^
  String.concat "" (List.map string_of_var_decl odecl.data_members) ^
  String.concat "" (List.map string_of_fdecl odecl.function_members) ^
  "}\n"

let string_of_program (vars, funcs, classes) =
  String.concat "" (List.map string_of_var_decl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_odecl classes)


(* ****************** Literal Utility Functions ****************** *)

(* convert literal into a boolean *)
let rec bool_of_literal = function
      IntLiteral(lit) -> (lit != 0)
      | BoolLiteral(lit) -> lit
      | ListLiteral(lit) -> (List.length lit) != 0
      | StringLiteral(lit) -> (lit <> "")
      | DoubleLiteral(lit) -> ((abs_float lit) > epsilon_float)
      | LhsLiteral(var, lit) -> bool_of_literal lit
    | _ -> raise (Failure("Class boolean operation not supported"))

(* perform the passed in binary operation on the literal type *)
let rec do_literal_operation int_op double_op string_op (v1, v2) =
  match v1,v2 with
      (IntLiteral(lit1), IntLiteral(lit2)) ->  IntLiteral(int_op lit1 lit2)
      | (DoubleLiteral(lit1), DoubleLiteral(lit2)) ->  DoubleLiteral(double_op lit1
lit2)
      | (StringLiteral(lit1), StringLiteral(lit2)) ->  StringLiteral(string_op lit1
lit2)
      | (LhsLiteral(var, lit), (_ as second)) -> do_literal_operation (int_op)
(double_op) (string_op) (lit, second)
      | ((_ as first), LhsLiteral(var, lit)) -> do_literal_operation (int_op)
(double_op)  (string_op) (first, lit)
      | (_ as first), (_ as second) -> raise (Failure("Binary operations not
supported for: " ^

      (string_of_literal first) ^ " and " ^ (string_of_literal second)))

(* perform the passed in comparison operation on the literal type *)
let rec do_literal_compare int_op double_op string_op bool_op (v1, v2) =
      match v1,v2 with
            (IntLiteral(lit1), IntLiteral(lit2)) ->  BoolLiteral(int_op lit1 lit2)
            | (DoubleLiteral(lit1), DoubleLiteral(lit2)) ->  BoolLiteral(double_op
lit1 lit2)
            | (StringLiteral(lit1), StringLiteral(lit2)) ->  BoolLiteral(string_op
lit1 lit2)
            | (BoolLiteral(lit), (_ as second)) -> BoolLiteral(bool_op lit
(bool_of_literal second))
            | ((_ as first), BoolLiteral(lit)) -> BoolLiteral(bool_op lit
(bool_of_literal first))
            | (LhsLiteral(var, lit), (_ as second)) -> do_literal_compare (int_op)
(double_op) (string_op) (bool_op) (lit, second)
            | ((_ as first), LhsLiteral(var, lit)) -> do_literal_compare (int_op)
(double_op) (string_op) (bool_op) (first, lit)
            | _ -> raise (Failure("Comparison not supported for this type"))

let rec negate_of_literal = function
      IntLiteral(lit)    -> IntLiteral(-lit)
      | DoubleLiteral(lit) -> DoubleLiteral(-.lit)
      | LhsLiteral(var, lit) -> negate_of_literal lit
      | _ -> raise (Failure("Negation not defined for this type"))

let rec cast_of_literal (v_type, v1) =
      match v_type, v1 with
            String, IntLiteral(lit) -> StringLiteral(string_of_int lit)
            | String, DoubleLiteral(lit) -> StringLiteral(string_of_float lit)
            | _, LhsLiteral(var, lit) -> cast_of_literal (v_type, lit)
            | _, _ -> raise (Failure("Casting not defined for this type"))

let rec double_of_literal = function
      DoubleLiteral(lit) -> lit
      | LhsLiteral(var, lit) -> double_of_literal lit
      | _ -> raise (Failure("Not a double type"))
```

```
(* ****************** List FUNCTIONS ****************** *)

(* a list item can be a value or a key value pair *)
let get_list_literal item eval env =
      let v1, env = eval env item.lvalue in
      if item.lkey = "_" then
            v1, env
      else
            KVPLiteral(item.lkey, v1), env

let get_list = function
      ListLiteral(items) -> items
      | _ -> raise (Failure("Class constructors must be initialized with key value
pairs"))


(* ****************** Utility FUNCTIONS ****************** *)

let get_lhs_varname = function
  LhsLiteral(var, lit) -> var
    | _ as lit -> raise (Failure("Variable is NOT a LHS:" ^ string_of_literal lit))

let is_lhs_literal = function
  LhsLiteral(var, lit) -> true
    | _ -> false



(* return a list of strings that were separated by the delimiter *)
let rec rec_split_string delim str_test id_list =
  try
    let delim_index = String.index str_test delim in
    let part1 = (String.sub str_test 0 delim_index) in
    let part2 = (String.sub str_test (delim_index + 1) ((String.length str_test) -
(delim_index + 1))) in
      (rec_split_string delim part2 (part1::id_list))
  with Not_found -> if (String.length str_test) != 0 then (List.rev
(str_test::id_list)) else (List.rev id_list)

let rec insert_item (item, index) = function
      [] ->
            if index = 0 then
                  [item]
            else
                  raise(Failure("Insert index is invalid"))
      | hd::tl as lst ->
            if index = 0 then
                  item::lst
            else
                  hd::(insert_item (item, (index - 1)) tl)

let rec remove_item index = function
```

```
      [] ->
            raise(Failure("Remove index is invalid"))
      | hd::tl ->
            if index = 0 then
                  tl
            else
                  hd::(remove_item (index - 1) tl)

let rec pop_item ret_lst = function
      [] ->
            raise(Failure("Cannot pop an empty list"))
      | hd::[] ->
            ([], hd)
      | hd::tl ->
            if List.length tl = 1 then
                  (List.rev (hd::ret_lst), List.hd tl)
            else
                  pop_item (hd::ret_lst) tl
```

8.1.6 class.ml

