iDrive Programming Language

Final Project Report

(for COMS W4115)

Parag Jain

pj2171@columbia.edu

Contents

Chapter 1

Introduction

Cars equipped with a system that is capable of driving from one point to another without input from a human operator have got significant attention from academic, commercial, defense, and government sectors over the past few decades. Early prototypes date as far back as late 1970s, however important progress only came to be seen in the late 2000s through programs like DARPA Grand Challenge competitions and Google Driverless Car project. Some proposed systems depend on infrastructure-based guidance systems; while more advanced systems propose to simulate human perception and decision-making during steering of a car via advanced computer software linked to a range of sensors such as cameras, radar, and GPS. Some of the advantages of "Driverless Cars" include managing traffic flow to increase road capacity, avoid accidents by eliminating driver error, relieving vehicle occupants from driving and navigating chores, transporting loads in dangerous zones such as battlefields, and reducing costs of employing drivers.

Chapter 2

About the `iDrive` Programming Language

`iDrive` is a high level programming language that provides basic constructs to develop algorithms to simulate an environment where cars drive without a need for human intervention. Currently, no such language exists and given the complexity of the paradigm, `iDrive` is extremely useful since it is specialized to accomplish this challenging task. The primary focus of `iDrive` is the abstraction and reduction of non-essential or monotonous tasks so that the researchers no longer need to spend time specifying the characteristics of driving and just need to focus on creating intelligent algorithms for developing fully autonomous cars. `iDrive` provides the researcher with a rich set of tools that may be used intuitively to implement algorithms for driving cars. Every effort has been made to ensure that the language is intuitive and easy to read and understand.

Using `iDrive`, a program first creates a simple car object and subsequently creates other simple objects including cars, traffic signals, stop signs, and pedestrians, and assigns basic properties which characterize them. In the real world, the program would be made aware of the presence of such objects using advanced computer software linked to a range of sensors such as cameras, radar, and GPS. To simulate such input, the program creates these objects randomly and incrementally. As the car heads toward its destination, the program determines the flow of traffic and outputs the outcome of interactions between the objects.

Chapter 3

Language Tutorial

`iDrive` uses a C and Java like syntax. An `iDrive` program consists of global and local object declarations, function declarations, and the `main()` function. `iDrive` supports four simple types - `boolean, integer, decimal,` and `string` - and five complex types – `object, vehicle, pedestrian, trafficsignal,` and `stopsign`. It has support for built-in expressions that operate on one or more of these types. The functions in `iDrive` are similar to C and Java and the `main()` function is the entry point of an `iDrive` program. The following sections elaborate this in detail.

3.1 A First Example

A few variations of a simple "Hello World" program can be used to illustrute how `iDrive` may be used:

First:

```
/* Hello world program */

function main()
{
print("Hello World!!!");
}
```

Second:

```
/* Hello world program */

object a(string x);
```

```
function main()
{
a.x = "Hello World!!!";
print(a.x);
}
```

Third:

```
/* Hello world program */

function main()
{
object a(string x = "Hello John!!!");
a.x = "Hello World!!!";

printSomething();
}

function printSomething()
{
print(a.x);
}
```

3.2 Compiling `iDrive` programs

An `iDrive` program consists of a single source code file. Once this file is created, it must be saved with a .iDrive file extension. Using command line, the following statement compiles an arbitrary `test.iDrive` program and saves the output of the program in a file called `test.out`:

```
iDrive test.iDrive > test.out
```

3.3 More Examples

A rather involved example of an `iDrive` program RepresentativeProgram.iDrive has been included in the appendix. The goal of this program is to safely transport a vehicle object (a car) from its source position to destination position in the presence of other objects like vehicles, pedestrians, traffic signals, and stop signs. Here is a snippet of the output of this program:

```
Current speed is 0 miles/hr
Current position is (0, 0)
Destination position is (0, 50)
Current heading is North
Current distance from destination = 50. miles

Current speed increased to 5. miles/hr
Current position is (0, 0.0277777777778)
Current distance from destination = 49.9722222222 miles
...
DECELERATING DUE TO A STOP SIGN UP AHEAD

Current speed decreased to 40. miles/hr
Current position is (0, 1.47222222222)
Current distance from destination = 48.5277777778 miles

Current speed decreased to 35. miles/hr
Current position is (0, 1.66666666666)
Current distance from destination = 48.3333333333 miles
...
DECELERATING QUICKLY DUE TO AN UNEXPECTED APPROACHING CAR

Current speed decreased to 25. miles/hr
Current position is (0, 6.33333333334)
Current distance from destination = 43.6666666667 miles

Current speed decreased to 5. miles/hr
Current position is (0, 6.36111111112)
Current distance from destination = 43.6388888889 miles
...
DECELERATING DUE TO A TRAFFIC SIGNAL UP AHEAD

Current speed decreased to 15. miles/hr
Current position is (0, 7.27777777779)
Current distance from destination = 42.7222222222 miles

Current speed decreased to 10. miles/hr
Current position is (0, 7.33333333335)
Current distance from destination = 42.6666666667 miles
...
```

```
DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 0. miles/hr
Current position is (0, 12.1111111112)
Current distance from destination = 37.8888888888 miles

Current speed increased to 5. miles/hr
Current position is (0, 12.138888889)
Current distance from destination = 37.861111111 miles
...
Approaching destination

Current speed decreased to 40. miles/hr
Current position is (0, 49.8333333334)
Current distance from destination = 0.1666666666 miles

Current speed decreased to 35. miles/hr
Current position is (0, 50.0277777778)
Current distance from destination = 0.0277777778 miles

Arrived at destination
```

Chapter 4

Language Reference Manual

4.1 Lexical Conventions

`iDrive` uses syntax based on that of C and Java because the idea is to create a development environment that is relatively familiar to those who already know C and Java, hence reducing the learning curve for transitioning to `iDrive`.

`iDrive` uses a standard grammar and character set. Characters in the source code are grouped into tokens, which can be identifiers, keywords, operators, punctuators, or string literals. The compiler forms the longest possible token from a given string of characters; tokens end when white space is encountered, or when it would not be possible for the next character to be part of the token.

4.1.1 Character Set

`iDrive` accepts standard ASCII characters.

4.1.2 Identifiers

An identifier is a sequence of letters, digits, and the underscore character and represents the names of user defined variables and functions. All identifiers start with a letter. Identifiers are case sensitive and keywords can not be used as identifiers.

4.1.3 Keywords

Keywords are identifiers that are reserved words in `iDrive`. They have specific function and can not be used as identifiers. Keywords are case sensitive and valid keywords are:

```
object vehicle pedestrian trafficsignal stopsign
boolean int decimal string
true false
while
for
if else
function
print
random
sqrt
streq
main
```

4.1.4 Constants

A constant is used to set value for an identifier that forms an attribute for an object.

Integer constants

Integer constants are represented with whole numbers in decimal format. An integer constant constitutes only of digits; decimal point and exponent are not allowed. A unary - operator is allowed. For example, 4 or -342.

Floating point constants

Floating point constants are represented with a whole part, a decimal point and a fractional part. The whole part and the fractional part are made up only of digits. A unary - operator is allowed. For example, 8.1 or -0.42322.

String constants

String constants are made up of a sequence of zero or more characters that are enclosed in quotes. For example, “John Smith” or “2”.

4.1.5 Operators

Operators are tokens that specify an operation on at least one operand. They are used in expressions, assignments and object dereferencing.

`< <= > >= == !=` Relational operators
`!` && || Logical operators

`+ - * / ^`	Mathematical operators
`++`	String concatenation operator
`-`	Unary operator
`=`	Assignment operator
`.`	To dereference attribute of an object

The table below shows the precedence the `iDrive` compiler uses to evaluate operators. Operators with the highest precedence appear at the top of the table; those with the lowest precedence appear at the bottom. Operators of equal precedence appear in the same row.

Category	**Operator**	**Associativity**
Dot	`.`	Left to right
Unary	`-`	Right to left
Mathematical	`^`	Left to right
Mathematical	`* /`	Left to right
Mathematical	`+ -`	Left to right
String	`++`	Left to right
Relational	`< <= > >=`	Left to right
Relational	`== !=`	Left to right
Logical	`!`	Left to right
Logical	&&	Left to right
Logical	||	Left to right
Assignment	`=`	Right to left

Associativity relates to precedence, and resolves any ambiguity over the grouping of operators with the same precedence. Most operators associate left-to-right, so the leftmost expressions are evaluated first. The assignment operator and the unary operators associate right-to-left.

4.1.6 Punctuators

`‘ ’` To enclose string constants
`{ }` To enclose a group of statements in a procedure or function
`( )` To enclose a group of arguments for a function
`;` To separate a statement from another
`,` To separate arguments within a function

4.1.7 Comments

Inline comments begin with the // character sequence and end with a line feed. Alternatively, comments may begin with the opening character sequence /* and close with the sequence */. Comments cannot be nested.

4.1.8 White Space

White space characters which include spaces, tabs, and line feed characters may used to separate keywords, operators, and code tokens in the input but are discarded during parsing.

4.1.9 Semicolons and Line Breaks

Semicolons serve as a statement separator, and line breaks serve as a terminator. Multiple statements may be put on a single line of source code using semicolons in between each statement.

4.2 Data Types

4.2.1 Simple Types

These can be defined on their own, or they can serve as attributes within a complex type.

`boolean` Used to hold true/false data
`int` Used to hold integer data
`decimal` Used to hold decimal data
`string` Used to hold string data

4.2.2 Complex Types

`object`

An `object` is used to define a simple object. An `object` is characterized by one or more user defined fields or attributes holding some data. An `object` has no predefined attributes. Attributes are all custom and could be added at initialization as well as later in the program. For example,

```
object identifier (int identifier1 = 1, string identifier2 = ‘‘John Smith’’,
int identifier3 = 5);
```

or

```
object identifier (int identifier1, string identifier2, int identifier3);
identifier.identifier1 = 1;
identifier.identifier2 = ‘‘John Smith’’;
identifier.identifier3 = 5;
```

`vehicle`

A `vehicle` is used to define a car. Similar to `object`, a `vehicle` is characterized by one or more user defined fields or attributes holding some data. At this point `vehicle` has no predefined attributes but such predefined attributes may be introduced in a later version of `iDrive`.

`pedestrian`

A `pedestrian` is used to define a pedestrian. Similar to `object`, a `pedestrian` is characterized by one or more user defined fields or attributes holding some data. At this point `pedestrian` has no predefined attributes but such predefined attributes may be introduced in a later version of `iDrive`.

`trafficsignal`

A `trafficsignal` is used to define a traffic signal. Similar to `object`, a `trafficsignal` is characterized by one or more user defined fields or attributes holding some data. At this point `trafficsignal` has no predefined attributes but such predefined attributes may be introduced in a later version of `iDrive`.

`stopsign`

A `stopsign` is used to define a car stop sign. Similar to `object`, a `stopsign` is characterized by one or more user defined fields or attributes holding some data. At this point `stopsign` has no predefined attributes but such predefined attributes may be introduced in a later version of `iDrive`.

