WarmFusion

A language for developing web applications

COMS W4115 – Final Report

Robert Dong (rd2439)

December 19, 2012

Table of Contents

1 Introduction

WarmFusion is a web application development language similar to Adobe ColdFusion which was initially released in 1995. Most existing web application development platforms require development in a language similar to C. Learning these languages usually isn't a problem for experienced programmers, however they can often be difficult to pick up for people who only know HTML and have little programming experience. WarmFusion has syntax similar to HTML/XML and is placed inline with existing HTML/XML code. This allows the language to be learned more easily by people who understand HTML.

Once compiled, WarmFusion executables can be used as CGI binaries on a web server such as Microsoft IIS. The output generated by a WarmFusion application is generally viewed in a web browser. The ease of writing WarmFusion code along with the option to deploy compiled code on a web server allows for the rapid development of efficient web applications.

2 Language Tutorial

Getting Started

To create your first application, create a file named file.wfm with the following text:

```
<wfset total = 5 * 3 />
<wfoutput>Five multiplied by three is #total#</wfoutput>
```

Next, you will learn how to compile and run your application.

Compiling and Running

To compile the application in Linux, the Mono C# compiler (mono-mcs) must be installed as well as libmono-system-web2.0-cil. These can generally be installed using the following commands:

```
apt-get install mono-mcs
apt-get install libmono-system-web2.0-cil
```

Use the warmfusionc.sh compilation script to compile your application. Assuming you have a file.wfm source file, you can compile the code as follows:

```
./warmfusionc.sh file.wfm
```

A file named file.exe should be generated which can be executed by running:

```
./file.exe
```

Although the file has an .exe extension, it should execute without a problem in Linux.

Alternatively, if you would like to compile using the Microsoft C# compiler in Windows, this can be done manually by running:

```
make clean
make
ren warmfusion warmfusion.exe
copy stub.cs file.cs
warmfusion.exe < file.wfm >> file.cs
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /r:System.Web.dll file.cs
```

A file named file.exe should then be generated which can be executed.

Deploying as a CGI Binary on a Web Server (Optional)

Your executable can also be optionally deployed as a CGI binary to a web server. Below are instructions for deployment to Microsoft IIS 7.5:

1. Ensure IIS is installed along with the CGI option in application development features. Your .exe should be placed somewhere in the C:\Inetpub\wwwroot\ directory.
2. Open IIS Manager, click on the server in the Connections section and open the "ISAPI and CGI Restrictions" settings page.
3. Click Add, enter the path to the .exe file that was generated after compilation, enter a description and check the box stating "Allow extension path to execute". Click OK.
4. Click on the server again in the Connections section and open the "Handler Mappings" settings page.
5. Click on the "CGI-exe" item and click "Edit Feature Permissions" and ensure the box that says "Execute" is checked. Click OK.
6. You should now be able to browse to your page using a URL similar to http://localhost/file.exe

Variables

You do not need to declare variables in WarmFusion. Variables can hold either a number or a string. For example, here is how you set a variable named price to 1299.95:

```
<wfset price = 1299.95 />
```

Numbers in WarmFusion are precise unlike many other languages. Therefore, if you added the value 0.01 to a variable 100,000 times, you are guaranteed to end up with the value 1000. Other languages use floating point arithmetic and might end up with a value such as 999.92, which is not great when working with monetary values.

You can also set a variable to a string as follows:

```
<wfset item = 'Computer' />
```

Note that when placing a string literal in the code, you will need to use single quotes. There is a set of built-in string functions listed in the Language Reference Manual. If you attempt to use a string in an operator that requires a number, WarmFusion will attempt to do the conversion automatically for you.

Outputting text

To output text to the user, the wfoutput tag is used. This tag is often used to return HTML to the user. To output a variable within a wfoutput tag, use the syntax #var#. No other WarmFusion statements or expressions can be placed within a wfoutput tag. Here is an example:

```
<wfset price = 1299.95 />
<wfset item = 'Computer' />
```

```
<wfoutput>
	<div id="main">
		<b>Checkout Page</b><br />
		You are purchasing the #item# for a total of $#price#
	</div>
</wfoutput>
```

Loops and Conditionals

Loops can be created using the wfloop tag. To create a "for" loop using this tag, provide an "index" attribute indicating the variable that is set during the loop, a "from" attribute indicating what number the loop starts at, and a "to" attribute indicating what number the loop ends at.

Conditional statements can be created using the wfif tag. These are some operators that may be helpful when creating conditional statements: +, -, *, /, %, NOT, LT, LTE, GT, GTE, EQ, NEQ, AND, OR. If these operators are not self-explanatory, see the Language Reference Manual. Here is an example:

```
<wfloop index="i" from="1" to="100">
	<wfif i % 2 EQ 0>
		<wfoutput>#i# is even!<br /></wfoutput>
	<wfelse>
		<wfoutput>#i# is odd!<br /></wfoutput>
	</wfif>
</wfloop>
```

Functions

Functions can have arguments and can optionally return a value. Here is an example:

```
<wffunction name="CalculateAreaOfSquare">
	<wfargument name="sideLength" />
	<wfset area = sideLength * sideLength />
	<wfreturn area />
</wffunction>
```

As expected, the name of this function is CalculateAreaOfSquare, which takes one argument named sideLength. You do not need to specify the type of a variable in WarmFusion. The area of a square is calculated and returned. Here is an example of how to call this function:

```
<wfset calculatedArea = CalculateAreaOfSquare(5) />
```

The variable calculatedArea will be set to 25. You can also call a function without assigning the return value to a variable as follows:

```
<wfset DisplayAreaOfSquare(5) />
```

End of Tutorial

You are now ready to start developing web applications in WarmFusion! To see a full example using all of the features above, see the Example section of the Language Reference Manual.

3 WarmFusion Reference Manual

1. Reference Manual Introduction

WarmFusion has syntax similar to HTML/XML and is placed inline with existing HTML/XML code. The output generated by a WarmFusion application is often viewed in a web browser.

2. Lexical Conventions

2.1. Comments

The characters `<!---` introduce a comment, which terminates with the characters `--->`. Comments can be placed within comments.

2.2. Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter. The underscore counts as a letter. Identifiers are case-sensitive. Identifiers may not be longer than 100 characters.

2.3. Keywords

The following identifiers are reserved keywords: `NOT`, `LT`, `LTE`, `GT`, `GTE`, `EQ`, `NEQ`, `AND`, `OR`.