```
open Ast
open Utility


(* ****************** CLASS FUNCTIONS ****************** *)

(* Initalize a class object *)
let get_class_default classes classname init_var =
      try
        let cls_decl = List.find (fun cls_decl -> cls_decl.class_name = classname)
classes in
        let cls_data = List.fold_left
            (fun accum member -> NameMap.add member.vname (init_var classes
member.vtype) accum)
            NameMap.empty  cls_decl.data_members in
        ClassLiteral(NameMap.add "_cls.Name" (StringLiteral(classname)) cls_data)
      with Not_found -> raise (Failure ("Class not found " ^ classname))

let rec init_var class_map vtype =
  match vtype with
    Int -> IntLiteral(0)
    | Bool -> BoolLiteral(false)
    | List(var) -> ListLiteral([])
    | String -> StringLiteral("")
    | Double -> DoubleLiteral(0.0)
    | ClassType(classname) ->  get_class_default class_map classname init_var

(* find the class name *)
let rec get_class_name = function
```

```
      IntLiteral(lit) -> raise (Failure ("Int is not a class object: " ^
string_of_int lit))
      | BoolLiteral(lit) -> raise (Failure ("Bool is not a class object: " ^
string_of_bool lit))
      | StringLiteral(lit) -> raise (Failure ("String is not a class object: " ^
lit))
      | DoubleLiteral(lit) -> raise (Failure ("Double is not a class object: " ^
string_of_float lit))
      | LhsLiteral(var, lit) -> get_class_name lit
      | ListLiteral(items) ->  raise (Failure ("List is not a class object"))
      | KVPLiteral(key, lit)-> raise (Failure ("KVP is not a class object"))
      | ClassLiteral(cls_map) as cls_lit ->
            try
                  let cls_name = NameMap.find "_cls.Name" cls_map in
                  string_of_literal cls_name
            with Not_found -> raise (Failure ("Class not intialized correctly: " ^
(string_of_literal cls_lit) ))

(* get a class member map *)
let rec get_class_value_map = function
  IntLiteral(lit) -> raise (Failure ("Int is not a class object: " ^ string_of_int
lit))
  | BoolLiteral(lit) -> raise (Failure ("Bool is not a class object: " ^
string_of_bool lit))
  | StringLiteral(lit) -> raise (Failure ("String is not a class object: " ^ lit))
  | DoubleLiteral(lit) -> raise (Failure ("Double is not a class object: " ^
string_of_float lit))
  | LhsLiteral(var, lit) -> get_class_value_map lit
  | ListLiteral(items) ->  raise (Failure ("List is not a class object"))
  | KVPLiteral(key, lit)-> raise (Failure ("KVP is not a class object"))
  | ClassLiteral(cls_map) -> cls_map

(* get a class member value *)
let get_class_value (member, lit) =
  try
    NameMap.find member (get_class_value_map lit)
  with Not_found -> raise (Failure ("Class does not contain member: " ^ member))


(* ****************** CLASS fdecl FUNCTIONS ****************** *)

(* See if the function is a class type and create an instance of that class *)
let try_func_as_class_constructor (actuals, env, classes, func, init_var) =
      let cls_lit = (get_class_default classes func init_var) in
      if (List.length actuals) = 0 then
            cls_lit, env
      else if (List.length actuals) != 1 then
            raise (ClassConstructorException("Class constructors must be initialized
with key value pairs"))
      else
            (* parse key (member name) value pair list and set class data *)
            let get_kvp = function
                  KVPLiteral(key, lit) -> (key, lit)
                  | _ -> raise (ClassConstructorException("Class constructors must
be initialized with key value pairs")) in
```

```
             let tmp_items = (get_list (List.hd actuals)) in
             let key_value_pairs = List.map get_kvp tmp_items in
             let cls_map = get_class_value_map cls_lit in
             let cls_map = List.fold_left
                                        (fun accum item ->
                                               if NameMap.mem (fst item) accum then
                                                      NameMap.add (fst item) (snd
item) accum
                                               else
                                                      raise
(ClassConstructorException("Class constructor key does not exist: " ^ (fst item))) )
                                        cls_map key_value_pairs in
             ClassLiteral(cls_map), env
```

8.1.7 environment.ml

```
open Ast
open Utility
open Class

type environment = {
    env_globals  : literal NameMap.t;
    env_locals   : literal NameMap.t;
    env_context : string * literal NameMap.t; (* context (if inside a class function
call *)
  }

exception ReturnException of literal * environment

(* ****************** envirnoment FUNCTIONS ****************** *)

(* by default, there is no context.  Only set when inside class function call *)
let get_def_env_context = ("", NameMap.empty)