4.3 Statements

Statements are executed in the sequence in which they appear in the code.

4.3.1 Compound Statements

A compound statement, or block, allows a sequence of statements to be treated as a single statement. A compound statement begins with a {, (optionally) contains statements, and ends with a }. For example,

{
statement1
statement2

```
...
}
```

4.3.2 Expression Statements

An expression statement can be a combination of operators, identifiers, and literals. Upon evaluation, an expression returns a value. The value type is dependant on the expressions being combined. For example,

identifier == ‘‘John Smith’’

or

identifier1 > *identifier2*.

4.3.3 Control Statements

To control a program's flow the following control statements are supported much like in C and Java:

`while` loop

The `while` loop has the following syntax:

```
while (expression)
{
statement1
statement2
...
}
```

In a `while` loop, the statements inside the `while` loop are repeatedly executed as long as *expression* is true.

`for` loop

The `for` loop has the following syntax:

```
for (expression1 ; expression2 ; expression3)
{
```

```
statement1
statement2
...
}
```

In a `for` loop, first *expression1* is executed. Then the statements inside the `for` loop are repeatedly executed followed by *expression3* as long as *expression2* is true.

`if else` condition

The `if else` statement has the following syntax:

```
if (expression)
{
statement1
statement2
...
}
else
{
statement3
statement4
...
}
```

The statements following the control expression are executed if the value of the control expression is true (nonzero). The statements in the `else` clause are executed if the control expression is false (zero).

4.4 Functions

4.4.1 User-defined Functions

The `function` keyword is used to define user-defined functions. A function in `iDrive` does not have a return type and all parameters passed to the function are "passed by value" and the outcome of the function is reflected only on the passed arguments. Functions can access global variables, parameters, and locally declared variables. `function` has the following syntax:

```
function identifier (parameter1, parameter2, ...)
```

{
statement1
statement2
...
}

4.4.2 Built-in Functions

`iDrive` provides the following built-in functions which may not be redefined.

`print`

The `print` function is used to output text to the screen. This function takes a combinations of string constants (elcosed in “”) and identifiers (holding some data) as input, resolves the identifiers to text, and outputs the resulting text to the screen. The `print` statement has the following syntax:

`print(‘‘My name is ’’` *identifier* `‘‘. What is your name?’’)`

`random`

The `random` function is used to randomly pick an integer between 0 (inclusive) and ***bound*** (exclusive). This function takes an integer value for ***bound*** as input and outputs a randomly chosen integer. This function is used when the program needs to randomly create objects to interact with the primary vehicle object that is on its way to its destination. The `random` statement has the following syntax:

`random(`*bound*`)`

`sqrt`

The `sqrt` function is used to find the square root of an integer or a decimal. This function takes an integer or decimal value for ***var*** as input and outputs the sqaure root of ***var***. This function is used while computing the distance between a vehicle's current position and destination position. The `sqrt` statement has the following syntax:

`sqrt(`*var*`)`

`streq`

The `streq` function is used to compare two strings. This function takes two strings ***str1*** and ***str2*** as inputs and outputs `true` if the strings are equal and `false` otherwise. The `streq` statement has the following syntax:

`streq(`***str1***`, `***str2***`)`

4.5 Scope

There are two types of scope – local and global. Identifiers declared within a function are local only to that function and may not be used outside the funtion. Global identifiers are declared outside any functions and may be used anywhere in the program. `iDrive` follows the standard rules of static scoping.

Chapter 5

Project Plan

5.1 Responsibilities

There were no formal roles and responsibility assignments done since the project was done by one person.

5.2 Methodology

The `MicroC` project was used as a starting point. So work began by analyzing the design and architecture of the `MicroC` language. Several days were spent brainstorming how this language could be enhanced and transformed into the initial concept of the `iDrive` language. After spending sometime with `MicroC` it was decided to considerably trim down the scope of the `iDrive` language due to time constraints and familiarity with O'Caml. Eventually the project was implemented using an iterative methodology. The idea was to rapidly develop a simple version of `iDrive` using `MicroC` and to iteratively improve it in small testable steps. Each iteration was built upon the work done in previous iterations and added some new features in the language. The steps are shown in more detail in the Project Timeline section. Many small unit tests were created to test the features of `iDrive` – running these gave visibility into what was working and what wasn't. As testing progressed, sometimes implementation details needed to be changed.

5.3 Software Development Environment

`iDrive` was written using Objective Caml Programming Language (`O'Caml`). OCamlLex Lexical Analyzer (`OCamllex`) and OCamlYacc Syntactical Analyzer (`Ocamlyacc`) were used to create the scanner and parser, respectively. The version of `O'Caml` used for development

was 3.11.0.

The project was developed on a Windows XP 32-bit workstation. No source control program was used as this was a one person development effort. A Windows batch file Make.bat was created to compile the source code and another batch file Cleanup.bat was created to perform cleanup chores.

5.4 Programming Style

Since the `MicroC` project was utilized to build `iDrive`, the programming style of `MicroC` was used.

5.5 Project Timeline

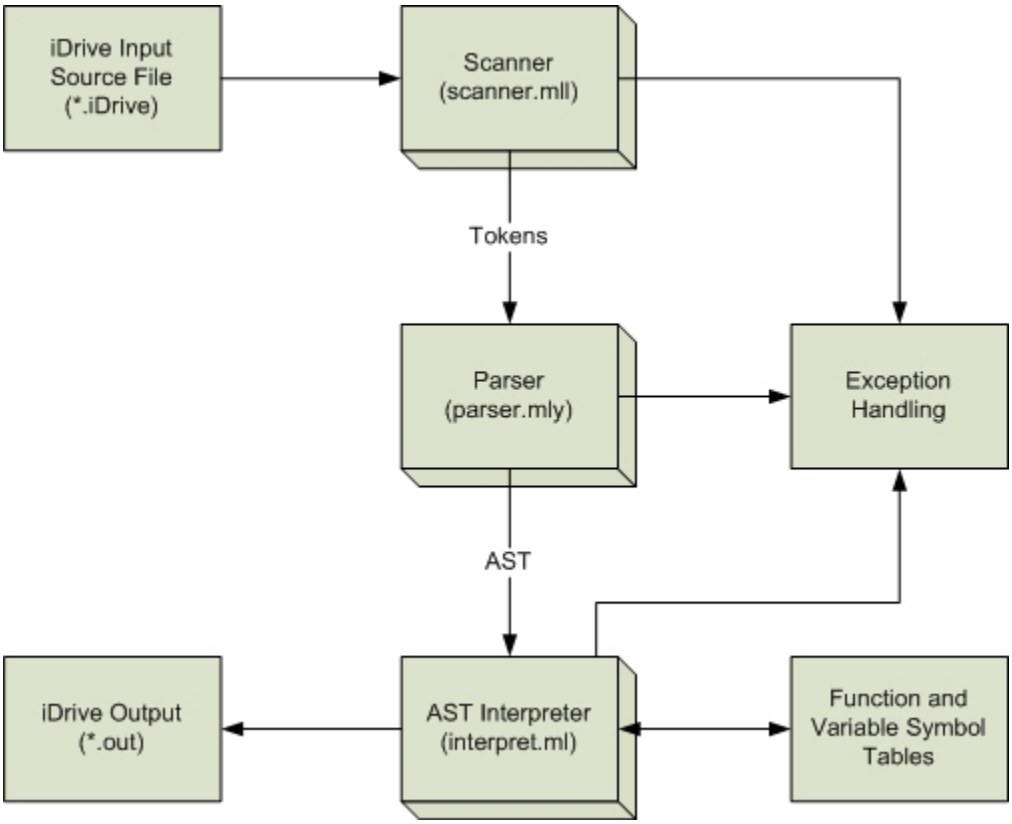
Here is the project timeline:

1/24-2/9	Started brainstorm language ideas
2/5	`iDrive` conceived
2/9	Project proposal submitted
2/9 - 2/28	Familiarize with `MicroC`
2/28 - 3/14	Compiled first version of `iDrive` with little to no customizations
3/21	Language Reference Manual submitted
3/21 - 3/28	Parsing and lexing of basic types
3/28 - 4/11	Parsing and lexing of complex types
4/11 - 4/18	Parsing and lexing of statements and expressions
4/18 - 4/20	Control statements
4/20 - 4/25	Built-in functions added
4/25 - 5/8	Logical Operators and several miscellaneous features added
3/21 - 5/19	Build and run test cases and debug issues
4/11 - 5/19	Final project report
5/16 - 5/19	Final testing, documentation, and final report

Chapter 6

Architectural Design

6.1 Architectural Block Diagram



6.2 Interfaces and Flow

As shown in the block diagram above, `iDrive` consists of several components such as the scanner, the parser, the abstract syntax tree (AST), the AST interpreter, and the symbol table. The relationship between these components is depicted in the diagram.

Essentially, the scanner receives the input source file and performs lexical analysis of the input file. The lexical analyzer separates the input character stream and produces a stream of tokens. Irrelevant details such as whitespace and comments are removed at this stage. If any invalid token is captured, an exception is raised.

The parser receives the token stream from the scanner, parses these tokens, and analyzes the structure of the program. It checks whether the program conforms to the grammar of `iDrive` and generates an abstract syntax tree (AST) for the given source program. If any syntax error is captured, a parsing exception is raised.

The AST interpreter traverses through the AST and creates the symbol tables for funtion and variable declarations. These symbol tables are are checked in order to resolve functions and variables, check data types, and evaluate expressions. For any invalid arguments, it throws appropriate exceptions.

6.3 Enhancements over `MicroC`

The scanner, the parser, and the AST interpreter were substantially enhanced from `MicroC` to support the requirements of the `iDrive` language. Support was added in these components for:

- Additional simple types like boolean, decimal, and string
- Complex types like objects, vehicles, pedestrians, traffic signals, and stop signs
- Mathematical operators like exponentiation
- String concatenation
- Logical operators
- User-defined functions
- Several built-in functions to support printing, string comparision, random number generation, and square root operation

Most importantly, a new type of expression rule was introduced in the parser to specially handle the newly introduced objects and their attributes. Lastly, to minimize the complexity of the language due to introduction of new simple types, all object attributes are internally stored in string format. The attributes are converted to decimal format when mathematical operations are performed on them.

Chapter 7

Test Plan

A test plan was created to reflect the aspects of the program that were covered in the Language Reference Manual. Most of the test cases were generated while developing the various features of `iDrive`. Creating tests in this manner allowed to create a test suite that was comprehensive and at the same time ensured the clarity of the manual. While most tests were geared towards validating that a certain piece of funtionality worked, certain tests were designed to fail to make sure that various parts of the system failed properly. Still other tests were composite tests and were designed to test multiple features at once. The test cases used are listed below.