2.4. Numerical Constants

A numerical constant consists of an integer part and an optional decimal part. The integer part consists of one or more digits. The decimal part consists of a "`.`" followed by one or more digits. Numerical constants may not contain more than 25 digits.

2.5. String Constants

A string constant is a sequence of characters surrounded by single quotes. To place a single quote character within a string, use two single quotes. e.g. 'they''re' is a valid string constant.

3. Types

WarmFusion supports two fundamental types: number and string. A numerical constant as described above is considered a number while a string constant is considered a string.

WarmFusion is a dynamically typed language. Strings are automatically converted to numbers when necessary and vice-versa. An error will be shown at runtime in the event a string is used in an operation that requires a number.

Variable declarations are not necessary. If a variable is read before it is written to, an empty string will be returned. Empty strings evaluate to 0 if being used as a number.

4. Expressions

4.1. Unary Operators

4.1.1. – expression

The result is the negative of the expression. The type of the expression must be number.

4.1.2. NOT expression

The result of the logical negation operator NOT is 1 if the value of the expression is 0 or 'false' (with any variation of capitalization), or 0 otherwise.

4.2. Multiplicative Operators

*4.2.1. expression * expression*

The binary * operator indicates multiplication. The type of both expressions must be number.

4.2.2. expression / expression

The binary / operator indicates division. The type of both expressions must be number.

4.3. Additive Operators

4.3.1. expression + expression

The result is the sum of the expressions. The type of both expressions must be number.

4.3.2. expression – expression

The result is the difference of the expressions. The type of both expressions must be number.

4.4. Relational Operators

4.4.1. expression LT expression

4.4.2. expression GT expression

4.4.3. expression LTE expression

4.4.4. expression GTE expression

The operators LT (less than), GT (greater than), LTE (less than or equal to) and GTE (greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true. The type of both expressions must be number.

4.5. Equality Operators

4.5.1. expression EQ expression

4.5.2. expression NEQ expression

The EQ (equal to) and the NEQ (not equal to) operators yield 0 if the specified relation is false and 1 if it is true.

4.6. *expression % expression*

The binary % operator yields the remainder from the division of the first expression by the second. The type of both expressions must be number.

4.7. *expression & expression*

The result is a string of the two concatenated expressions.

5. Statements

Except as described, statements are executed in sequence.

5.1. Assignment Statement

Assigning a value to a variable takes place as follows:

```
<wfset lvalue = expression />
```

5.2. Conditional Statement

The two forms of the conditional statement are:

```
<wfif expression>
     statement
</wfif>
```

and

```
<wfif expression>
     statement
<wfelse>
     statement
</wfif>
```

In both cases, the expression is evaluated and if it is not 0 and is not the string 'false' (with any variation of capitalization), the first substatement will be executed. In the second case, the second substatement is executed if the expression is 0 or 'false'. The `wfelse` statement is associated with the most recent `wfif` statement.

5.3. Loop Statements

5.3.1. Conditional Loop

A conditional loop has the form:

```
<wfloop condition="expression">
      statement
</wfloop>
```

The sub-statements are executed repeatedly as long as the expression is not 0 and it is not the string 'false' (with any variation of capitalization). The test takes place before each execution of the statement.

5.3.2. Loop with counter

A loop with a counter has the form:

```
<wfloop index="variable" from="number1" to="number2">
      statement
</wfloop>
```

The variable is initialized to number1. The sub-statements will be executed repeatedly as long as the value of variable is less than or equal to number2. The test takes place before each execution of the statement. After a sub-statement execution, variable is incremented by one.

5.4. Argument Statement

A function specifies the names of its arguments in the following form:

```
<wfargument name="argName" />
```

A `wfargument` tag may only follow the opening `wffunction` tag or another `wfargument` tag. The name of the argument may not exceed 100 characters.

5.5. Return Statement

A function returns to its caller by means of the `wfreturn` statement, which has the form:

```
<wfreturn expression />
```

The use of a return statement within a function is optional.

5.6. Output Statement

All text between a beginning <wfoutput> and the ending </wfoutput> tag will be outputted. Within the wfoutput tag, variables can be outputted by using the following notation:

```
#variable#
```

To output a single # symbol, ## should be used. No statements or expressions will be evaluated within a wfoutput tag. This tag is generally used to output HTML to the user.

6. Functions

6.1. Function Declarations

A function declaration has the form:

```
<wffunction name="NameOfFunction">
	<wfargument name="arg1" />
	<wfargument name="arg2" />
	…
	<wfargument name="argN" />

	statement

	<wfreturn 0 />
</wffunction>
```

A simple example of a complete function definition is:

```
<wffunction name="CalculateAreaOfSquare">
	<wfargument name="sideLength" />
	<wfset area = sideLength * sideLength />
	<wfreturn area />
</wffunction>
```

Function names cannot exceed 100 characters in length. A function cannot have more than 100 arguments. Multiple arguments within a function cannot have the same name. Multiple functions may not have the same name.

6.2. Built-in Functions

GetTickCount()	Returns the number of milliseconds since the system was started. The value wraps around after $2^{32} - 1$ milliseconds.
Sqr(number)	Returns the square root of the given number.
LCase(string)	Returns the string in lower case format.
UCase(string)	Returns the string in upper case format.
Len(string)	Returns the length of the string.
Left(string, length)	Returns the leftmost [length] characters of the given string. An error will occur if the string is shorter than [length] characters.
Mid(string, position, length)	Returns a substring of the given string starting at [position] and continuing for [length]. An error will occur if the string is shorter than ([length] - [position]) + 1 characters. The first character in the string starts at position 1. Example: Mid('Hello', 1, 1) will return 'H'.
Right(string, length)	Returns the rightmost [length] characters of the given string. An error will occur if the string is shorter than [length] characters.