(* look at locals first, then at class scope, finally look in globals *)
let variable_lookup (var_name, env, err_msg) =
  if NameMap.mem var_name env.env_locals then
    env.env_locals
  else if NameMap.mem var_name (snd env.env_context) then
    (snd env.env_context)
  else if NameMap.mem var_name env.env_globals then
    env.env_globals
  else raise (Failure (err_msg ^ var_name))

let id_variable_lookup (var_name, env, err_msg) =
      let map_match = variable_lookup (var_name, env, err_msg) in
      try
             LhsLiteral(var_name, (NameMap.find var_name map_match)), env
      with Not_found -> raise (Failure ("Unable to look up variable" ^ var_name))
```

```
(* recurse through a list of class variables and find the nested member variable *)
let rec rec_lst_cls_variable_lookup cls_data_map = function
	[] -> raise (Failure ("Internal error: Variable lookup failure"))
	| hd::[] -> raise (Failure ("Internal error: Variable lookup failure"))
	| hd::tl ->
		let cls_member_name = (List.hd tl) in
		if NameMap.mem cls_member_name cls_data_map then
			try
				let match_data = (NameMap.find cls_member_name
cls_data_map) in
				if (List.length tl) = 1 then
					match_data
				else
					rec_lst_cls_variable_lookup (get_class_value_map
match_data) tl
			with Not_found -> raise (Failure ("Unable to find class member "
^ cls_member_name))
		else raise (Failure ("WHAAAT"))

let id_cls_variable_lookup (in_cls_var_name, cls_member_name, env, err_msg) =
	(* split the variable name on the access operator (.) *)
	let split_var_name = (rec_split_string '.' in_cls_var_name []) in
	let cls_var_lst = List.rev (cls_member_name::(List.rev split_var_name)) in
	let cls_var_name = (List.hd cls_var_lst) in
	let map_match = variable_lookup (cls_var_name, env, err_msg) in
	try
		(* recurse through the class members to find the matching variables *)
		let cls_data_map = get_class_value_map (NameMap.find cls_var_name
map_match) in
		let long_var_name = in_cls_var_name ^ "." ^ cls_member_name in
		try
			let cls_lit = (rec_lst_cls_variable_lookup cls_data_map
cls_var_lst) in
			let lit = LhsLiteral(long_var_name, cls_lit) in
			lit
		with Failure s -> raise (Failure (err_msg ^ long_var_name))
	with Not_found -> raise (Failure ("Unable to find member in class: " ^
cls_var_name))

(* recurse through a list of class variables and find the nested member variable *)
let rec rec_cls_assign_variable_lookup (cls_data_map, v_val) = function
	[] -> raise (Failure ("Internal error: Variable assignment failure"))
	| hd::[] ->
		if NameMap.mem hd cls_data_map then
			ClassLiteral(NameMap.add hd v_val cls_data_map)
		else raise (Failure ("Internal error: Variable assignment failure"))
	| hd::tl ->
		if NameMap.mem hd cls_data_map then
			let child_cls_data_map = get_class_value_map (NameMap.find hd
cls_data_map) in
			let cls_lit = (rec_cls_assign_variable_lookup
(child_cls_data_map, v_val) tl) in
			ClassLiteral(NameMap.add hd cls_lit cls_data_map)
		else raise (Failure ("Internal error: Variable assignment failure"))
```

```
let rec assign_variable_lookup (var_name, v_val, env, err_msg) =
	(* split the variable name on the access operator (.) *)
	let split_var_name = (rec_split_string '.' var_name []) in
	if (List.length split_var_name) > 1 then
		(* recurse through the class members to find the matching variables *)
		let class_var_name = (List.hd split_var_name) in
		let map_match = variable_lookup (class_var_name, env, err_msg) in
		try
			let cls_data_map = get_class_value_map (NameMap.find
class_var_name map_match) in
			let cls_val = rec_cls_assign_variable_lookup (cls_data_map,
v_val) (List.tl split_var_name) in
			let _, env = assign_variable_lookup(class_var_name, cls_val, env,
err_msg) in
			v_val, env
		with
			| Not_found -> raise (Failure ("Could not find class variable: "
^ class_var_name))
			| Failure s -> raise (Failure ("Could not find class variable: "
^ class_var_name))
	else
		(* find the class member and update value *)
		if NameMap.mem var_name env.env_locals then
			v_val, {env with env_locals = NameMap.add var_name v_val
env.env_locals }
		else if NameMap.mem var_name (snd env.env_context) then
			let context = ((fst env.env_context), (NameMap.add var_name v_val
(snd env.env_context))) in
			v_val, {env with env_context = context}
		else if NameMap.mem var_name env.env_globals then
			v_val, {env with env_globals = NameMap.add var_name v_val
env.env_globals}
		else raise (Failure (err_msg ^ var_name))

(* update the environment after a class function call *)
let update_env_class_call (pre_call_env, post_call_env, err_msg) =
	(* split the function name on the access operator (.) *)
	let split_var_name = (rec_split_string '.' (fst post_call_env.env_context) [])
in
	let var_name = (List.hd split_var_name) in
	let rec rec_update_local_map (cls_map, original_cls_map) = function
		[] -> raise (Failure ("Internal error: Update env contains invalid
data"))
		| hd::[] ->
			(* Reached the member variable, so update the class map *)
			let updated_cls_value = ClassLiteral((snd
post_call_env.env_context))  in
			ClassLiteral(NameMap.add hd updated_cls_value original_cls_map)
		| hd::tl ->
			(* Update member class object *)
			let child_cls_data_map = get_class_value_map (NameMap.find hd
cls_map) in
			let cls_lit = (rec_update_local_map (child_cls_data_map,
child_cls_data_map) tl) in
```