7.1 Test Cases

File Name	**Purpose of the Test**
test1.iDrive	Hello world program
test2.iDrive	To test comments
test3.iDrive	To test identifiers: declare a global variable, define two attributes on it with the same key but different case, and print them
test4.iDrive	To test global variables: declare a global variable, print it, update it, and print it again
test5.iDrive	To test local variables: declare a local variable, print it, update it, and print it again
test6.iDrive	To test assignment of constants to variable attributes: declare a global variable with four simple attributes and assign constant values to the attributes in all possible forms
test7.iDrive	To test default initialization of variable attributes: declare a global variable with four attributes without initializing them,

	print them, update them, and print them again
test8.iDrive	To test custom initialization of variable attributes: declare a global variable with four initialized attributes, print them, update them, and print them again
test9.iDrive	To test the unary '-' operator: declare a global variable with two integer attribbutes, print them, perform unary operations, and print them again
test10.iDrive	To test mathematical operators: declare a global variable with an integer attribute, perform mathematical operations on it using mathematical operators '+', '-', '*', '/', exponentiation operator '^', and the built-in sqrt funtion, and print the results of the operations. This is done twice, first with a positive integer and then a negative integer
test11.iDrive	To test mathematical operators: declare a global variable with a decimal attribute, perform mathematical operations on it using mathematical operators '+', '-', '*', '/', exponentiation operator '^', and the built-in sqrt funtion, and print the results of the operations. This is done twice, first with a positive decimal and then a negative decimal
test12.iDrive	To test mathematical operator precedence: declare a global variable, perform mathematical operations to test operator precedence using mathematical operators '+', '-', '*', '/', exponentiation operator '^', and parentheses, and print the results of the operations
test13.iDrive	To test string concatenation: declare a global variable and perform string concatenation and addition operations using the concatenation operator '++' and the addition operator '+', and print the results of the operations
test14.iDrive	To test the built-in random function: declare a global variable, invoke the built-in random function, and print the results
test15.iDrive	To test the boolean data type: declare a global variable with a boolean attribute, test the boolean attribute in several ways using if statements, and print the results
test16.iDrive	To test relational operators: declare a global variable with an integer attribute, test the relational operators `< <= > >= == !=` in several ways using if statements, and print the results of the operations. This is done twice, first with a positive integer and then a negative integer
test17.iDrive	To test relational operators:

	declare a global variable with a decimal attribute, test the relational operators `< <= > >= == !=` in several ways using if statements, and print the results of the operations. This is done twice, first with a positive decimal and then a negative decimal
test18.iDrive	To test the built-in streq funtion: declare a global variable with a string attribute, test the built-in streq funtion using if statements, and print the results of the operations
test19.iDrive	To test logical operators: declare a global variable with a string and an integer attribute, test the logical operators '&&', '||', '!' in several ways using if statements, and print the results of the operations
test20.iDrive	To test if else statements: declare a global variable, test the if else statement in various forms including nested if else statements and block of statements, and print the results
test21.iDrive	To test for loop: declare a global variable, run a for loop, and print the results
test22.iDrive	To test while loop: declare a global variable, run a while loop, and print the results
test23.iDrive	To test nested for loops and while loops: declare a global variable, run all combinations of nested for loops and while loops, and print the results
test24.iDrive	To test multiple global variables: declare global variables in various forms, print them, update them, and print them again
test25.iDrive	To test user-defined functions: declare global variables, define a function printObject() that prints attributes of a variable, and invoke this function to print the global variable
test26.iDrive	To test user-defined functions on multiple inputs: declare two global variables, define a function printObject() that prints attributes of a variable, update the global variables, and invoke the printObject() function once for each global variable to print them
test27.iDrive	To test static scoping of variables: declare a global variable and a local variable with the same name, update the local variable, and print both the variables
test28.iDrive	To test pass by value: declare a local variable, pass the local variable to another funtion mutateObject() (pass by value), update the variable in that function, and print the variable just before the funtion returns to show the altered state of the variable and just after the function returns to show the

	unaltered state of the local variable
test29.iDrive	To further test scope of a global variable: declare a global variable, invoke another funtion mutateGlobalObject(), update the global variable in that function, and print the variable just before the funtion returns to show the altered state of the variable and just after the function returns to again show the altered state of the variable
test30.iDrive	To test miscellaneous object operations: declare a global variable and a local variable, invoke a funtion createDuplicateAndRandomlyMutate() to create a duplicate of the local variable, and perform miscellaneous updates to this variable

7.2 Automation

A Windows batch file RunTests.bat was created to automatically run all the test cases. This batch file ran the `iDrive` interpreter on all `iDrive` input source files, saved the output of each test as an output file, and compared it to the expected output file corresponding to that test. Only if the two files matched, was the test was deemed successful. Another batch file CleanupTests.bat was used to perform cleanup of the test output files generated after each regression.

Since these tests only took a few seconds to run, for sometime they were run after each source code compilation. This provided a reasonable assurance that the new code did not break any existing and working features.

7.3 Testing Results

The following test results were generated after each run of the test suite:

```
>RunTests.bat

test1 passed
test10 passed
test11 passed
test12 passed
test13 passed
```

```
test14 failed
test15 passed
test16 passed
test17 passed
test18 passed
test19 passed
test2 passed
test20 passed
test21 passed
test22 passed
test23 passed
test24 passed
test25 passed
test26 passed
test27 passed
test28 passed
test29 passed
test3 passed
test30 passed
test4 passed
test5 passed
test6 passed
test7 passed
test8 passed
test9 passed
```

As expected, `test14.iDrive` failed almost all the time because it tested the built-in `random` function in `iDrive`, which randomly generated an integer.

Chapter 8

Lessons Learned

My initial project proposal was a little too ambitious especially given the time frame, my familiarity with O'caml, a full-time job, and a single resource working on the project. I believe it was beneficial for me to realize this early on and accordingly change the course of action. I cut down quite a bit from the original scope of the language by eliminating almost all the built-in functions that I had initially proposed. In the end I implemented a lighter version of `iDrive` instead of the full-fledged "Java-like" `iDrive` without compromising much functionality and ease of use. The language is now a simple object-oriented language that focuses on simulating an environment where cars drive without a need for human intervention in the presence of other objects like cars, pedestrians, traffic signals, and stop signs.

Despite the fact that I realized early how challenging designing and developing a language could be using O'Caml, I still was unable to get the AST to bytecode compiler to work. I think too much time was spent on understanding O'Caml (and eliminating misunerstandings of the language) and getting the AST interpreter to work even with the slimmed down version of `iDrive`. The smallest feature of the language sometimes took almost an entire day to implement. If it were not for the `MicroC` example and the iterative approach of development that I implemented I do not think I could have developed `iDrive` in the given time. The O'Caml learning curve is just too steep, hence this project work is perhaps best suited for a team of 2-3 individuals. I have in the past collaborated with other CVN students to work on course projects and I feel it would have been beneficial to work as part of a team and having some divison of labor given the extensiveness of the project.

I was not able to master O'Caml like any other new language that I have picked up in the past, nonetheless, I learned that O'Caml is a great language when it comes to creating programming languages because of its strong data type system and the way its functions are created and used. I also learned how simple each layer of indirection within an interpreter can be and how to logically reason and resolve the shift/reduce and reduce/reduce conflicts.

The concepts taught in class guided me through every layer in the project.

Creating a test suite early on proved to be extremely helpful. The automated regression tests were invaluable in finding bugs. I found it best to write test cases for new features immediately after the features had been written. This process of adding features and immediately testing provided more assurance that the interpreter is working within expectations.

My advice to others is to get a simple version of the lanaguage working first before delving into complex details. It was very beneficial for me to develop the basic features of the language first. After which it became clear how to expand the language and build up the complexity iteratively. Additionally, I would highly recommend others to get familiar with the syntax of O'Caml very early in the design. I would also advise others to work in a team – I feel this may be the ultimate key to success of this project.

Overall I learned a lot about the inner workings of compilers which I was not aware of earlier. I also learned about the many considerations that go into designing a programming language such as scope, syntax, and structure. Furthermore I enjoyed learning O'Caml – a new and fundamentally very different type of language. I gratefully acknowledge the valuable suggestions and insights provided by the course professor that helped me work on this project.

Chapter 9

Appendix

9.1 scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }  (* White space characters *)
  | "/*"      { comment lexbuf }            (* Multiline Comments *)
  | "//"      { linecomment lexbuf }        (* Inline Comments *)
  | '('       { LPAREN } (* Left parenthesis *)
  | ')'       { RPAREN } (* Right parenthesis *)
  | '{'       { LBRACE } (* Left brace *)
  | '}'       { RBRACE } (* Left brace *)
  | ';'       { SEMI } (* Semicolon, to separate statements *)
  | ','       { COMMA } (* Comma, to separate arguments within a function *)
  | '+'       { PLUS } (* Addition mathematical operator *)
  | '-'       { MINUS } (* Subtraction mathematical operator *)
  | '*'       { TIMES } (* Multiplication mathematical operator *)
  | '/'       { DIVIDE } (* Division mathematical operator *)
  | '^'       { POW } (* Exponentiation mathematical operator *)
  | '='       { ASSIGN } (* Assignment operator *)
  | '.'       { DEREFERENCE } (* Dereference an attribute of an object *)
  | "=="      { EQ } (* Equality relational operator *)
  | "!="      { NEQ } (* Inequality relational operator *)
  | '<'       { LT } (* Less than relational operator *)
  | "<="      { LEQ } (* Less than or equal to relational operator *)
  | '>'       { GT } (* Greater than relational operator *)
```

```
| ">="      { GEQ } (* Greater than or equal to relational operator *)
| "!"     { NOT } (* Not logical operator *)
| "&&"     { AND } (* And logical operator *)
| "||"     { OR } (* Or logical operator *)
| "++"      { CONCAT } (* String concatenation operator *)
| "if"      { IF } (* If statement *)
| "else"    { ELSE } (* Part of if statement *)
| "for"     { FOR } (* For loop *)
| "while"   { WHILE } (* While Loop *)
| "function" { FUNCTION } (* To declare user-defined funtions *)
| "vehicle"  { VEHICLE } (* Vehicle object *)
| "pedestrian"   { PEDESTRIAN } (* Pedestrian object *)
| "trafficsignal"   { TRAFFICSIGNAL } (* Traffic Signal object *)
| "stopsign"    { STOPSIGN } (* Stop sign object *)
| "object"  { OBJECT } (* Simple object *)
| "boolean"  { BOOLEAN } (* Boolean data type *)
| "int"  { INT } (* Interger data type *)
| "decimal"  { DECIMAL } (* Float data type *)
| "string"    { STRING } (* String data type *)
| "print"   { PRINT } (* Built-in print function *)
| "random"  { RANDOM } (* Built-in random function *)
| "sqrt"    { SQRT } (* Built-in sqrt function *)
| "streq"    { STREQ } (* Built-in streq function *)

| "true" as lxm { LITERAL(lxm) }   (* Boolean *)

| "false" as lxm { LITERAL(lxm) }   (* Boolean *)

| ['0'-'9']+  as lxm { LITERAL(lxm) }   (* Integers *)

| ['0'-'9']+ '.' ['0'-'9']* ('e' ('+'|'-')? ['0'-'9']+)?
as lxm { LITERAL(lxm) }   (* Floats *)

|           '.' ['0'-'9']+ ('e' ('+'|'-')? ['0'-'9']+)?
as lxm { LITERAL(lxm) }   (* Floats *)

| ['0'-'9']+                 'e' ('+'|'-')? ['0'-'9']+
as lxm { LITERAL(lxm) }   (* Floats *)