ListLen(string)	A list is a comma delimited string. ListLen returns the number of values in the list. Example: ListLen('a,b,c') will return 3.
ListGetAt(string, position)	A list is a comma delimited string. ListGetAt returns the value of the list at [position]. The first element of the list is element 1. An error will occur if [position] is not a valid position in the list. Example: ListGetAt('a,b,c', 2) will return 'b'.
ListSetAt(string, position, value)	A list is a comma delimited string. ListSetAt returns a list with the value at [position] changed. The first element of the list is element 1. An error will occur if [position] is not a valid position in the list. Example: ListSetAt('a,b,c', 2, 'x') will return 'a,x,c'.
ListAppend(string, value)	A list is a comma delimited string. ListAppend returns a list with a new item added at the end. Example: ListAppend('a,b,c', 'x') will return 'a,b,c,x'.
ListFind(string, value)	A list is a comma delimited string. ListFind will return the index of the first occurrence of [value] within the list. The first element in the list is 1. 0 will be returned if the item could not be found. Example: ListFind('a,b,c', 'b') will return 2.
Rand()	Returns a random number between 0 and 1.
Now()	Returns a string of the current date and time. Example: '12/19/2012 11:59:00 PM'
HTMLEditFormat(string)	Takes a string and returns the encoded version with HTML entities for proper output to an HTML page. Example: HTMLEditFormat('hello<>') returns 'he&llo<>'.
URLEncodedFormat(string)	Takes a string and returns the encoded version with entities appropriate for use in a URL. Example: URLEncodedFormat('hello<>') returns 'he%26llo%3c%3e'.
GetQueryParameter(string)	Returns the URL query parameter associated with the given name. Returns an empty string if the name does not exist in the query string. Example: If the user is at the URL 'http://localhost/wf.exe?page=7', then GetQueryParameter('page') would return '7'. (This function is only useful when running a WarmFusion application as a CGI binary on a web server.)

7. Scope Rules

A function's arguments are considered local to that function. All other variables share the same global scope. Even if a variable is first set within a function, that variable will be visible after the function has finished executing. Similarly, if a variable is first set within a loop, the variable can still be used after the loop has completed.

8. Example

An example of WarmFusion code is shown below. WarmFusion code is mixed in with the HTML code which allows rapid development of simple web applications. The code declares two functions `CalculateAreaOfSquare` and `CalculateDiagonalOfSquare`, both of which take one argument. An example of a comment in WarmFusion is shown in the second function.

An HTML table is printed out and WarmFusion loops through the numbers 1 through 10 to display each row. In the table, there is a conditional statement which highlights a row gray if it is odd. The two functions are called in the loop to create the table of data. As shown, all WarmFusion statements are HTML/XML tags that begin with "`wf`".

8.1. Example Code

```
<wffunction name="CalculateAreaOfSquare">
	<wfargument name="sideLength" />
	<wfset area = sideLength * sideLength />
	<wfreturn area />
</wffunction>

<wffunction name="CalculateDiagonalOfSquare">
	<wfargument name="sideLength" />
	<!--- Length of a side multiplied by the square root of 2 --->
	<wfset diagonal = sideLength * Sqr(2) />
	<wfreturn diagonal />
</wffunction>

<wfoutput>
	<h1>Square Information:</h1>
	<table border="1">
		<tr>
			<th>Side Length</th>
			<th>Area of Square</th>
			<th>Diagonal of Square</th>
		</tr>
</wfoutput>

<wfloop index="i" from="1" to="10">
	<wfif i % 2 EQ 1>
		<wfset rowBgColor = 'lightgray' />
	<wfelse>
		<wfset rowBgColor = 'white' />
	</wfif>

	<wfset area = CalculateAreaOfSquare(i) />
	<wfset diagonal = CalculateDiagonalOfSquare(i) />

	<wfoutput>
		<tr bgcolor="#rowBgColor#">
			<td>#i#</td>
			<td>#area#</td>
			<td>#diagonal#</td>
		</tr>
	</wfoutput>
</wfloop>
```

```
<wfoutput>
        </table>
</wfoutput>
```

8.2. Output From Example

HTML code would be generated from the previous example. It would be shown in a web browser as follows:

Square Information:

Side Length	Area of Square	Diagonal of Square
1	1	1.4142135623731
2	4	2.8284271247462
3	9	4.2426406871193
4	16	5.6568542494924
5	25	7.0710678118655
6	36	8.4852813742386
7	49	9.8994949366117
8	64	11.3137084989848
9	81	12.7279220613579
10	100	14.1421356237310

4 Project Plan

Process

The project initially started off with the proposal followed by the development of the language reference manual. When developing the language reference manual, I often referred to the C language reference manual and the Adobe ColdFusion documentation to ensure I was coming up with an appropriate language specification. I then researched several O'Caml IDEs, and decided on using OCaIDE for development.

Before working on any code for WarmFusion, I thoroughly inspected the MicroC example code and reviewed the MicroC lecture several times. I wanted to ensure I had a full understanding of how MicroC worked to ensure I could build WarmFusion without any problems and have a clean design. I later moved on to creating the scanner and parser. While working on the scanner and parser, I made modifications to the MicroC bytecode sample to test my changes. Although I originally considered the scanner and parser the easier part, after moving on to the abstract syntax tree and compilation parts, I had to go back and change several parts of the scanner and parser. I also ran into problems that could probably only be solved cleanly by developing several different scanners that work in different ways based on what section of the code is being lexically analyzed.

I later worked on the compiler and the built-in functions. After the whole system worked reasonably well, I constructed test cases and ran them frequently to ensure any modifications did not break a previously working piece. I eventually ended up fixing all of the problems identified and had the entire project working to my satisfaction.

Style Guide

I used OCaIDE's default settings which helped structure some of the code. This includes two space tab indentations. When uncertain, I tried to follow the style provided in the MicroC sample code. Also, since I certainly do not consider myself an expert in O'Caml, I placed many comments throughout the source files to ensure everything was clear and to avoid having to spend time re-reading code that might take me a while to understand.

Project Timeline

An outline of planned target dates is below:

9/26/2012	Initial project proposal
10/31/2012	Language reference manual
10/31/2012	Scanner and parser
12/1/2012	Abstract syntax tree and compilation
12/7/2012	Test cases completed and checked
12/19/2012	Final report complete

Roles and Responsibilities

I worked on the entire project without teammates.