```
                    (* return the new object with the updated class member map *)
                    ClassLiteral(NameMap.add hd cls_lit cls_map)
        in
        let update_local_map var_map =
                (* if this is not a member class object update *)
                if (List.length split_var_name) = 1 then
                        let cls_map = get_class_value_map (NameMap.find var_name var_map)
in
                        let updated_cls_value = ClassLiteral(NameMap.fold
                                (fun key value accum -> NameMap.add key value accum)
                                (snd post_call_env.env_context) cls_map) in
                        NameMap.add var_name updated_cls_value var_map
                else
                        let cls_map = get_class_value_map (NameMap.find var_name var_map)
in
                        let updated_cls_value = (rec_update_local_map (cls_map, cls_map)
(List.tl split_var_name)) in
                        NameMap.add var_name updated_cls_value var_map
        in
        try
                (* Update of a local variable *)
                if NameMap.mem var_name pre_call_env.env_locals then
                        { pre_call_env with
                                env_locals = (update_local_map pre_call_env.env_locals);
                                env_globals = post_call_env.env_globals }
                (* Update of a global variable *)
                else if NameMap.mem var_name pre_call_env.env_globals then
                        { pre_call_env with env_globals = (update_local_map
pre_call_env.env_globals) }
                (* if the updated object is the calling object *)
                else if (fst pre_call_env.env_context) = (fst post_call_env.env_context)
then
                        { pre_call_env with env_globals = post_call_env.env_globals;
                                env_context = post_call_env.env_context }
                (* if the updated object is a member of the calling object *)
                else
                        let update_implict_map = update_local_map (snd
pre_call_env.env_context) in
                        { pre_call_env with env_globals = post_call_env.env_globals;
                                env_context = ((fst pre_call_env.env_context),
update_implict_map)}
        with Not_found -> raise (Failure ("Did not find the class variable " ^
var_name))


(* ****************** Function Mapping FUNCTIONS ****************** *)



(* return class functions with the format class.function_name *)
let class_func_decls cls_decl =
        List.map (fun fun_decl -> ((cls_decl.class_name ^ "." ^ fun_decl.fname) ,
fun_decl))
        cls_decl.function_members
```

```
(* return a map with all the functions *)
let map_funcs (funcs, classes) =
      let func_decls_first = List.fold_left
            (fun funcs fdecl -> NameMap.add fdecl.fname fdecl funcs)
            NameMap.empty funcs
      in
      let func_decls = List.fold_left
            (fun accum cls_decl -> List.fold_left
                  (fun accum name_fun_pair -> NameMap.add (fst name_fun_pair) (snd
name_fun_pair) accum)
                  accum (class_func_decls cls_decl))
            func_decls_first classes
      in
            func_decls


let find_fdecl (func, func_decls) =
      try NameMap.find func func_decls
      with Not_found -> raise (FunctionNotFoundException ("undefined function " ^
func))

let cls_call_func (call, env, func_decls, func_name, actuals, cls_lit, call_fdecl) =
      let fdecl = find_fdecl (func_name, func_decls) in
      let cls_context = ((get_lhs_varname cls_lit), (get_class_value_map cls_lit))
in
      let call_env = {env with env_locals = NameMap.empty; env_context =
cls_context} in
      let call_lit, post_call_env = call_fdecl (call, fdecl, actuals, call_env) in
      let err_msg = "Unknown context: " ^ (fst call_env.env_context) in
      let return_env = update_env_class_call (env, post_call_env, err_msg) in
      call_lit, return_env
```

8.1.8 geocode.ml

```
open Ast
open Utility

type action = Ast | Ncgenerator | Sast
exception ArgumentFailure of string
exception ParseFailure of string

let read_file file_name =
      let lines = ref [] in
      let channel = open_in file_name in
      try
            while true; do
                  lines := input_line channel :: !lines
            done; !lines
      with End_of_file ->
            close_in channel;
            List.rev !lines ;;

let _ =
      if Array.length Sys.argv == 2 then
            raise (ArgumentFailure("There must be two arguments: mode (-nc,-s, -a)
and file name."))
      else
            let pre_file_name = Sys.argv.(2) in
            let file_name = (pre_file_name ^ ".tmp") in
            let err_file_name = (pre_file_name ^ ".err") in
            let pre_parser_return = Sys.command ("python pre_parser.py " ^
pre_file_name) in
            if Sys.file_exists err_file_name then
                  let lines = read_file err_file_name in
                  List.iter (fun line -> print_endline line) lines
            else
                  if pre_parser_return != 0 then
                        print_endline ("***** Error running python pre_parser.py
on file: " ^ pre_file_name ^
                                                "\nSystem command returned " ^
(string_of_int pre_parser_return))
                  else
                        let action = List.assoc Sys.argv.(1) [ ("-a", Ast); ("-
nc", Ncgenerator); ("-s", Sast) ] in
                        let lexbuf = Lexing.from_channel (open_in file_name) in
                        try
                              let program = Parser.program Scanner.token lexbuf in
                              match action with
                                    Ast -> let listing = string_of_program
program
                                          in print_string listing
                                    | Ncgenerator ->
                                          ignore (Sast.semantic_check program);
                                          ignore (Ncgenerator.run program)
                                    | Sast ->
                                          ignore (Sast.semantic_check program);
                                          let listing = string_of_program program
                                          in print_string listing
                        with
                              | Parsing.Parse_error ->
                                    let token = Lexing.lexeme lexbuf in
                                    print_endline ("Parsing error on token: " ^
token)
                              | Failure s ->
                                    print_endline ("***** Syntax error: " ^ s)
                              | FunctionNotFoundException s ->
                                    print_endline ("***** Syntax error: " ^ s)
                              | Scanner.Syntax_error s ->
                                    print_endline ("***** Syntax error: " ^ s)
                              | ClassConstructorException s ->
                                    print_endline ("***** Syntax error: " ^ s)
                              | SemanticFailure s ->
                                    print_endline ("***** Semantic error: " ^ s)
```

8.1.8 lists.ml