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

```
   as lxm { ID(lxm) }   (* Identifiers *)

  | '\"' [^ '\"']* '\"'
   as lxm { STR(lxm) }   (* Strings *)

  | eof  { EOF }

  | _ as char  { raise (Failure("illegal character " ^ Char.escaped char)) }

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

and linecomment = parse
  ['\r' '\n']  { token lexbuf }
  | _     { linecomment lexbuf }
```

9.2 parser.mly

```
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE POW SQRT ASSIGN
%token EQ NEQ STREQ LT LEQ GT GEQ CONCAT
%token NOT AND OR
%token IF ELSE FOR WHILE
%token BOOLEAN INT DECIMAL STRING
%token <string> LITERAL
%token <string> ID
%token <string> STR
%token EOF
%token FUNCTION VEHICLE PEDESTRIAN TRAFFICSIGNAL STOPSIGN OBJECT
%token DEREFERENCE
%token PRINT RANDOM

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc RANDOM
```

```
%left ASSIGN
%left NOT AND OR
%left EQ NEQ STREQ
%left LT GT LEQ GEQ
%left CONCAT
%left PLUS MINUS
%left TIMES DIVIDE
%left POW SQRT
%nonassoc NEG

%start program
%type <Ast.program> program

%%

program:
  /* nothing */         { [], [] }
  | program vdecl       { ($2 :: fst $1), snd $1 }
  | program fdecl       { fst $1, ($2 :: snd $1) }

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

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

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

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

vdecl:
```

```
  OBJECT ID LPAREN attr_opt RPAREN SEMI  { { vtype = "Object";
       vname = $2; vattrs = $4;} }
  | VEHICLE ID LPAREN attr_opt RPAREN SEMI  { { vtype = "Vehicle";
       vname = $2; vattrs = $4;} }
  | PEDESTRIAN ID LPAREN attr_opt RPAREN SEMI  { { vtype = "Pedestrian";
       vname = $2; vattrs = $4;} }
  | TRAFFICSIGNAL ID LPAREN attr_opt RPAREN SEMI  { { vtype = "TrafficSignal";
       vname = $2; vattrs = $4;} }
  | STOPSIGN ID LPAREN attr_opt RPAREN SEMI  { { vtype = "StopSign";
       vname = $2; vattrs = $4;} }

attr_opt:
  /* nothing */                  { [] }
  | attr_list                       { $1 }

attr_list:
  attr                           { [$1] }
  | attr_list COMMA attr            { $3 :: $1 }

attr:
  STRING ID ASSIGN STR     { { key = $2;
       value = String.sub $4 1 ((String.length $4)-2);} }

  | STRING ID      { { key = $2; value = "";} }

  | DECIMAL ID ASSIGN MINUS LITERAL { { key = $2;
       value = string_of_float (-.(float_of_string $5));} }

  | DECIMAL ID ASSIGN LITERAL     { { key = $2; value = $4;} }

  | DECIMAL ID      { { key = $2; value = "0.0";} }

  | INT ID ASSIGN MINUS LITERAL { { key = $2;
       value = string_of_float (-.(float_of_string $5));} }

  | INT ID ASSIGN LITERAL     { { key = $2; value = $4;} }

  | INT ID      { { key = $2; value = "0";} }

  | BOOLEAN ID ASSIGN LITERAL     { { key = $2; value = $4;} }
```

```
  | BOOLEAN ID      { { key = $2; value = "true";} }

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

stmt:
  expr SEMI  { Expr($1) }

  | LBRACE stmt_list RBRACE  { Block(List.rev $2) }

  | IF LPAREN objexpr RPAREN stmt %prec NOELSE
   { If($3, $5, Block([])) }

  | IF LPAREN objexpr RPAREN stmt ELSE stmt
   { If($3, $5, $7) }

  | FOR LPAREN objexpr_opt SEMI objexpr_opt SEMI objexpr_opt RPAREN stmt
   { For($3, $5, $7, $9) }

  | WHILE LPAREN objexpr RPAREN stmt  { While($3, $5) }

  | PRINT objexpr SEMI       { Print($2) }

expr:
  ID                                        { Id($1) }
  | ID DEREFERENCE ID ASSIGN objexpr      { Assign($1, $3, $5) }
  | ID LPAREN actuals_opt RPAREN            { Call($1, $3) }
  | LPAREN expr RPAREN                      { $2 }

objexpr_opt:
  /* nothing */  { Noexpr }
  | objexpr           { $1 }

objexpr:
  LITERAL                                     { Literal($1) }
  | STR                                       { Str($1) }
  | MINUS objexpr %prec NEG         { Neg($2) }
  | objexpr PLUS objexpr                { Binop($1, Add, $3) }
```

```
  | objexpr MINUS objexpr               { Binop($1, Sub, $3) }
  | objexpr TIMES objexpr               { Binop($1, Mult, $3) }
  | objexpr DIVIDE objexpr              { Binop($1, Div, $3) }
  | objexpr POW objexpr                 { Binop($1, Pow, $3) }
  | objexpr EQ objexpr                  { Binop($1, Equal, $3) }
  | objexpr NEQ objexpr                 { Binop($1, Neq, $3) }
  | objexpr LT objexpr                  { Binop($1, Less, $3) }
  | objexpr LEQ objexpr                 { Binop($1, Leq, $3) }
  | objexpr GT objexpr                  { Binop($1, Greater, $3) }
  | objexpr GEQ objexpr                 { Binop($1, Geq, $3) }
  | objexpr CONCAT objexpr              { Binop($1, Concat,  $3) }
  | objexpr AND objexpr         { Binop($1, And, $3) }
  | objexpr OR objexpr         { Binop($1, Or, $3) }
  | STREQ LPAREN objexpr COMMA objexpr RPAREN
   { Streq($3, $5) }
  | NOT objexpr                { Not($2) }
  | SQRT objexpr                     { Sqrt($2) }
  | RANDOM objexpr                     { Random($2) }
  | ID DEREFERENCE ID ASSIGN objexpr        { ObjAssign($1, $3, $5) }
  | ID DEREFERENCE ID                           { Attribute($1, $3) }
  | LPAREN objexpr RPAREN                   { $2 }

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

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

9.3 ast.mli

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

(* object expressions *)
type objexpr =
  Literal of string
  | Noexpr
```

```
  | Str of string
  | Neg of objexpr
  | Binop of objexpr * op * objexpr
  | Streq of objexpr * objexpr
  | Not of objexpr
  | Sqrt of objexpr
  | Random of objexpr
  | ObjAssign of string * string * objexpr
  | Attribute of string * string

(* expressions *)
type expr =
  Id of string
  | Assign of string * string * objexpr
  | Call of string * expr list

(* statements *)
type stmt =
  Expr of expr
  | Block of stmt list
  | If of objexpr * stmt * stmt
  | For of objexpr * objexpr * objexpr * stmt
  | While of objexpr * stmt
  | Print of objexpr

(* variable attribute declarations *)
type attr_decl = {
  key   : string;          (* attribute key *)
  value : string;                   (* attribute value *)
}

(* variable declarations *)
type var_decl = {
  vtype  : string;           (* variable type *)
  vname  : string;                  (* variable name *)
  vattrs : attr_decl list;          (* variable attributes *)
}

(* function declarations *)
type func_decl = {
```

```
    fname   : string;           (* function name *)
    formals : string list;          (* function arguments *)
    locals  : var_decl list;          (* list of local variables *)
    body    : stmt list;          (* list of statements *)
}

(* the program *)
type program = var_decl list * func_decl list
```