Development Environment Used

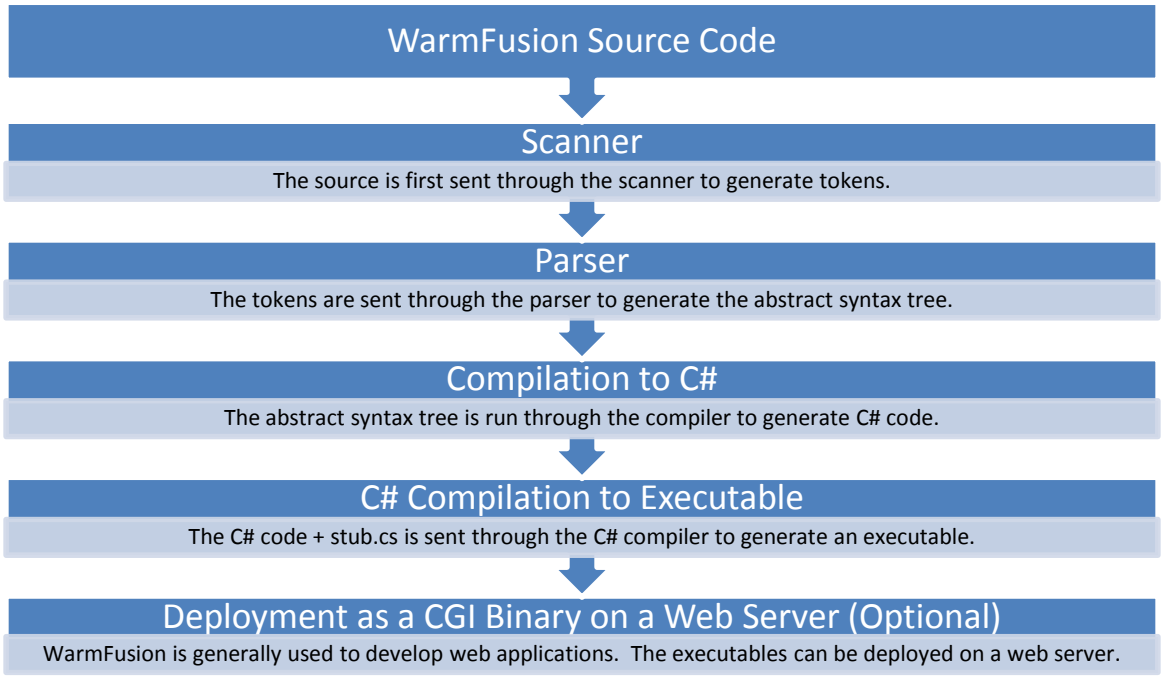
The main project development was done in Windows 7 while utilizing Cygwin for making and compiling. Eclipse with OCaIDE was used when working with the O'Caml pieces of the project. Visual Studio 2012 was used when testing the C# portions of the code. The generated code was also tested in the Mono C# compiler. I also tested all pieces on a machine running Linux to ensure everything would still work properly.

Project Log

An outline of actual milestone completion dates is below:

9/26/2012	Initial project proposal
10/31/2012	Language reference manual
11/18/2012	Scanner and parser mostly finalized
12/16/2012	Abstract syntax tree and compilation mostly finalized
12/17/2012	Test cases completed and checked
12/19/2012	Final report complete

5 Architectural Design



The design first takes a WarmFusion source file and sends it through the scanner/lexer. Lexical analysis is performed on the source file and the sequence of characters is converted into a sequence of tokens. These tokens are defined by regular expressions within the scanner.mll file. Ocamllex converts the .mll file into O'Caml code. Some checking is done at this level to reject invalid programs.

The list of tokens is then passed to the parser. The parser generates an abstract syntax tree (AST) from these tokens. More checking is done at this point and errors may be raised if the input source file does not follow the defined rules. This layer is defined with within the parser.mly and ast.ml files. Ocamlyacc converts the .mly file into O'Caml code.

The abstract syntax tree is then passed to the compiler. At this point, the compiler takes the abstract syntax tree, does appropriate semantic checking and generates C# code. At this stage, the additional checking helps detect errors such as duplicate function definitions, excessively long function/variable names, and other issues that would cause problems.

The generated C# code and a stub file containing pre-written C# code with built-in WarmFusion functions are both concatenated and sent to the C# compiler. This could either be the Mono C# compiler or the Microsoft C# compiler. An executable is then generated. This executable could either be run alone, or deployed as a CGI binary on a web server.

I developed the scanner, parser and compilation to C# layers based on the provided MicroC sample code.

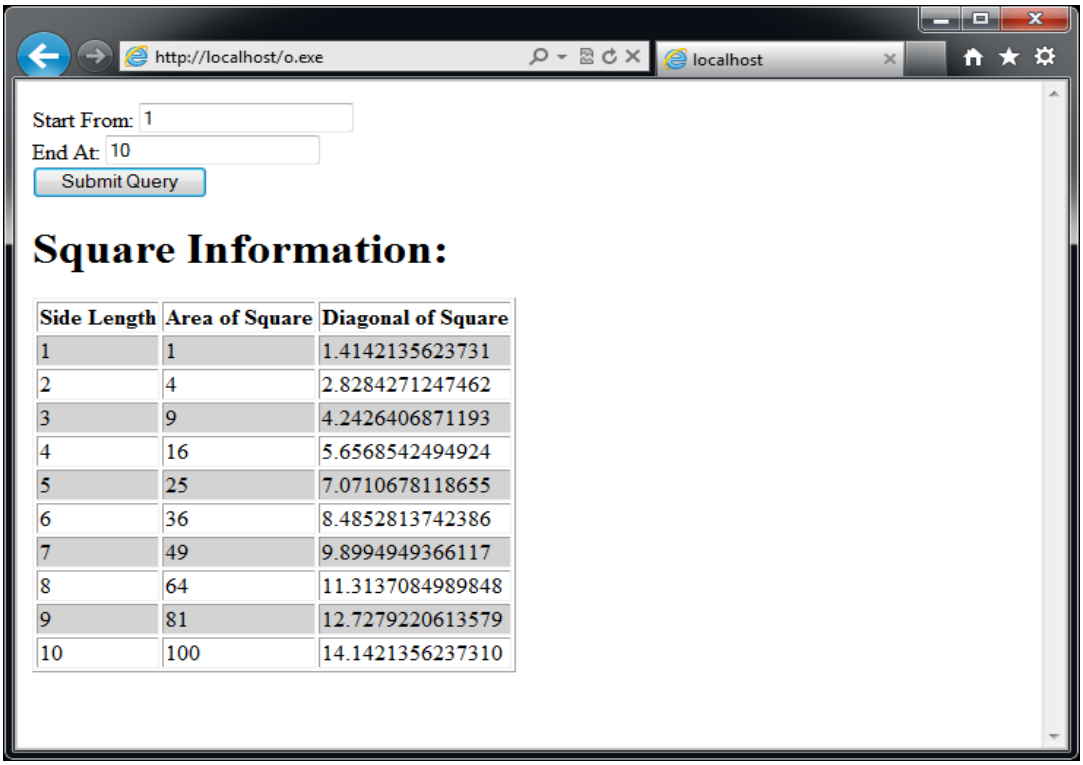
6 Test Plan

The tests were divided into 3 different categories and have 3 separate directories. The tests-as-cgi-bin directory contains test scripts that are meant to be run as CGI binaries on a web server. The tests-basic directory contains basic tests of many features in the language. These can be automatically tested using the testall.sh script. The tests-errors directory contains test cases where errors should occur.