```
open Ast
open Utility
open Environment

(* ****************** List Functions ****************** *)
let get_index index actuals =
      if (List.length actuals) <= index then
            raise (Failure("List function does not have enough parameters"))
      else
            (List.nth actuals index)

let check_paramter_size func_name size actuals =
      if (List.length actuals) != size then
            let err_func_name = "List function " ^ func_name ^ " should have " in
            raise (SemanticFailure(err_func_name ^ (string_of_int size) ^ "
parameters"))
      else
            List.length actuals


let check_list_func (cls_lit, cls_func, actuals, env) =
      let rec get_int_of_literal = function
            IntLiteral(lit)    -> lit
            | LhsLiteral(var, lit) -> get_int_of_literal lit
            | _ -> raise (SemanticFailure("List function parameter must be an
integer")) in
      let items = get_list_items cls_lit in
      (* Can reuse most of the code for push and insert into a list *)
      let list_generic_insert (new_item, index) =
            let new_list = ListLiteral(insert_item (new_item, index) items) in
            let var_name = get_lhs_varname cls_lit in
            let err_msg = "Unable to perform " ^ cls_func ^ " on the list" in
            let ret_val, env = assign_variable_lookup (var_name, new_list, env,
err_msg) in
            new_item, env in
      match cls_func with
            "at" ->
                  let _ = (check_paramter_size cls_func 1 actuals) in
                  let index = (get_int_of_literal (get_index 0 actuals)) in
                  if (index < 0) then
                        raise(SemanticFailure("List At index cannot be negative: "
^ (string_of_int index)))
                  else if (List.length items == 0) then
                        raise(SemanticFailure("Semantic Parsing Error: Cannot
index an empty list"))
                  else
                        let item = List.hd items in
                        item, env
            | "count" ->
                  let _ = (check_paramter_size cls_func 0 actuals) in
                  IntLiteral(List.length items), env
            | "insert" ->
```

```
                    let _ = (check_paramter_size cls_func 2 actuals) in
                    let new_item = (get_index 1 actuals) in
                    let _ = (get_int_of_literal (get_index 0 actuals)) in
                    list_generic_insert (new_item, 0)
             | "pop" ->
                    let _ = (check_paramter_size cls_func 0 actuals) in
                    if (List.length items == 0) then
                           raise(SemanticFailure("Semantic Parsing Error: Cannot pop
an empty list"))
                    else
                           List.hd items, env
             | "push" ->
                    let _ = (check_paramter_size cls_func 1 actuals) in
                    let new_item = (get_index 0 actuals) in
                    let index = List.length items in
                    list_generic_insert (new_item, index)
             | "remove" ->
                    let _ = (check_paramter_size cls_func 1 actuals) in
                    let index = (get_int_of_literal (get_index 0 actuals)) in
                    if (index < 0) then
                           raise(SemanticFailure("List Remove index cannot be
negative: " ^ (string_of_int index)))
                    else if (List.length items == 0) then
                           raise(SemanticFailure("Semantic Parsing Error: Cannot
remove an empty list"))
                    else
                           List.hd items, env
             | "reverse" ->
                    let _ = (check_paramter_size cls_func 0 actuals) in
                    cls_lit, env
             | _ as func -> raise (SemanticFailure("List function not definied: " ^
func))


let eval_list_func (cls_lit, cls_func, actuals, env) =
       let rec get_int_of_literal = function
             IntLiteral(lit)    -> lit
             | LhsLiteral(var, lit) -> get_int_of_literal lit
             | _ -> raise (Failure("List function parameter must be an integer")) in
       let items = get_list_items cls_lit in
       (* Can reuse most of the code for push and insert into a list *)
       let list_generic_insert (new_item, index) =
             let new_list = ListLiteral(insert_item (new_item, index) items) in
             let var_name = get_lhs_varname cls_lit in
             let err_msg = "Unable to perform " ^ cls_func ^ " on the list" in
             let ret_val, env = assign_variable_lookup (var_name, new_list, env,
err_msg) in
             new_item, env in
       match cls_func with
             "at" ->
                    let index = (get_int_of_literal (get_index 0 actuals)) in
                    if ((List.length items) <= index) then
                           raise(Failure("List At index is out of range: " ^
(string_of_int index)))
                    else
```