9.4 interpret.ml

```
open Ast

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

exception ReturnException of string NameMap.t * string NameMap.t NameMap.t

(* Main entry point: run a program *)

let run (vars, funcs) =

  (* Put function declarations in a symbol table *)

  let func_decls = List.fold_left
    (fun funcs fdecl ->
      if NameMap.mem fdecl.fname funcs then
        raise (Failure ("function " ^ fdecl.fname ^
         " is defined more than once!"));
        NameMap.add fdecl.fname fdecl funcs )
    NameMap.empty funcs

  in

  (* Put variable declarations in a symbol table *)

  let var_decls = List.fold_left
```

```
  (fun globals vdecl ->
    let attrs = List.fold_left
      (fun attr_map attr_decl ->
      NameMap.add attr_decl.key attr_decl.value attr_map)
      NameMap.empty vdecl.vattrs in
        if NameMap.mem vdecl.vname globals then
          raise (Failure ("variable " ^ vdecl.vname ^
           " is defined more than once!"));
          NameMap.add vdecl.vname attrs globals;)

in

(* Invoke a function and return an updated global symbol table *)

let rec call fdecl actuals globals =

  (* Evaluate an object expression and return (value, updated environment) *)

  let rec objeval env = function

    Literal(i) -> i, env

    | Noexpr -> "1", env  (* must be a string;
        must be non-zero for the for loop predicate *)

    | Str(str) -> (String.sub str 1 (( String.length str)-2)), env

    | Neg(e) ->
let v, env = objeval env e in
  string_of_float(-.(float_of_string(v))), env

    | Binop(e1, op, e2) ->
        let v1, env = objeval env e1 in
        let v2, env = objeval env e2 in
        let boolean_to_string i = if i then "true" else "false" in
          (match op with

            Add -> string_of_float(float_of_string(v1) +. float_of_string(v2))

            | Sub -> string_of_float(float_of_string(v1) -. float_of_string(v2))
```

```
            | Mult -> string_of_float(float_of_string(v1) *. float_of_string(v2))

            | Div -> string_of_float(float_of_string(v1) /. float_of_string(v2))

            | Pow -> string_of_float(float_of_string(v1) ** float_of_string(v2))

            | Equal -> boolean_to_string(float_of_string(v1) = float_of_string(v2))

            | Neq -> boolean_to_string(float_of_string(v1) <> float_of_string(v2))

            | Less -> boolean_to_string(float_of_string(v1) < float_of_string(v2))

            | Leq -> boolean_to_string(float_of_string(v1) <= float_of_string(v2))

            | Greater -> boolean_to_string(float_of_string(v1) > float_of_string(v2))

            | Geq -> boolean_to_string(float_of_string(v1) >= float_of_string(v2))

            | Concat -> String.concat "" (v1::(v2::[]))

            | And -> if (0 = (String.compare v1 "true")
             && 0 = (String.compare v2 "true")) then "true" else "false"

            | Or -> if (0 = (String.compare v1 "true")
             || 0 = (String.compare v2 "true")) then "true" else "false"), env

    | Streq(e1, e2) ->
        let v1, env = objeval env e1 in
        let v2, env = objeval env e2 in
let int_to_string i = if (i = 0) then "true" else "false" in
  int_to_string (String.compare v1 v2), env

    | Not(e) ->
let v, env = objeval env e in
  let not_e i = if (0 = (String.compare i "true")) then "false" else "true" in
  not_e (v), env

    | Sqrt(e) ->
let v, env = objeval env e in
```

```
  string_of_float(sqrt(float_of_string(v))), env

    | Random(e) ->
let v, env = objeval env e in
  string_of_int(Random.self_init (); Random.int (int_of_string(v))), env

    | ObjAssign(var, attr, e) ->
        let v, (locals, globals) = objeval env e in
          if NameMap.mem var locals then
            v, (NameMap.add var
              (NameMap.add attr v (NameMap.find var locals) ) locals, globals)
          else if NameMap.mem var globals then
            v, (locals, NameMap.add var
              (NameMap.add attr v (NameMap.find var globals) ) globals)
          else raise (Failure ("undeclared identifier " ^ var))

    | Attribute(var, attr) ->
        let (locals, globals) = env in
          if NameMap.mem var locals then
            if NameMap.mem attr (NameMap.find var locals) then
              NameMap.find attr (NameMap.find var locals),(locals, globals)
            else raise (Failure ("undeclared attribute " ^ attr ^
             " for local variable " ^ var))
          else if NameMap.mem var globals then
            if NameMap.mem attr (NameMap.find var globals) then
              NameMap.find attr (NameMap.find var globals),(locals, globals)
            else raise (Failure ("undeclared attribute " ^ attr ^
             " for global variable " ^ var))
          else raise (Failure ("undeclared identifier " ^ var))

  in

  (* Evaluate an expression and return (value, updated environment) *)

  let rec eval env = function

      Id(var) ->
        let locals, globals = env in
          if NameMap.mem var locals then
           (NameMap.find var locals), env
```

```
        else if NameMap.mem var globals then
          (NameMap.find var globals), env
        else raise (Failure ("undeclared identifier " ^ var))

    | Assign(var, attr, e) ->
        let v, (locals, globals) = objeval env e in
          if NameMap.mem var locals then
            NameMap.empty, (NameMap.add var (NameMap.add attr v
                    (NameMap.find var locals) ) locals, globals)
          else if NameMap.mem var globals then
            NameMap.empty, (locals, NameMap.add var
                    (NameMap.add attr v (NameMap.find var globals) ) globals)
          else raise (Failure ("undeclared identifier " ^ var))

    | Call(f, actuals) ->
        let fdecl =
          try NameMap.find f func_decls
          with Not_found -> raise (Failure ("undefined function " ^ f))

        in

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

        in

        let (locals, globals) = env in

          try

            let globals = call fdecl actuals globals in
              NameMap.empty, (locals, globals)

          with ReturnException(v, globals) -> v, (locals, globals)

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 = objeval env e in
          exec env (if (0 = (String.compare v "true")) then s1 else s2)

    | While(e, s) ->
        let rec loop env =
    let v, env = objeval env e in
    if (0 = (String.compare v "true")) then loop (exec env s) else env
in loop env

    | For(e1, e2, e3, s) ->
        let _, env = objeval env e1 in
          let rec loop env =
            let v, env = objeval env e2 in
              if (0 = (String.compare v "true")) then
                let _, env = objeval (exec env s) e3 in
                loop env
              else
                env
          in loop env

    | Print(e) ->
        let str, env = objeval env e in
          print_endline str; env

  in

  (* Enter the function: bind actual values to formal arguments *)

  let locals =
    try List.fold_left2
      (fun locals formal actual -> NameMap.add formal actual locals)
      NameMap.empty fdecl.formals actuals
```

```
      with Invalid_argument(_) ->
        raise (Failure ("wrong number of arguments passed to " ^ fdecl.fname))

    in

    (* Add local variables to the symbol table *)

    let locals = var_decls locals fdecl.locals in

    (* Execute each statement in sequence, return updated global symbol table *)

    snd (List.fold_left exec (locals, globals) fdecl.body)

  in

  (* Run a program: add global variables to the symbol table, find and run "main"  *)

  let globals = var_decls NameMap.empty vars in

  try
    call (NameMap.find "main" func_decls) [] globals
  with Not_found -> raise (Failure ("did not find the main() function"))
```

9.5 iDrive.ml

```
(* read the input source file, lex it, parse it, and intepret it *)

let _ =

  let lexbuf = Lexing.from_channel (open_in Sys.argv.(1)) in

  let program = Parser.program Scanner.token lexbuf in

    ignore (Interpret.run program)
```

9.6 Make.bat

```
@echo off
```

```
rem create scanner.ml
ocamllex scanner.mll

rem create parser.ml and parser.mli
ocamlyacc parser.mly

rem compile AST types
ocamlc -c ast.mli

rem compile parser types
ocamlc -c parser.mli

rem compile the scanner
ocamlc -c scanner.ml

rem compile the parser
ocamlc -c parser.ml

rem compile the interpreter
ocamlc -c interpret.ml

rem compile iDrive
ocamlc -c iDrive.ml

rem package the executable
ocamlc -o iDrive.exe parser.cmo scanner.cmo interpret.cmo iDrive.cmo

echo.
```

9.7 Cleanup.bat

```
@echo off

rem remove compiled output files
del scanner.ml
del parser.mli
del parser.ml
del *.cmo
```

```
del *.cmi
del iDrive.exe
```

9.8 RunTests.bat

```
@echo off

rem iDrive test1.iDrive > test1.out

rem fc test1.out test1.expectedout > test1.diff

rem loop over all iDrive files in the current folder
for %%X in (*.iDrive) do (

rem compile the file and send output to .out file
        iDrive %%~nX.iDrive > %%~nX.out

rem compare the .out file with .expectedout file
fc %%~nX.out %%~nX.expectedout > %%~nX.diff

rem in case of mismatch show failure message
IF ERRORLEVEL 1 echo %%~nX failed

rem in case of match show success message
IF ERRORLEVEL 0 echo %%~nX passed

)
```

9.9 CleanupTests.bat

```
@echo off

rem remove testing output
del *.out
del *.diff
```

9.10 Representative Program

Here is a representative program that demonstrates a simple `iDrive` program, which first creates a simple `vehicle` object and subsequently creates other objects. As the car heads from its source position to its destination position, the program determines the flow of traffic and outputs the outcome of interactions between these objects:

```
/* This is the vehicle object that needs to be transported from
source position to destination position */

vehicle myCar(decimal currentXPosition=0, decimal currentYPosition=0,
      decimal currentSpeed=0, string heading=‘‘North’’,
      decimal destXPosition=0, decimal destYPosition=50,
      decimal distanceToDest, boolean isClearToGo = true);

/* This is an object that stores global variables */

object global(int rand, decimal acceleration = 5, decimal deceleration = 5);

/* The main() function must always be present.
It is the entry point for an iDrive program much like C and Java.*/

function main()
{
      print(‘‘Current speed is ’’ ++ myCar.currentSpeed ++ ‘‘ miles/hr’’);
      print(‘‘Current position is (’’ ++ myCar.currentXPosition ++ ‘‘, ’’
                        ++ myCar.currentYPosition ++ ‘‘)’’);
      print(‘‘Destination position is (’’ ++ myCar.destXPosition ++ ‘‘, ’’
                        ++ myCar.destYPosition ++ ‘‘)’’);
      print(‘‘Current heading is ’’ ++ myCar.heading);
      distanceToDestination();
      print(‘‘’’);

      while (myCar.distanceToDest > 0.5) {

            /* myCar.isClearToGo will take into account pedestrians, traffic signals,
            stop signs, and other vehicles on the roadway */

            if (myCar.isClearToGo) {
```

```
        /* To simulate such input in absence of sensors such as cameras,
        radar, and GPS, the program creates these objects randomly and
        incrementally.  */

        global.rand = random(5); {
        if (global.rand == 0 && myCar.currentSpeed >= 20) {
              myCar.isClearToGo = false;

              global.rand = random(4);

              if (global.rand == 0) {
                    createNewCar();
                    global.deceleration = 20;
                    print(‘‘DECELERATING QUICKLY DUE TO AN UNEXPECTED
                          APPROACHING CAR’’);
              }
              if (global.rand == 1) {
                    createNewPedestrian();
                    global.deceleration = 20;
                    print(‘‘DECELERATING QUICKLY DUE TO AN UNEXPECTED
                          PEDESTRIAN’’);
              }
              if (global.rand == 2) {
                    createNewTrafficSignal();
                    global.deceleration = 5;
                    print(‘‘DECELERATING DUE TO A TRAFFIC SIGNAL UP
                          AHEAD’’);
              }
              if (global.rand == 3) {
                    createNewStopSign();
                    global.deceleration = 5;
                    print(‘‘DECELERATING DUE TO A STOP SIGN UP AHEAD’’);
              }
              print(‘‘’’);
        }
}

if (myCar.isClearToGo) {
        if (myCar.currentSpeed < 45) {
              accelerate();
```

```
                }

                updateCurrentPosition();
                distanceToDestination();
            }
            else {
                if (myCar.currentSpeed > 0) {
                    decelerate();
                }
                else {
                    global.deceleration = 5;
                    myCar.isClearToGo = true;
                    accelerate();
                }

                updateCurrentPosition();
                distanceToDestination();
            }
            print(‘‘’’);
        }

        if (myCar.currentSpeed == 0) {
            print(‘‘Please walk this distance, its good for your health!!!’’);
            print(‘‘’’);
        }
        else {
            print(‘‘Approaching destination’’);
            print(‘‘’’);
            while (myCar.currentSpeed > 0 && myCar.distanceToDest > 0.1) {
                decelerate();
                updateCurrentPosition();
                distanceToDestination();
                print(‘‘’’);
            }
            print(‘‘Arrived at destination’’);
        }
}

/* Below are all the user defined functions */
```

```
/* accelerate increases the car speed by a specified amount */

function accelerate()
{
      myCar.currentSpeed = myCar.currentSpeed + global.acceleration;
      if (myCar.currentSpeed > 45) {
            myCar.currentSpeed = 45;
      }
      print(``Current speed increased to '' ++ myCar.currentSpeed
                  ++ `` miles/hr'');
}

/* decelerate decreases the car speed by a specified amount */

function decelerate()
{
      myCar.currentSpeed = myCar.currentSpeed - global.deceleration;
      if (myCar.currentSpeed < 0) {
            myCar.currentSpeed = 0;
      }
      print(``Current speed decreased to '' ++ myCar.currentSpeed
                  ++ `` miles/hr'');
}

/* updateCurrentPosition updates the current position of the car taking current
speed into account */

function updateCurrentPosition()
{
      if (streq(myCar.heading, ``North'')) {
            myCar.currentXPosition = 0;
            myCar.currentYPosition = myCar.currentYPosition +
                        (myCar.currentSpeed / 3600) * 20;
      }
      if (streq(myCar.heading, ``East'')) {
            myCar.currentXPosition = myCar.currentXPosition +
                        (myCar.currentSpeed / 3600) * 20;
            myCar.currentYPosition = 0;
      }
      if (streq(myCar.heading, ``West'')) {
```

```
            myCar.currentXPosition = myCar.currentXPosition -
                        (myCar.currentSpeed / 3600) * 20;
            myCar.currentYPosition = 0;
      }
      if (streq(myCar.heading, ``South'')) {
            myCar.currentXPosition = 0;
            myCar.currentYPosition = myCar.currentYPosition -
                        (myCar.currentSpeed / 3600) * 20;
      }

      print(``Current position is ('' ++ myCar.currentXPosition ++ ``, ''
                        ++ myCar.currentYPosition ++ ``)'');
}

/* distanceToDestination updates the distance left to the destination position
*/

function distanceToDestination()
{
      myCar.distanceToDest =
            sqrt((myCar.destXPosition - myCar.currentXPosition) ^ 2
            + (myCar.destYPosition - myCar.currentYPosition) ^ 2);
      print(``Current distance from destination = '' ++ myCar.distanceToDest
                  ++ `` miles'');
}

/* createNewCar creates a new vehicle object */

function createNewCar()
{
      vehicle newCar();
}

/* createNewPedestrian creates a new pedestrian object */

function createNewPedestrian()
{
      pedestrian newPedestrian();
}
```

```
/* createNewTrafficSignal creates a new traffic signal object */

function createNewTrafficSignal()
{
      trafficsignal newTrafficSignal();
}

/* createNewStopSign creates a new stop sign object */

function createNewStopSign()
{
      stopsign newStopSign();
}
```