I chose these test cases to try and ensure all of the lines of code in the compiler are executed at least once. After creating the test WarmFusion source files and their expected output files, I re-ran the tests after any changes were made to the compiler to ensure all features were still working as expected.

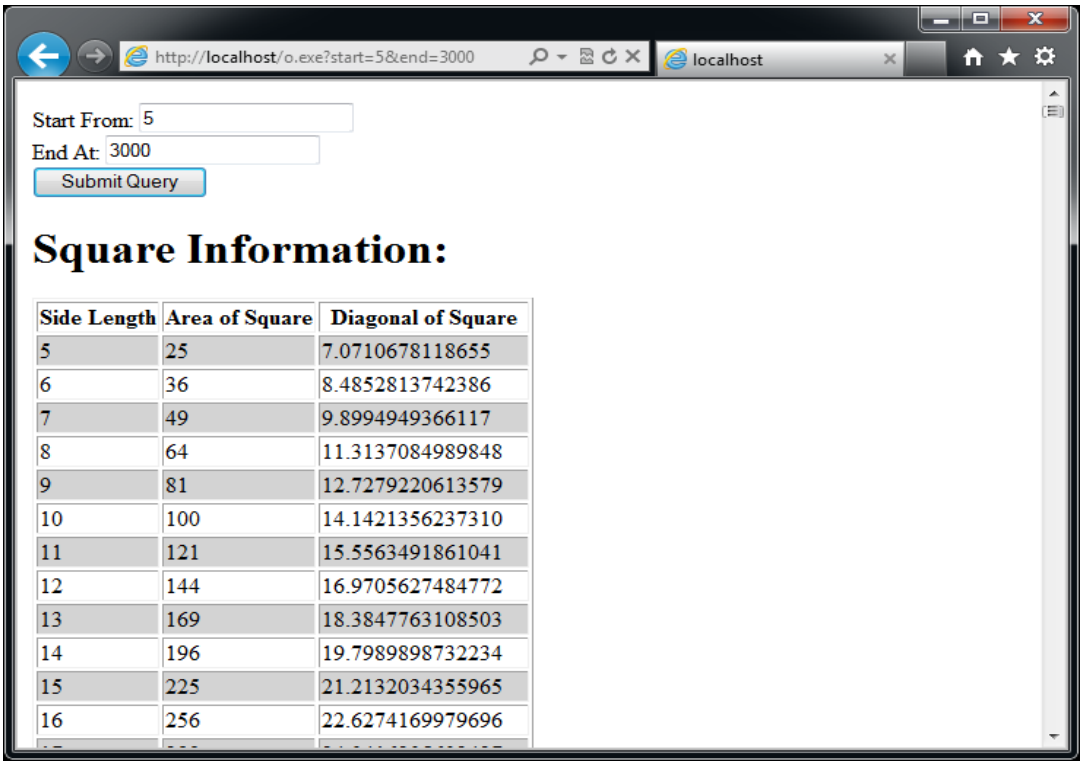
6.1 Square Information Example

Below is an example of the square information WarmFusion application running as a CGI binary on a web server, while being viewed in a web browser. This WarmFusion application allows the user to enter two numbers. For values between those two numbers, it will calculate and output the area of the square, and the diagonal of the square. This example demonstrates the use of many WarmFusion features including reading user input, user-defined functions, built-in functions, mathematical calculations, conditional statements, loop statements, assignment statements, variables with strings, variables with real numbers including integers and non-integers, automatic type conversion from string to number, and the output of standard HTML and variables.

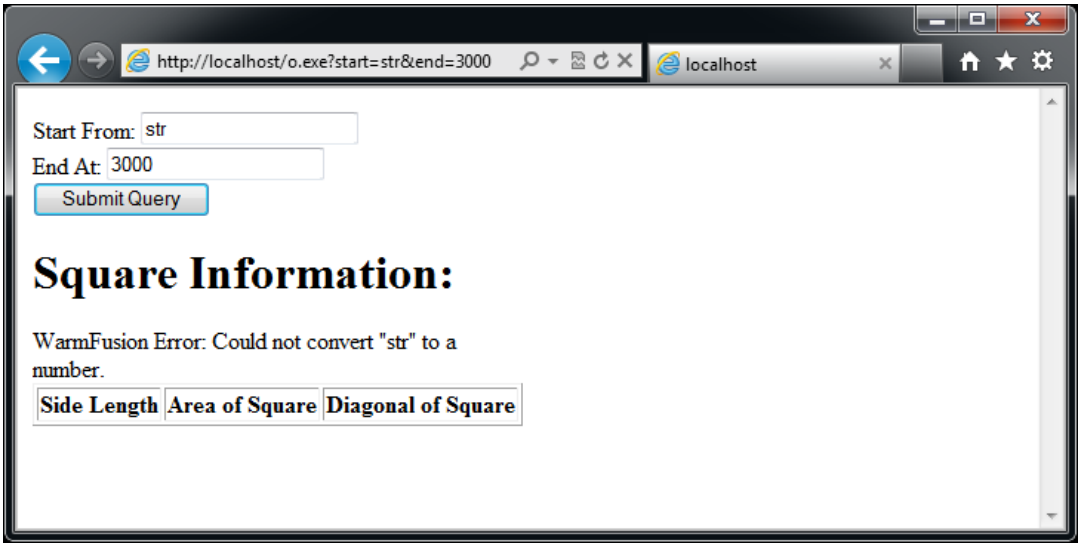


Side Length	Area of Square	Diagonal of Square
1	1	1.4142135623731
2	4	2.8284271247462
3	9	4.2426406871193
4	16	5.6568542494924
5	25	7.0710678118655
6	36	8.4852813742386
7	49	9.8994949366117
8	64	11.3137084989848
9	81	12.7279220613579
10	100	14.1421356237310

The user can change the input values, and re-run the query by pressing the “Submit Query” button. The new query parameters are read in by the application, and a new table is generated. An example is shown below when the user requests the table to be generated from the numbers 5 through 3000.



An error is raised if the user enters a string as input when a number is required.