```
                          let item = List.nth items (get_int_of_literal (get_index 0
actuals))  in
                          item, env
             | "count" ->
                   IntLiteral(List.length items), env
             | "insert" ->
                   let new_item = (get_index 1 actuals) in
                   let index = (get_int_of_literal (get_index 0 actuals)) in
                   list_generic_insert (new_item, index)
             | "pop" ->
                   let (new_list, last_item) = pop_item [] items in
                   let var_name = get_lhs_varname cls_lit in
                   let err_msg = "Unable to pop from list" in
                   let ret_val, env = assign_variable_lookup (var_name,
ListLiteral(new_list), env, err_msg) in
                   last_item, env
             | "push" ->
                   let new_item = (get_index 0 actuals) in
                   let index = List.length items in
                   list_generic_insert (new_item, index)
             | "remove" ->
                   let index = (get_int_of_literal (get_index 0 actuals)) in
                   if ((List.length items) <= index) then
                          raise(Failure("List At index is out of range: " ^
(string_of_int index)))
                   else
                          let removed_item = List.nth items (get_int_of_literal
(get_index 0 actuals))  in
                          let new_list = ListLiteral(remove_item index items) in
                          let var_name = get_lhs_varname cls_lit in
                          let err_msg = "Unable to remove from list" in
                          let ret_val, env = assign_variable_lookup (var_name,
new_list, env, err_msg) in
                          removed_item, env
             | "reverse" ->
                   let new_list = ListLiteral(List.rev items) in
                   let var_name = get_lhs_varname cls_lit in
                   let err_msg = "Unable to reverse list" in
                   assign_variable_lookup (var_name, new_list, env, err_msg)
             | _ as func -> raise (Failure("List function not definied: " ^ func))


let list_concat (lit1, lit2) =
      let items1 = get_list_items lit1 in
      let items2 = get_list_items lit2 in
      ListLiteral(items1 @ items2)
```

8.1.9 ncgenerator.ml

```
open Ast
open Utility
open Class
open Environment
open Lists


let call_fdecl (call, fdecl, actuals, env) =
       try
              let env = { (call fdecl actuals env) with env_locals = NameMap.empty }
              in IntLiteral(0), env
       with ReturnException(v, env) -> v, { env with env_locals = NameMap.empty }

let eval_actual_list (actuals, env, eval) =
       List.fold_left
              (fun (actuals, env) actual -> let v, env = eval env actual in v ::
actuals, env)
              ([], env) (List.rev actuals)


(* Main entry point: run a program *)


(* ****************** Runner/Exec/Eval FUNCTIONS ****************** *)

let run (vars, funcs, classes) =
       (* Put function declarations in a symbol table *)
       let func_decls = map_funcs (funcs, classes)
       in

       (* Invoke a function and return an updated global symbol table *)
       let rec call fdecl actuals call_env =
              (* Evaluate an expression and return (value, updated environment) *)
              let rec eval env = function
                     Literal(i) -> i, env
                     | Noexpr -> IntLiteral(1), env (* must be non-zero for the for
loop predicate *)
                     | Id(var) -> id_variable_lookup (var, env, "ID: undeclared
identifier ")
                     | Cast(v_type, e1) ->
                            let v1, env = eval env e1 in
                            (cast_of_literal (v_type, v1)), env
                     | Negate(e1) ->
                            let v1, env = eval env e1 in
                            (negate_of_literal v1), env
                     | Binop(e1, op, e2) ->
                            let v1, env = eval env e1 in
                            let v2, env = eval env e2 in
                            let bool_compare_exception = (fun x y -> raise
(Failure("Boolean greater/less than operators not supported"))) in
                            (match op with
                                   Add ->
                                          if ((is_list_literal v1) && (is_list_literal
v2)) then
                                                 list_concat (v1, v2)
                                          else
```

```
                                             do_literal_operation (+) (+.) (^)
(v1,v2)
                                 | Sub -> do_literal_operation (-) (-.) (fun x y ->
raise (Failure("String subtract not supported"))) (v1,v2)
                                 | Mult -> do_literal_operation ( * ) ( *. ) (fun x y
-> raise (Failure("String multiply not supported"))) (v1,v2)
                                 | Div -> do_literal_operation ( / ) ( /. ) (fun x y
-> raise (Failure("String divide not supported"))) (v1,v2)
                                 | Equal -> do_literal_compare (==) (=) (=) (==)
(v1,v2)
                                 | Neq -> do_literal_compare (!=) (<>) (<>) (!=)
(v1,v2)
                                 | Less -> do_literal_compare (<) (<) (<)
(bool_compare_exception) (v1,v2)
                                 | Leq -> do_literal_compare (<=) (<=) (<=)
(bool_compare_exception) (v1,v2)
                                 | Greater -> do_literal_compare (>) (>) (>)
(bool_compare_exception) (v1,v2)
                                 | Geq -> do_literal_compare (>=) (>=) (>=)
(bool_compare_exception) (v1,v2)), env
                     | ListItems(items) ->
                           let list_with_env = List.fold_left (fun env_accum item ->
                                 let lit_and_env = get_list_literal item eval (fst
env_accum) in
                                 ((snd lit_and_env), (fst lit_and_env)::(snd
env_accum))) (env, []) items in
                           ListLiteral((snd list_with_env)), (fst list_with_env)
                     | Assign(e_var, e_val) ->
                           let v_var, env = eval env e_var in
                           let v_val, env = eval env e_val in
                           let var_name = (get_lhs_varname v_var) in
                           assign_variable_lookup (var_name, v_val, env, "Assign:
undeclared identifier ")
                     | Call("print", [e]) ->
                           let v, env = eval env e in
                           print_endline (string_of_literal v);
                           IntLiteral(0), env
                     | Call("cos", [e]) ->
                           let v, env = eval env e in
                           DoubleLiteral(cos (double_of_literal v)), env
                     | Call("sin", [e]) ->
                           let v, env = eval env e in
                           DoubleLiteral(sin (double_of_literal v)), env
                     | Call("sqrt", [e]) ->
                           let v, env = eval env e in
                           DoubleLiteral(sqrt (double_of_literal v)), env
                     | ClassCall (e, cls_func, actuals) ->
                           let cls_lit, env = eval env e in
                           let actuals, env = eval_actual_list (actuals, env, eval)
in
                           if (is_list_literal cls_lit) = true then
                                 eval_list_func (cls_lit, cls_func, actuals, env)
                           else
                                 let class_name = (get_class_name cls_lit) in
                                 let func_name = class_name ^ "." ^ cls_func in
```