9.11 Representative Program Output

```
Current speed is 0 miles/hr
Current position is (0, 0)
Destination position is (0, 50)
Current heading is North
Current distance from destination = 50. miles

Current speed increased to 5. miles/hr
Current position is (0, 0.0277777777778)
Current distance from destination = 49.9722222222 miles

Current speed increased to 10. miles/hr
Current position is (0, 0.0833333333334)
Current distance from destination = 49.9166666667 miles

Current speed increased to 15. miles/hr
Current position is (0, 0.166666666667)
Current distance from destination = 49.8333333333 miles

Current speed increased to 20. miles/hr
Current position is (0, 0.277777777778)
Current distance from destination = 49.7222222222 miles

Current speed increased to 25. miles/hr
```

```
Current position is (0, 0.416666666667)
Current distance from destination = 49.5833333333 miles

Current speed increased to 30. miles/hr
Current position is (0, 0.583333333334)
Current distance from destination = 49.4166666667 miles

Current speed increased to 35. miles/hr
Current position is (0, 0.777777777778)
Current distance from destination = 49.2222222222 miles

Current speed increased to 40. miles/hr
Current position is (0, 1.)
Current distance from destination = 49. miles

Current speed increased to 45. miles/hr
Current position is (0, 1.25)
Current distance from destination = 48.75 miles

DECELERATING DUE TO A STOP SIGN UP AHEAD

Current speed decreased to 40. miles/hr
Current position is (0, 1.47222222222)
Current distance from destination = 48.5277777778 miles

Current speed decreased to 35. miles/hr
Current position is (0, 1.66666666666)
Current distance from destination = 48.3333333333 miles

Current speed decreased to 30. miles/hr
Current position is (0, 1.83333333333)
Current distance from destination = 48.1666666667 miles

Current speed decreased to 25. miles/hr
Current position is (0, 1.97222222222)
Current distance from destination = 48.0277777778 miles

Current speed decreased to 20. miles/hr
Current position is (0, 2.08333333333)
Current distance from destination = 47.9166666667 miles
```

```
Current speed decreased to 15. miles/hr
Current position is (0, 2.16666666666)
Current distance from destination = 47.8333333333 miles

Current speed decreased to 10. miles/hr
Current position is (0, 2.22222222222)
Current distance from destination = 47.7777777777 miles

Current speed decreased to 5. miles/hr
Current position is (0, 2.25)
Current distance from destination = 47.75 miles

Current speed decreased to 0. miles/hr
Current position is (0, 2.25)
Current distance from destination = 47.75 miles

Current speed increased to 5. miles/hr
Current position is (0, 2.27777777778)
Current distance from destination = 47.7222222223 miles

Current speed increased to 10. miles/hr
Current position is (0, 2.33333333334)
Current distance from destination = 47.6666666667 miles

Current speed increased to 15. miles/hr
Current position is (0, 2.41666666667)
Current distance from destination = 47.5833333333 miles

Current speed increased to 20. miles/hr
Current position is (0, 2.52777777778)
Current distance from destination = 47.4722222222 miles

DECELERATING DUE TO A STOP SIGN UP AHEAD

Current speed decreased to 15. miles/hr
Current position is (0, 2.61111111111)
Current distance from destination = 47.3888888889 miles

Current speed decreased to 10. miles/hr
```

```
Current position is (0, 2.66666666667)
Current distance from destination = 47.3333333333 miles

Current speed decreased to 5. miles/hr
Current position is (0, 2.69444444445)
Current distance from destination = 47.3055555556 miles

Current speed decreased to 0. miles/hr
Current position is (0, 2.69444444445)
Current distance from destination = 47.3055555556 miles

Current speed increased to 5. miles/hr
Current position is (0, 2.72222222223)
Current distance from destination = 47.2777777778 miles

Current speed increased to 10. miles/hr
Current position is (0, 2.77777777779)
Current distance from destination = 47.2222222222 miles

Current speed increased to 15. miles/hr
Current position is (0, 2.86111111112)
Current distance from destination = 47.1388888889 miles

Current speed increased to 20. miles/hr
Current position is (0, 2.97222222223)
Current distance from destination = 47.0277777778 miles

Current speed increased to 25. miles/hr
Current position is (0, 3.11111111112)
Current distance from destination = 46.8888888889 miles

Current speed increased to 30. miles/hr
Current position is (0, 3.27777777779)
Current distance from destination = 46.7222222222 miles

Current speed increased to 35. miles/hr
Current position is (0, 3.47222222223)
Current distance from destination = 46.5277777778 miles

Current speed increased to 40. miles/hr
```

```
Current position is (0, 3.69444444445)
Current distance from destination = 46.3055555556 miles

Current speed increased to 45. miles/hr
Current position is (0, 3.94444444445)
Current distance from destination = 46.0555555555 miles

Current position is (0, 4.19444444445)
Current distance from destination = 45.8055555556 miles

Current position is (0, 4.44444444445)
Current distance from destination = 45.5555555556 miles

Current position is (0, 4.69444444445)
Current distance from destination = 45.3055555556 miles

Current position is (0, 4.94444444445)
Current distance from destination = 45.0555555556 miles

Current position is (0, 5.19444444445)
Current distance from destination = 44.8055555556 miles

Current position is (0, 5.44444444445)
Current distance from destination = 44.5555555556 miles

Current position is (0, 5.69444444445)
Current distance from destination = 44.3055555556 miles

Current position is (0, 5.94444444445)
Current distance from destination = 44.0555555556 miles

Current position is (0, 6.19444444445)
Current distance from destination = 43.8055555555 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED APPROACHING CAR

Current speed decreased to 25. miles/hr
Current position is (0, 6.33333333334)
Current distance from destination = 43.6666666667 miles
```

```
Current speed decreased to 5. miles/hr
Current position is (0, 6.36111111112)
Current distance from destination = 43.6388888889 miles

Current speed decreased to 0 miles/hr
Current position is (0, 6.36111111112)
Current distance from destination = 43.6388888889 miles

Current speed increased to 5. miles/hr
Current position is (0, 6.3888888889)
Current distance from destination = 43.611111111 miles

Current speed increased to 10. miles/hr
Current position is (0, 6.44444444446)
Current distance from destination = 43.5555555555 miles

Current speed increased to 15. miles/hr
Current position is (0, 6.52777777779)
Current distance from destination = 43.4722222222 miles

Current speed increased to 20. miles/hr
Current position is (0, 6.6388888889)
Current distance from destination = 43.3611111111 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED APPROACHING CAR

Current speed decreased to 0. miles/hr
Current position is (0, 6.6388888889)
Current distance from destination = 43.3611111111 miles

Current speed increased to 5. miles/hr
Current position is (0, 6.66666666668)
Current distance from destination = 43.3333333332 miles

Current speed increased to 10. miles/hr
Current position is (0, 6.72222222224)
Current distance from destination = 43.2777777777 miles

Current speed increased to 15. miles/hr
Current position is (0, 6.80555555557)
```

```
Current distance from destination = 43.1944444444 miles

Current speed increased to 20. miles/hr
Current position is (0, 6.91666666668)
Current distance from destination = 43.0833333333 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED APPROACHING CAR

Current speed decreased to 0. miles/hr
Current position is (0, 6.91666666668)
Current distance from destination = 43.0833333333 miles

Current speed increased to 5. miles/hr
Current position is (0, 6.94444444446)
Current distance from destination = 43.0555555555 miles

Current speed increased to 10. miles/hr
Current position is (0, 7.00000000002)
Current distance from destination = 43. miles

Current speed increased to 15. miles/hr
Current position is (0, 7.08333333335)
Current distance from destination = 42.9166666667 miles

Current speed increased to 20. miles/hr
Current position is (0, 7.19444444446)
Current distance from destination = 42.8055555554 miles

DECELERATING DUE TO A TRAFFIC SIGNAL UP AHEAD

Current speed decreased to 15. miles/hr
Current position is (0, 7.27777777779)
Current distance from destination = 42.7222222222 miles

Current speed decreased to 10. miles/hr
Current position is (0, 7.33333333335)
Current distance from destination = 42.6666666667 miles

Current speed decreased to 5. miles/hr
Current position is (0, 7.36111111113)
```

```
Current distance from destination = 42.6388888889 miles

Current speed decreased to 0. miles/hr
Current position is (0, 7.36111111113)
Current distance from destination = 42.6388888889 miles

Current speed increased to 5. miles/hr
Current position is (0, 7.38888888891)
Current distance from destination = 42.6111111111 miles

Current speed increased to 10. miles/hr
Current position is (0, 7.44444444447)
Current distance from destination = 42.5555555555 miles

Current speed increased to 15. miles/hr
Current position is (0, 7.5277777778)
Current distance from destination = 42.4722222222 miles

Current speed increased to 20. miles/hr
Current position is (0, 7.63888888891)
Current distance from destination = 42.3611111111 miles

DECELERATING DUE TO A TRAFFIC SIGNAL UP AHEAD

Current speed decreased to 15. miles/hr
Current position is (0, 7.72222222224)
Current distance from destination = 42.2777777778 miles

Current speed decreased to 10. miles/hr
Current position is (0, 7.7777777778)
Current distance from destination = 42.2222222222 miles

Current speed decreased to 5. miles/hr
Current position is (0, 7.80555555558)
Current distance from destination = 42.1944444444 miles

Current speed decreased to 0. miles/hr
Current position is (0, 7.80555555558)
Current distance from destination = 42.1944444444 miles
```

```
Current speed increased to 5. miles/hr
Current position is (0, 7.83333333336)
Current distance from destination = 42.1666666666 miles

Current speed increased to 10. miles/hr
Current position is (0, 7.88888888892)
Current distance from destination = 42.1111111111 miles

Current speed increased to 15. miles/hr
Current position is (0, 7.97222222225)
Current distance from destination = 42.0277777778 miles

Current speed increased to 20. miles/hr
Current position is (0, 8.08333333336)
Current distance from destination = 41.9166666666 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 0. miles/hr
Current position is (0, 8.08333333336)
Current distance from destination = 41.9166666666 miles

Current speed increased to 5. miles/hr
Current position is (0, 8.11111111114)
Current distance from destination = 41.8888888889 miles

Current speed increased to 10. miles/hr
Current position is (0, 8.1666666667)
Current distance from destination = 41.8333333332 miles

Current speed increased to 15. miles/hr
Current position is (0, 8.25000000003)
Current distance from destination = 41.75 miles

Current speed increased to 20. miles/hr
Current position is (0, 8.36111111114)
Current distance from destination = 41.6388888889 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN
```

```
Current speed decreased to 0. miles/hr
Current position is (0, 8.36111111114)
Current distance from destination = 41.6388888889 miles

Current speed increased to 5. miles/hr
Current position is (0, 8.38888888892)
Current distance from destination = 41.6111111111 miles

Current speed increased to 10. miles/hr
Current position is (0, 8.44444444448)
Current distance from destination = 41.5555555555 miles

Current speed increased to 15. miles/hr
Current position is (0, 8.52777777781)
Current distance from destination = 41.4722222222 miles

Current speed increased to 20. miles/hr
Current position is (0, 8.63888888892)
Current distance from destination = 41.361111111 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 0. miles/hr
Current position is (0, 8.63888888892)
Current distance from destination = 41.361111111 miles

Current speed increased to 5. miles/hr
Current position is (0, 8.6666666667)
Current distance from destination = 41.3333333333 miles

Current speed increased to 10. miles/hr
Current position is (0, 8.72222222226)
Current distance from destination = 41.2777777778 miles

Current speed increased to 15. miles/hr
Current position is (0, 8.80555555559)
Current distance from destination = 41.1944444444 miles

Current speed increased to 20. miles/hr
Current position is (0, 8.9166666667)
```

```
Current distance from destination = 41.0833333334 miles

Current speed increased to 25. miles/hr
Current position is (0, 9.05555555559)
Current distance from destination = 40.9444444444 miles

Current speed increased to 30. miles/hr
Current position is (0, 9.22222222226)
Current distance from destination = 40.7777777777 miles

Current speed increased to 35. miles/hr
Current position is (0, 9.4166666667)
Current distance from destination = 40.5833333333 miles

Current speed increased to 40. miles/hr
Current position is (0, 9.63888888892)
Current distance from destination = 40.3611111111 miles

Current speed increased to 45. miles/hr
Current position is (0, 9.88888888892)
Current distance from destination = 40.1111111111 miles

Current position is (0, 10.1388888889)
Current distance from destination = 39.8611111111 miles

Current position is (0, 10.3888888889)
Current distance from destination = 39.6111111112 miles

Current position is (0, 10.6388888889)
Current distance from destination = 39.3611111111 miles

Current position is (0, 10.8888888889)
Current distance from destination = 39.111111111 miles

Current position is (0, 11.1388888889)
Current distance from destination = 38.8611111111 miles

Current position is (0, 11.3888888889)
Current distance from destination = 38.6111111111 miles
```

```
DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 25. miles/hr
Current position is (0, 11.5277777778)
Current distance from destination = 38.4722222221 miles

Current speed decreased to 5. miles/hr
Current position is (0, 11.5555555556)
Current distance from destination = 38.4444444444 miles

Current speed decreased to 0 miles/hr
Current position is (0, 11.5555555556)
Current distance from destination = 38.4444444444 miles

Current speed increased to 5. miles/hr
Current position is (0, 11.5833333334)
Current distance from destination = 38.4166666666 miles

Current speed increased to 10. miles/hr
Current position is (0, 11.638888889)
Current distance from destination = 38.361111111 miles

Current speed increased to 15. miles/hr
Current position is (0, 11.7222222223)
Current distance from destination = 38.2777777777 miles

Current speed increased to 20. miles/hr
Current position is (0, 11.8333333334)
Current distance from destination = 38.1666666666 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 0. miles/hr
Current position is (0, 11.8333333334)
Current distance from destination = 38.1666666666 miles

Current speed increased to 5. miles/hr
Current position is (0, 11.8611111112)
Current distance from destination = 38.1388888888 miles
```

```
Current speed increased to 10. miles/hr
Current position is (0, 11.9166666668)
Current distance from destination = 38.0833333332 miles

Current speed increased to 15. miles/hr
Current position is (0, 12.0000000001)
Current distance from destination = 37.9999999999 miles

Current speed increased to 20. miles/hr
Current position is (0, 12.1111111112)
Current distance from destination = 37.8888888888 miles