The WarmFusion code for the Square Information example is located in tests-as-cgi-bin/square_information.wfm and also listed below:

```
<wffunction name="CalculateAreaOfSquare">
        <wfargument name="sideLength" />
        <wfset area = sideLength * sideLength />
        <wfreturn area />
</wffunction>

<wffunction name="CalculateDiagonalOfSquare">
        <wfargument name="sideLength" />
        <!--- Length of a side multiplied by the square root of 2 --->
        <wfset diagonal = sideLength * Sqr(2) />
        <wfreturn diagonal />
</wffunction>

<!--- Check the user's input.  If it's blank, go from 1 - 10. --->
<wfset start = GetQueryParameter('start') />
<wfif start EQ ''>
        <wfset start = 1 />
</wfif>
<wfset end = GetQueryParameter('end') />
<wfif end EQ ''>
        <wfset end = 10 />
</wfif>

<wfoutput>
        <form action="">
                Start From: <input name="start" type="text" maxlength="4" value="#start#" />
                <br />
                End At: <input name="end" type="text" maxlength="4" value="#end#" />
                <br />
                <input type="submit" />
        </form>

        <h1>Square Information:</h1>
        <table border="1">
                <tr>
                        <th>Side Length</th>
                        <th>Area of Square</th>
                        <th>Diagonal of Square</th>
                </tr>
</wfoutput>

<wfloop index="i" from="start" to="end">
        <wfif i % 2 EQ 1>
                <wfset rowBgColor = 'lightgray' />
        <wfelse>
                <wfset rowBgColor = 'white' />
        </wfif>

        <wfset area = CalculateAreaOfSquare(i) />
        <wfset diagonal = CalculateDiagonalOfSquare(i) />

        <wfoutput>
                <tr bgcolor="#rowBgColor#">
                        <td>#i#</td>
                        <td>#area#</td>
                        <td>#diagonal#</td>
                </tr>
        </wfoutput>
</wfloop>

<wfoutput>
        </table>
</wfoutput>
```

Below is the generated C# code that would be produced. Before being sent through the C# compiler, the file stub.cs would be prepended to the generated code below. The stub.cs code is included in the Appendix.

```
public class Program
{
    public static string udv_rowBgColor = "";
    public static string udv_i = "";
    public static string udv_end = "";
    public static string udv_start = "";
    public static string udv_diagonal = "";
    public static string udv_sideLength = "";
    public static string udv_area = "";
    public static string udf_CalculateAreaOfSquare(string udv_sideLength)
    {
        udv_area = WF.Mult(udv_sideLength, udv_sideLength);
        return udv_area;

        return "";
    }

    public static string udf_CalculateDiagonalOfSquare(string udv_sideLength)
    {
        udv_diagonal = WF.Mult(udv_sideLength, WF.Sqr("2"));
        return udv_diagonal;

        return "";
    }

    public static void Main()
    {

        try
        {

            Console.WriteLine(@"Content-type: text/html

");


            udv_start = WF.GetQueryParameter("start");
            if (WF.ConvertStringToBool(WF.Equal(udv_start, "")))
            {
                udv_start = "1";
            }
            else
            {
            }
            udv_end = WF.GetQueryParameter("end");
            if (WF.ConvertStringToBool(WF.Equal(udv_end, "")))
            {
                udv_end = "10";
            }
            else
            {
            }
            Console.Write(@"
        <form action="""">
                Start From: <input name=""start"" type=""text"" maxlength=""4"" value=""");
            Console.Write(udv_start);
            Console.Write(@""" />
                <br />
                End At: <input name=""end"" type=""text"" maxlength=""4"" value=""");
            Console.Write(udv_end);
            Console.Write(@""" />
                <br />
                <input type=""submit"" />
        </form>
```

```
        <h1>Square Information:</h1>
        <table border=""1"">
                <tr>
                        <th>Side Length</th>
                        <th>Area of Square</th>
                        <th>Diagonal of Square</th>
                </tr>
");
            for (udv_i = (udv_start); WF.ConvertStringToBool(WF.Leq(udv_i, udv_end)); udv_i =
WF.Add(udv_i, "1"))
            {
                if (WF.ConvertStringToBool(WF.Equal(WF.Modulus(udv_i, "2"), "1")))
                {
                    udv_rowBgColor = "lightgray";
                }
                else
                {
                    udv_rowBgColor = "white";
                }
                udv_area = udf_CalculateAreaOfSquare(udv_i);
                udv_diagonal = udf_CalculateDiagonalOfSquare(udv_i);
                Console.Write(@"
                 <tr bgcolor=""");
                Console.Write(udv_rowBgColor);
                Console.Write(@""">
                        <td>");
                Console.Write(udv_i);
                Console.Write(@"</td>
                        <td>");
                Console.Write(udv_area);
                Console.Write(@"</td>
                        <td>");
                Console.Write(udv_diagonal);
                Console.Write(@"</td>
                 </tr>
        ");
            }
            Console.Write(@"
        </table>
");
        }
        catch (Exception e)
        {

            Console.WriteLine("WarmFusion Error: " + e.Message);

        }

    }

}
```

6.2 List and Operator Example

Although WarmFusion does not have arrays, it does have lists. A list is basically a comma-delimited string (e.g. 'a,b,c,d'). WarmFusion has several built-in functions for working with lists. The code below performs an operation (e.g. 1 + 2) and appends the result to the end of the list that has the variable name "val". After all of the values are appended to the list, the length of the list is retrieved, and all of the items of the list are looped through. The result of each operation is printed out.

This example can be run from a terminal. The WarmFusion code for the List and Operator example is located in tests-basic/test-ops1.wfm and also listed below:

```
<wfset val = ListAppend(val, 1 + 2) />
<wfset val = ListAppend(val, 1 - 2) />
<wfset val = ListAppend(val, 1 * 2) />
<wfset val = ListAppend(val, 100 / 2) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 EQ 2) />
<wfset val = ListAppend(val, 1 EQ 1) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 NEQ 2) />
<wfset val = ListAppend(val, 1 NEQ 1) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 LT 2) />
<wfset val = ListAppend(val, 2 LT 1) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 LTE 2) />
<wfset val = ListAppend(val, 1 LTE 1) />
<wfset val = ListAppend(val, 2 LTE 1) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 GT 2) />
<wfset val = ListAppend(val, 2 GT 1) />
<wfset val = ListAppend(val, 99) />
<wfset val = ListAppend(val, 1 GTE 2) />
<wfset val = ListAppend(val, 1 GTE 1) />
<wfset val = ListAppend(val, 2 GTE 1) />
<wfset totalItems = ListLen(val) />

<wfloop index="i" from="1" to="totalItems">
        <wfset curItem = ListGetAt(val, i) />
        <wfoutput>
                #curItem#
        </wfoutput>
</wfloop>
```