```
                                cls_call_func (call, env, func_decls, func_name,
actuals, cls_lit, call_fdecl)
                    | Access (e, member) ->
                          let cls_lit, env = eval env e in
                          id_cls_variable_lookup ((get_lhs_varname cls_lit), member,
env, "undeclared access identifier: "), env
                    | Call(func, actuals) ->
                          let get_context_cls_func =
                                if NameMap.cardinal (snd env.env_context) = 0 then
                                      ""
                                else
                                      let cls_name = (get_class_name
(ClassLiteral(snd env.env_context))) in
                                      let cls_func_name = cls_name ^ "." ^ func in
                                      if NameMap.mem cls_func_name func_decls then
                                            cls_func_name
                                      else
                                            "" in
                          let cls_func_name = get_context_cls_func in
                          if String.length cls_func_name > 0 then
                                let cls_lit = LhsLiteral((fst env.env_context),
ClassLiteral(snd env.env_context)) in
                                let actuals, env = eval_actual_list (actuals, env,
eval) in
                                cls_call_func (call, env, func_decls, cls_func_name,
actuals, cls_lit, call_fdecl)
                          else
                                let actuals, env = eval_actual_list (actuals, env,
eval) in

                                try
                                      let fdecl = find_fdecl (func, func_decls) in
                                      let call_env = { env with env_locals =
NameMap.empty; env_context = get_def_env_context } in
                                      let call_lit, post_call_env = call_fdecl
(call, fdecl, actuals, call_env) in
                                      call_lit, {env with env_globals =
post_call_env.env_globals}
                                with FunctionNotFoundException s ->
                                      try
                                            try_func_as_class_constructor (actuals,
env, classes, func, init_var)
                                      with Failure s1 ->
                                            raise (FunctionNotFoundException(s))
              in
              (* Execute a statement and return an updated environment *)
              let rec exec env = function
                    Block(stmts) -> List.fold_left exec env stmts
                    | Expr(e) -> let _, env = eval env e in env
                    | If(e, s1, s2) ->
                          let v, env = eval env e in
                          exec env (if (bool_of_literal v) then s1 else s2)
                    | While(e, s) ->
                          let rec loop env =
                                let v, env = eval env e in
                                if (bool_of_literal v) then loop (exec env s) else
env
                                in loop env
                        | For(e1, e2, e3, s) ->
                                let _, env = eval env e1 in
                                let rec loop env =
                                        let v, env = eval env e2 in
                                        if (bool_of_literal v) then
                                                let _, env = eval (exec env s) e3 in
                                                loop env
                                        else
                                                env
                                in loop env
                        | Return(e) ->
                                let v, env = eval env e in
                                raise (ReturnException(v, env))
                in

                (* Enter the function: bind actual values to formal arguments *)
                let locals =
                        try List.fold_left2
                                (fun locals formal actual -> NameMap.add formal.vname
actual locals)
                                NameMap.empty fdecl.formals actuals
                        with Invalid_argument(_) -> raise (Failure ("wrong number of
arguments passed to " ^ fdecl.fname))
                in
                (* Initialize local variables to 0 *)
                let locals = List.fold_left
                        (fun accum local -> NameMap.add local.vname (init_var classes
local.vtype) accum)
                        locals fdecl.locals
                in
                let locals = NameMap.fold
                        (fun key value accum -> NameMap.add key value accum)
                        call_env.env_locals locals
                in
                (* Execute each statement in sequence, return updated global symbol
table *)
                (List.fold_left exec { call_env with env_locals = locals } fdecl.body)

        (* Run a program: initialize global variables to 0, find and run "main" *)
        in let globals = List.fold_left
                (fun globals vdecl -> NameMap.add vdecl.vname (init_var classes
vdecl.vtype) globals)
                NameMap.empty vars
        in try
                call (NameMap.find "main" func_decls) []
                        { env_globals = globals;
                          env_locals = NameMap.empty;
                          env_context = get_def_env_context }
        with Not_found -> raise (Failure ("did not find the main() function"))
```

8.2 Standard Libraries Files

8.2.1 utility.geo

```
GCodeInit(class Config config):
        print("G21")
        print("T1")
        print("G0 Z" + <string>(config.retract_height))
        print("G21")

class Math:
        Pi():
                return 3.14159265358979323846 26

        ToRad(double degree):
                return ((degree / 180.0) * 3.1415926535897932384626)

        Tolerance():
                return .00000001

class Point:
        double x
        double y

        Magnitude():
                return sqrt((x * x) + (y * y))

        DotProduct(double x1, double y1):
                return (x * x1) + (y * y1)

        Translate(double u, double v):
                x = x + u
                y = y + v

        Rotate(double radians):
                double temp

                temp = (x * cos(radians)) - (y * sin(radians))
                y = (y * cos(radians)) + (x * sin(radians))
                x = temp