DECELERATING QUICKLY DUE TO AN UNEXPECTED PEDESTRIAN

Current speed decreased to 0. miles/hr
Current position is (0, 12.1111111112)
Current distance from destination = 37.8888888888 miles

Current speed increased to 5. miles/hr
Current position is (0, 12.138888889)
Current distance from destination = 37.861111111 miles

Current speed increased to 10. miles/hr
Current position is (0, 12.1944444446)
Current distance from destination = 37.8055555554 miles

Current speed increased to 15. miles/hr
Current position is (0, 12.2777777779)
Current distance from destination = 37.7222222221 miles

Current speed increased to 20. miles/hr
Current position is (0, 12.388888889)
Current distance from destination = 37.6111111109 miles

Current speed increased to 25. miles/hr
Current position is (0, 12.5277777779)
Current distance from destination = 37.4722222221 miles

Current speed increased to 30. miles/hr
Current position is (0, 12.6944444446)
```

```
Current distance from destination = 37.3055555554 miles

Current speed increased to 35. miles/hr
Current position is (0, 12.888888889)
Current distance from destination = 37.111111111 miles

Current speed increased to 40. miles/hr
Current position is (0, 13.1111111112)
Current distance from destination = 36.8888888888 miles

Current speed increased to 45. miles/hr
Current position is (0, 13.3611111112)
Current distance from destination = 36.6388888889 miles

Current position is (0, 13.6111111112)
Current distance from destination = 36.3888888888 miles

Current position is (0, 13.8611111112)
Current distance from destination = 36.1388888888 miles

Current position is (0, 14.1111111112)
Current distance from destination = 35.8888888888 miles

Current position is (0, 14.3611111112)
Current distance from destination = 35.6388888888 miles

Current position is (0, 14.6111111112)
Current distance from destination = 35.3888888887 miles

Current position is (0, 14.8611111112)
Current distance from destination = 35.1388888888 miles

Current position is (0, 15.1111111112)
Current distance from destination = 34.8888888889 miles

Current position is (0, 15.3611111112)
Current distance from destination = 34.6388888888 miles

Current position is (0, 15.6111111112)
Current distance from destination = 34.3888888889 miles
```

```
Current position is (0, 15.8611111112)
Current distance from destination = 34.1388888888 miles

Current position is (0, 16.1111111112)
Current distance from destination = 33.8888888888 miles

Current position is (0, 16.3611111112)
Current distance from destination = 33.6388888888 miles

Current position is (0, 16.6111111112)
Current distance from destination = 33.3888888888 miles

Current position is (0, 16.8611111112)
Current distance from destination = 33.1388888887 miles

Current position is (0, 17.1111111112)
Current distance from destination = 32.8888888888 miles

Current position is (0, 17.3611111112)
Current distance from destination = 32.6388888889 miles

Current position is (0, 17.6111111112)
Current distance from destination = 32.3888888888 miles

Current position is (0, 17.8611111112)
Current distance from destination = 32.1388888889 miles

Current position is (0, 18.1111111112)
Current distance from destination = 31.8888888888 miles

Current position is (0, 18.3611111112)
Current distance from destination = 31.6388888888 miles

Current position is (0, 18.6111111112)
Current distance from destination = 31.3888888888 miles

Current position is (0, 18.8611111112)
Current distance from destination = 31.1388888888 miles
```

```
Current position is (0, 19.1111111112)
Current distance from destination = 30.8888888888 miles

Current position is (0, 19.3611111112)
Current distance from destination = 30.6388888888 miles

Current position is (0, 19.6111111112)
Current distance from destination = 30.3888888888 miles

Current position is (0, 19.8611111112)
Current distance from destination = 30.1388888888 miles

Current position is (0, 20.1111111112)
Current distance from destination = 29.8888888888 miles

Current position is (0, 20.3611111112)
Current distance from destination = 29.6388888888 miles

Current position is (0, 20.6111111112)
Current distance from destination = 29.3888888888 miles

Current position is (0, 20.8611111112)
Current distance from destination = 29.1388888888 miles

Current position is (0, 21.1111111112)
Current distance from destination = 28.8888888888 miles

Current position is (0, 21.3611111112)
Current distance from destination = 28.6388888888 miles

Current position is (0, 21.6111111112)
Current distance from destination = 28.3888888888 miles

Current position is (0, 21.8611111112)
Current distance from destination = 28.1388888888 miles

Current position is (0, 22.1111111112)
Current distance from destination = 27.8888888888 miles

Current position is (0, 22.3611111112)
```

```
Current distance from destination = 27.6388888888 miles

Current position is (0, 22.6111111112)
Current distance from destination = 27.3888888888 miles

Current position is (0, 22.8611111112)
Current distance from destination = 27.1388888888 miles

Current position is (0, 23.1111111112)
Current distance from destination = 26.8888888888 miles

Current position is (0, 23.3611111112)
Current distance from destination = 26.6388888888 miles

Current position is (0, 23.6111111112)
Current distance from destination = 26.3888888888 miles

Current position is (0, 23.8611111112)
Current distance from destination = 26.1388888888 miles

Current position is (0, 24.1111111112)
Current distance from destination = 25.8888888888 miles

Current position is (0, 24.3611111112)
Current distance from destination = 25.6388888888 miles

Current position is (0, 24.6111111112)
Current distance from destination = 25.3888888888 miles

Current position is (0, 24.8611111112)
Current distance from destination = 25.1388888888 miles

Current position is (0, 25.1111111112)
Current distance from destination = 24.8888888888 miles

Current position is (0, 25.3611111112)
Current distance from destination = 24.6388888888 miles

Current position is (0, 25.6111111112)
Current distance from destination = 24.3888888888 miles
```

```
Current position is (0, 25.8611111112)
Current distance from destination = 24.1388888888 miles

Current position is (0, 26.1111111112)
Current distance from destination = 23.8888888888 miles

Current position is (0, 26.3611111112)
Current distance from destination = 23.6388888888 miles

Current position is (0, 26.6111111112)
Current distance from destination = 23.3888888888 miles

Current position is (0, 26.8611111112)
Current distance from destination = 23.1388888888 miles

Current position is (0, 27.1111111112)
Current distance from destination = 22.8888888888 miles

Current position is (0, 27.3611111112)
Current distance from destination = 22.6388888888 miles

Current position is (0, 27.6111111112)
Current distance from destination = 22.3888888888 miles

Current position is (0, 27.8611111112)
Current distance from destination = 22.1388888888 miles

Current position is (0, 28.1111111112)
Current distance from destination = 21.8888888888 miles

Current position is (0, 28.3611111112)
Current distance from destination = 21.6388888888 miles

Current position is (0, 28.6111111112)
Current distance from destination = 21.3888888888 miles

Current position is (0, 28.8611111112)
Current distance from destination = 21.1388888888 miles
```

```
Current position is (0, 29.1111111112)
Current distance from destination = 20.8888888888 miles

Current position is (0, 29.3611111112)
Current distance from destination = 20.6388888888 miles

Current position is (0, 29.6111111112)
Current distance from destination = 20.3888888888 miles

Current position is (0, 29.8611111112)
Current distance from destination = 20.1388888888 miles

Current position is (0, 30.1111111112)
Current distance from destination = 19.8888888888 miles

Current position is (0, 30.3611111112)
Current distance from destination = 19.6388888888 miles

Current position is (0, 30.6111111112)
Current distance from destination = 19.3888888888 miles

Current position is (0, 30.8611111112)
Current distance from destination = 19.1388888888 miles

Current position is (0, 31.1111111112)
Current distance from destination = 18.8888888888 miles

Current position is (0, 31.3611111112)
Current distance from destination = 18.6388888888 miles

Current position is (0, 31.6111111112)
Current distance from destination = 18.3888888888 miles

Current position is (0, 31.8611111112)
Current distance from destination = 18.1388888888 miles

Current position is (0, 32.1111111112)
Current distance from destination = 17.8888888888 miles

Current position is (0, 32.3611111112)
```

```
Current distance from destination = 17.6388888888 miles

Current position is (0, 32.6111111112)
Current distance from destination = 17.3888888888 miles

Current position is (0, 32.8611111112)
Current distance from destination = 17.1388888888 miles

Current position is (0, 33.1111111112)
Current distance from destination = 16.8888888888 miles

Current position is (0, 33.3611111112)
Current distance from destination = 16.6388888888 miles

Current position is (0, 33.6111111112)
Current distance from destination = 16.3888888888 miles

Current position is (0, 33.8611111112)
Current distance from destination = 16.1388888888 miles

Current position is (0, 34.1111111112)
Current distance from destination = 15.8888888888 miles

Current position is (0, 34.3611111112)
Current distance from destination = 15.6388888888 miles

Current position is (0, 34.6111111112)
Current distance from destination = 15.3888888888 miles

Current position is (0, 34.8611111112)
Current distance from destination = 15.1388888888 miles

Current position is (0, 35.1111111112)
Current distance from destination = 14.8888888888 miles

Current position is (0, 35.3611111112)
Current distance from destination = 14.6388888888 miles

Current position is (0, 35.6111111112)
Current distance from destination = 14.3888888888 miles
```

```
Current position is (0, 35.8611111112)
Current distance from destination = 14.1388888888 miles

Current position is (0, 36.1111111112)
Current distance from destination = 13.8888888888 miles

Current position is (0, 36.3611111112)
Current distance from destination = 13.6388888888 miles

Current position is (0, 36.6111111112)
Current distance from destination = 13.3888888888 miles

Current position is (0, 36.8611111112)
Current distance from destination = 13.1388888888 miles

Current position is (0, 37.1111111112)
Current distance from destination = 12.8888888888 miles

Current position is (0, 37.3611111112)
Current distance from destination = 12.6388888888 miles

Current position is (0, 37.6111111112)
Current distance from destination = 12.3888888888 miles

Current position is (0, 37.8611111112)
Current distance from destination = 12.1388888888 miles

Current position is (0, 38.1111111112)
Current distance from destination = 11.8888888888 miles

Current position is (0, 38.3611111112)
Current distance from destination = 11.6388888888 miles

Current position is (0, 38.6111111112)
Current distance from destination = 11.3888888888 miles

Current position is (0, 38.8611111112)
Current distance from destination = 11.1388888888 miles
```

```
Current position is (0, 39.1111111112)
Current distance from destination = 10.8888888888 miles

Current position is (0, 39.3611111112)
Current distance from destination = 10.6388888888 miles

Current position is (0, 39.6111111112)
Current distance from destination = 10.3888888888 miles

Current position is (0, 39.8611111112)
Current distance from destination = 10.1388888888 miles

Current position is (0, 40.1111111112)
Current distance from destination = 9.8888888888 miles

Current position is (0, 40.3611111112)
Current distance from destination = 9.6388888888 miles

Current position is (0, 40.6111111112)
Current distance from destination = 9.3888888888 miles

Current position is (0, 40.8611111112)
Current distance from destination = 9.1388888888 miles

Current position is (0, 41.1111111112)
Current distance from destination = 8.8888888888 miles

Current position is (0, 41.3611111112)
Current distance from destination = 8.6388888888 miles

Current position is (0, 41.6111111112)
Current distance from destination = 8.3888888888 miles

Current position is (0, 41.8611111112)
Current distance from destination = 8.1388888888 miles

Current position is (0, 42.1111111112)
Current distance from destination = 7.8888888888 miles

Current position is (0, 42.3611111112)
```

```
Current distance from destination = 7.6388888888 miles

Current position is (0, 42.6111111112)
Current distance from destination = 7.3888888888 miles

Current position is (0, 42.8611111112)
Current distance from destination = 7.1388888888 miles

Current position is (0, 43.1111111112)
Current distance from destination = 6.8888888888 miles

Current position is (0, 43.3611111112)
Current distance from destination = 6.6388888888 miles

Current position is (0, 43.6111111112)
Current distance from destination = 6.3888888888 miles

Current position is (0, 43.8611111112)
Current distance from destination = 6.1388888888 miles

Current position is (0, 44.1111111112)
Current distance from destination = 5.8888888888 miles

Current position is (0, 44.3611111112)
Current distance from destination = 5.6388888888 miles

Current position is (0, 44.6111111112)
Current distance from destination = 5.3888888888 miles

Current position is (0, 44.8611111112)
Current distance from destination = 5.1388888888 miles

Current position is (0, 45.1111111112)
Current distance from destination = 4.8888888888 miles

Current position is (0, 45.3611111112)
Current distance from destination = 4.6388888888 miles

Current position is (0, 45.6111111112)
Current distance from destination = 4.3888888888 miles
```

```
Current position is (0, 45.8611111112)
Current distance from destination = 4.1388888888 miles

Current position is (0, 46.1111111112)
Current distance from destination = 3.8888888888 miles

Current position is (0, 46.3611111112)
Current distance from destination = 3.6388888888 miles

Current position is (0, 46.6111111112)
Current distance from destination = 3.3888888888 miles

Current position is (0, 46.8611111112)
Current distance from destination = 3.1388888888 miles

Current position is (0, 47.1111111112)
Current distance from destination = 2.8888888888 miles

Current position is (0, 47.3611111112)
Current distance from destination = 2.6388888888 miles

Current position is (0, 47.6111111112)
Current distance from destination = 2.3888888888 miles

Current position is (0, 47.8611111112)
Current distance from destination = 2.1388888888 miles

Current position is (0, 48.1111111112)
Current distance from destination = 1.8888888888 miles

Current position is (0, 48.3611111112)
Current distance from destination = 1.6388888888 miles

Current position is (0, 48.6111111112)
Current distance from destination = 1.3888888888 miles

Current position is (0, 48.8611111112)
Current distance from destination = 1.1388888888 miles
```

```
Current position is (0, 49.1111111112)
Current distance from destination = 0.8888888888 miles

Current position is (0, 49.3611111112)
Current distance from destination = 0.6388888888 miles

Current position is (0, 49.6111111112)
Current distance from destination = 0.3888888888 miles

Approaching destination

Current speed decreased to 40. miles/hr
Current position is (0, 49.8333333334)
Current distance from destination = 0.1666666666 miles

Current speed decreased to 35. miles/hr
Current position is (0, 50.0277777778)
Current distance from destination = 0.0277777778 miles

Arrived at destination
```