Below is the generated C# code that would be produced. Before being sent through the C# compiler, the file stub.cs would be prepended to the generated code below. The stub.cs code is included in the Appendix.

```
public class Program
{
    public static string udv_totalItems = "";
    public static string udv_i = "";
    public static string udv_curItem = "";
    public static string udv_val = "";
    public static void Main()
    {
        try
        {
            Console.WriteLine(@"Content-type: text/html

");
            udv_val = WF.ListAppend(udv_val, WF.Add("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Sub("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Mult("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Div("100", "2"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Equal("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Equal("1", "1"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Neq("1", "2"));
```

```
            udv_val = WF.ListAppend(udv_val, WF.Neq("1", "1"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Less("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Less("2", "1"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Leq("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Leq("1", "1"));
            udv_val = WF.ListAppend(udv_val, WF.Leq("2", "1"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Greater("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Greater("2", "1"));
            udv_val = WF.ListAppend(udv_val, "99");
            udv_val = WF.ListAppend(udv_val, WF.Geq("1", "2"));
            udv_val = WF.ListAppend(udv_val, WF.Geq("1", "1"));
            udv_val = WF.ListAppend(udv_val, WF.Geq("2", "1"));
            udv_totalItems = WF.ListLen(udv_val);
            for (udv_i = ("1"); WF.ConvertStringToBool(WF.Leq(udv_i, udv_totalItems)); udv_i =
WF.Add(udv_i, "1"))
            {
                udv_curItem = WF.ListGetAt(udv_val, udv_i);
                Console.Write(@"
                ");
                Console.Write(udv_curItem);
                Console.Write(@"
        ");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("WarmFusion Error: " + e.Message);
        }
    }
}
```

The following output is produced when the executable is run at the command line:

```
Content-type: text/html


                3

                -1

                2

                50

                99

                0

                1

                99

                1

                0

                99

                1

                0
```

```
99
1
1
0
99
0
1
99
0
1
1
```

7. Lessons Learned

I found that creating a compiler for a language with dynamic/loose typing requires a lot of work. WarmFusion automatically converts between numbers and strings. This required a lot of work to ensure the appropriate conversions were always made.

I also found that O'Caml takes a while to learn. Although the project source doesn't have many lines of code, it did take a lot of time to figure out how to do things that I could have easily done in other languages. Not having teammates to consult made this even more challenging. If I did have teammates to consult, it's likely at least one other person in the team would know the solution to simple problems I ran into in O'Caml that I had to spend a lot of time to figure out.

I thought that this project would be a long and tedious process. Although I did have my share of issues with O'Caml, I actually found the project to be rather exciting and satisfying. I am actually quite impressed with the final product. I think it actually works well enough that one could reasonably build a useful real-world web application using it. I am considering continuing to work on WarmFusion in my spare time, but perhaps writing it in Java or C# instead of O'Caml and instead compiling to assembly language.

8. Appendix

8.1 scanner.mll

```
{
        open Parser;;
        let strBuffer = Buffer.create 100;;
        let strOutput = Buffer.create 1000;;
}

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "<!---"     { comment 0 lexbuf }           (* Comments *)
| '"'      { Buffer.clear strBuffer; STRING(string lexbuf) }
| "<wfoutput>"      { Buffer.clear strOutput; RAWOUTPUT(wfoutput lexbuf) }
| '"'      { QUOTE }
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULUS }
| '&'      { CONCATENATE }
| '='      { ASSIGN }
| "NOT"     { NOT }
| "EQ"     { EQ }
| "NEQ"     { NEQ }
| "LT"      { LT }
| "LTE"     { LEQ }
| "GT"      { GT }
| "GTE"     { GEQ }
| "AND"     { AND }
| "OR"     { OR }
| "<wfset"     { SET }
| "<wffunction"     { FUNCTION }
| "<wfargument"     { ARGUMENT }
| "</wffunction>"     { FUNCTIONENDINGTAG }
| "<wfif"     { IF }
| "<wfelse>"   { ELSE }
| "</wfif>"     { IFENDINGTAG }
| "<wfloop" { LOOP }
| "</wfloop>" { LOOPENDINGTAG }
| "name="     { PARAMETERNAME }
| "condition="     { PARAMETERCONDITION }
| "index="     { PARAMETERINDEX }
| "from="     { PARAMETERFROM }
| "to="     { PARAMETERTO }
| "<wfreturn" { RETURN }
| ">"     { TAGCLOSE }
| "/>"     { ENDINGTAG }
| ['0'-'9']+ as lxm { LITERAL(lxm) }
| ['0'-'9']+'.'['0'-'9']+ as lxm { LITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

(* Allow comments inside of comments - I wish every language allowed this. *)
and comment inner = parse
        "<!---" { comment (inner + 1) lexbuf }
| "--->"        {
                                                if (inner = 0) then
                                                        token lexbuf
                                                else
```

```
                                                    comment (inner - 1) lexbuf
                                            }
| _    { comment (inner) lexbuf }

and string = parse
                "'"                                 { Buffer.contents strBuffer }
        | "''" { Buffer.add_string strBuffer "'"; string lexbuf }
        | _ as char { Buffer.add_char strBuffer char; string lexbuf }

(* Belive me, it was a lot of work figuring out how to make this tokenize & parse the content of wfoutput
tags properly *)
and wfoutput = parse
        "</wfoutput>"           { Buffer.contents strOutput }
        | _ as char { Buffer.add_char strOutput char; wfoutput lexbuf }

and wfoutput_scanner = parse
        "##" { OUTPUT("#") }
        |       "#" ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* "#" as str { OUTPUTVARIABLE(String.sub
str 1 (String.length str - 2)) }
        | [^'#']+ as str { OUTPUT(str) }
        | eof { EOF }
```