        ToIdentity(class LCS lcs):
                class Point end_pt
                double x_offset, y_offset

                x_offset = lcs.axes.u.DotProduct(1.0, 0.0) * x
                x_offset = x_offset + (lcs.axes.v.DotProduct(1.0, 0.0) * y)

                y_offset = lcs.axes.u.DotProduct(0.0, 1.0) * x
                y_offset = y_offset + (lcs.axes.v.DotProduct(0.0, 1.0) * y)

                end_pt = lcs.origin
```

```
            end_pt.Translate(x_offset, y_offset)

            return end_pt

      Print():
            print(x)
            print(y)

class Axes:
      class Point u
      class Point v

      Print():
            u.Print()
            v.Print()

/* local coordinate system */
class LCS:
      class Point origin
      class Axes axes

      Identity():
            origin.x = 0.0
            origin.y = 0.0

            axes.u.x = 1.0
            axes.u.y = 0.0

            axes.v.x = 0.0
            axes.v.y = 1.0

      Translate(double u, double v):
            origin.x = origin.x + u
            origin.y = origin.y + v

      Rotate(double radians):
            axes.u.Rotate(radians)
            axes.v.Rotate(radians)

      Print():
            origin.Print()
            axes.Print()

/* Machining configuration options */
class Config:
      class LCS lcs
      double retract_height
      double cutting_height
      double feed_rate
```

8.2.2 shapes.geo

```
DoubleString(double a):
      class Math math

      if a < math.Tolerance():
            if a > -(math.Tolerance()):
                  return "0."

      return <string>(a)

class Line:
      class Point end
      string desc
      class Config config

      EndPoint():
            return end.ToIdentity(config.lcs)

      Translate(double u, double v):
            config.lcs.Translate(u, v)

      Rotate(double radians):
            config.lcs.Rotate(radians)

      Print():
            string start_string, end_string, feed_rate
            class Point start_pt, end_pt

            start_pt = config.lcs.origin
            end_pt = EndPoint()

            if (desc):
                  print(desc)

            start_string = "X" + DoubleString(start_pt.x) + " Y" +
DoubleString(start_pt.y)
            end_string = "X" + DoubleString(end_pt.x) + " Y" +
DoubleString(end_pt.y)
            feed_rate = " F" + DoubleString(config.feed_rate)

            print("G0 " + start_string)
            print("G1 Z" + DoubleString(config.cutting_height) + feed_rate)
            print("G1 " + end_string + feed_rate)
            print("G0 Z" + DoubleString(config.retract_height))


class Polyline:
      list [class Point] points
      string desc
      class Config config

      AddPoint(class Point pt, class LCS lcs):
            points.push(pt)

      EndPoint():
            class Point end_pt
```

```
        if points:
              end_pt = points.at((points.count() - 1))
              return end_pt.ToIdentity(config.lcs)
        else:
              return config.lcs.origin

  Reverse():
        list [class Point] old_points
        class Point old_start_pt, pt
        double x_change, y_change
        int x

        if points:
              old_start_pt = config.lcs.origin
              config.lcs.origin = points.pop()

              x_change = old_start_pt.x - config.lcs.origin.x
              y_change = old_start_pt.y - config.lcs.origin.y

              old_points = points.reverse()
              points = []

              for (x = 0; x < old_points.count(); x = x + 1):
                    pt = old_points.at(x)
                    pt.Translate(x_change, y_change)
                    points.push(pt)

              pt = old_start_pt
              pt.Translate(x_change, y_change)
              points.push(pt)

  Translate(double u, double v):
        config.lcs.Translate(u, v)

  Rotate(double radians):
        config.lcs.Rotate(radians)

  Append(class Polyline polyline):
        list[class Point] new_pts
        class Point pt
        int x

        new_pts = polyline.points

        for (x = 0; x < new_pts.count(); x = x + 1):
              pt = new_pts.at(x)
              pt = pt.ToIdentity(polyline.config.lcs)
              points.push(pt)

  Print():
        string start_string, end_string, feed_rate
        class Point start_pt, end_pt
        int x
```

```
            start_pt = config.lcs.origin

            if (desc):
                  print(desc)

            start_string = "X" + DoubleString(start_pt.x) + " Y" +
DoubleString(start_pt.y)
            print("G0 " + start_string)

            if points:
                  feed_rate = " F" + DoubleString(config.feed_rate)
                  print("G1 Z" + DoubleString(config.cutting_height) + feed_rate)

            for (x = 0; x < points.count(); x = x + 1):
                  end_pt = points.at(x)
                  end_pt = end_pt.ToIdentity(config.lcs)

                  end_string = "X" + DoubleString(end_pt.x) + " Y" +
DoubleString(end_pt.y)
                  print("G1 " + end_string + feed_rate)

            print("G0 Z" + DoubleString(config.retract_height))

class Circle:
      double radius
      string desc
      class Config config

      Translate(double u, double v):
            config.lcs.Translate(u, v)

      Print():
            string circle_start, feed_rate
            string circle_start_x, circle_start_y

            if (desc):
                  print(desc)

            circle_start_x = DoubleString(config.lcs.origin.x + radius)
            circle_start_y = DoubleString(config.lcs.origin.y)
            circle_start = "X" + circle_start_x + " Y" + circle_start_y
            feed_rate = " F" + DoubleString(config.feed_rate)

            print("G0 " + circle_start)
            print("G1 Z" + DoubleString(config.cutting_height) + feed_rate)
            print("G2 " + circle_start + " I10 J0" + feed_rate)
            print("G0 Z" + DoubleString(config.retract_height))
```

8.2.3 standard.geo

```
include "utility.geo"
include "shapes.geo"
```