9.12 Some Sample Input Test Files

9.12.1 test23.iDrive

```
/* To test nested for loops and while loops:
declare a global variable, run all combinations of
nested for loops and while loops, and print the results */

object a(int x, int xx);

function main()
{
for (a.x=0 ; a.x < 5 ; a.x=a.x + 1) {
a.xx = 0;
while (a.xx < 5) {
print(a.x ++ "-" ++ a.xx);
a.xx = a.xx + 1;
}
```

```
print("");
}

for (a.x=0 ; a.x < 5 ; a.x=a.x + 1) {
for (a.xx=0 ; a.xx < 5 ; a.xx=a.xx + 1) {
print(a.x ++ "-" ++ a.xx);
}
print("");
}

a.x = 0;
while (a.x < 5) {
for (a.xx=0 ; a.xx < 5 ; a.xx=a.xx + 1) {
print(a.x ++ "-" ++ a.xx);
}
print("");
  a.x=a.x + 1;
}

a.x = 0;
while (a.x < 5) {
a.xx = 0;
while (a.xx < 5) {
print(a.x ++ "-" ++ a.xx);
a.xx = a.xx + 1;
}
print("");
  a.x=a.x + 1;
  }

print(a.x);
print(a.xx);
}
```

9.12.2 test27.iDrive

```
/* To test static scoping of variables:
declare a global variable and a local variable with the same name,
update the local variable, and print both the variables */
```

```
object a(int x=1, string y="John", boolean z=true, decimal d=1.0);

function printObject(obj)
{
print(obj.x);
print(obj.y);
print(obj.z);
print(obj.d);
}

function printGlobalA()
{
printObject(a);
}

function main()
{
object a(int x, string y, boolean z, decimal d);

a.x = 100;
a.y = "John Smith";
a.z = false;
a.d = 2.5;

printObject(a);

printGlobalA();
}
```

9.12.3 test29.iDrive

```
/* To further test scope of a global variable:
declare a global variable, invoke another funtion mutateGlobalObject(),
update the global variable in that function, and print
the variable just before the funtion returns to show the
altered state of the variable and just after the function returns
to again show the altered state of the variable */


object a(int x=1, string y="John", boolean z=true, decimal d=1.0);
```

```
function printObject(obj)
{
print(obj.x);
print(obj.y);
print(obj.z);
print(obj.d);
}

function mutateGlobalObject()
{
printObject(a);

a.x = 100;
a.y = "Random";
a.z = false;
a.d = sqrt(a.x);

printObject(a);
}

function main()
{
printObject(a);

mutateGlobalObject();

printObject(a);
}
```

■