8.2 parser.mly

```
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA QUOTE
%token PLUS MINUS TIMES DIVIDE MODULUS CONCATENATE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token AND OR NOT
%token RETURN
%token SET FUNCTION ARGUMENT FUNCTIONENDINGTAG
%token IF ELSE IFENDINGTAG
%token FOR LOOP LOOPENDINGTAG
%token PARAMETERNAME PARAMETERCONDITION PARAMETERINDEX PARAMETERFROM PARAMETERTO
%token TAGCLOSE ENDINGTAG
%token <string> LITERAL
%token <string> ID
%token <string> STRING
%token <string> RAWOUTPUT
%token <string> OUTPUT
%token <string> OUTPUTVARIABLE
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left AND OR
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULUS CONCATENATE
%right NOT
%nonassoc NEGATIVE

%start program
%type <Ast.program> program

%start wfoutput_parser
%type <Ast.wfoutput_parser> wfoutput_parser

%%

program:
   /* nothing */ { [], [] }
        /* Not all statements have to be inside functions */
```

```
 | program stmt { ($2 :: fst $1), snd $1 }
 | program fdecl        { fst $1, ($2 :: snd $1) }

fdecl:
FUNCTION PARAMETERNAME QUOTE ID QUOTE TAGCLOSE arg_list stmt_list FUNCTIONENDINGTAG
     { { fname = $4;
         formals = $7;
         body = List.rev $8 } }

arg_list:
    /* nothing */    { [] }
  | arg_list arg { $2 :: $1 }

arg:
   ARGUMENT PARAMETERNAME QUOTE ID QUOTE ENDINGTAG { $4 }


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

stmt:
        SET expr ENDINGTAG { Expr($2) }
  | SET ID ASSIGN expr ENDINGTAG { Assign($2, $4) }
        | SET ID ASSIGN QUOTE expr QUOTE ENDINGTAG { Assign($2, $5) }
  | RETURN expr ENDINGTAG { Return($2) }
  | IF expr TAGCLOSE stmt_list %prec NOELSE IFENDINGTAG { If($2, Block(List.rev $4), Block([])) }
  | IF expr TAGCLOSE stmt_list ELSE stmt_list IFENDINGTAG    { If($2, Block(List.rev $4), Block(List.rev
$6)) }
        | LOOP PARAMETERINDEX QUOTE ID QUOTE PARAMETERFROM QUOTE expr QUOTE PARAMETERTO QUOTE expr QUOTE
TAGCLOSE stmt_list LOOPENDINGTAG { For($4, $8, $12, Block(List.rev $15)) }
  | LOOP PARAMETERCONDITION QUOTE expr QUOTE TAGCLOSE stmt_list LOOPENDINGTAG { While($4, Block(List.rev
$7)) }
        | RAWOUTPUT { RawOutput($1) }


expr:
    LITERAL          { Literal($1) }
        | STRING                  { String($1) }
        | MINUS expr %prec NEGATIVE { Negative($2) }
  | ID               { Id($1) }
        | NOT expr { Negate($2) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
        | expr MODULUS expr { Binop($1, Modulus,   $3) }
        | expr CONCATENATE expr { Binop($1, Concatenate,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
        | expr AND    expr { Binop($1, And,   $3) }
        | expr OR    expr { Binop($1, Or,   $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }

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

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

wfoutput_parser:
               { [] }
```

```
        | wfoutput_parser wfoutput_token { $2 :: $1 }

wfoutput_token:
                OUTPUT { Output($1) }
        | OUTPUTVARIABLE { OutputVariable($1) }
```

8.3 ast.ml

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

type expr =
    Literal of string
        | String of string
  | Id of string
        | Negative of expr
        | Negate of expr
  | Binop of expr * op * expr
  | Call of string * expr list
  | Noexpr

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

type wfoutput =
        | Output of string
        | OutputVariable of string

type wfoutput_parser = wfoutput list

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

type program = stmt list * func_decl list
```

8.4 warmfusion.ml

```
type action = Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-c", Compile) ]
  else Compile in
    let lexbuf = Lexing.from_channel stdin in
    let program = Parser.program Scanner.token lexbuf in
    match action with
        Compile -> print_string (Compile.translate program)
```

8.5 compile.ml

```
open Ast
open Str

(* Translate a program in AST form into C#.  Throw an exception if something is wrong, e.g. a reference to
an unknown function *)
let translate (statements, functions) =
	(* A built-in function lookup hash table so we can check if a function is defined and has the
correct number of arguments *)
	let builtin_functions = Hashtbl.create 20 in
		Hashtbl.replace builtin_functions "GetTickCount" 0;
		Hashtbl.replace builtin_functions "Sqr" 1;
		Hashtbl.replace builtin_functions "LCase" 1;
		Hashtbl.replace builtin_functions "UCase" 1;
		Hashtbl.replace builtin_functions "Len" 1;
		Hashtbl.replace builtin_functions "Left" 2;
		Hashtbl.replace builtin_functions "Mid" 3;
		Hashtbl.replace builtin_functions "Right" 2;
		Hashtbl.replace builtin_functions "ListLen" 1;
		Hashtbl.replace builtin_functions "ListGetAt" 2;
		Hashtbl.replace builtin_functions "ListSetAt" 3;
		Hashtbl.replace builtin_functions "ListAppend" 2;
		Hashtbl.replace builtin_functions "ListFind" 2;
		Hashtbl.replace builtin_functions "Rand" 0;
		Hashtbl.replace builtin_functions "Now" 0;
		Hashtbl.replace builtin_functions "HTMLEditFormat" 1;
		Hashtbl.replace builtin_functions "URLEncodedFormat" 1;
		Hashtbl.replace builtin_functions "GetQueryParameter" 1;

	(* A hash table of user defined functions and the number of arguments for each function *)
	let user_defined_functions = Hashtbl.create 100 in
		(* Check all the functions *)
		List.iter (function f ->
			(* Don't allow the user to define multiple functions with the same name *)
			if Hashtbl.mem user_defined_functions f.fname then
				raise (Failure ("Duplicate definition of function " ^ f.fname))
			(* Make sure function names are 100 characters or less *)
			else if String.length f.fname > 100 then
				raise (Failure ("Function name too long: " ^ f.fname))
			(* If there are no problems with the function, add it to the hash table of
hashTbl[functionname] = (number of args) *)
			else
				Hashtbl.replace user_defined_functions f.fname (List.length f.formals);

			(* Make sure there are no more than 100 arguments in a function *)
			if (List.length f.formals) > 100 then
				raise (Failure ("Excessive number of arguments in function " ^ f.fname));

			(* Check all the arguments within each function *)
			List.iter (function argname ->
				(* Check if multiple arguments with the same name within a function  *)
				if (List.length (List.find_all (function x -> x = argname) f.formals)) >
1 then
					raise (Failure ("The argument " ^ argname ^ " cannot occur
multiple times in " ^ f.fname))
				(* Make sure argument names are 100 characters or less *)
				else if String.length argname > 100 then
					raise (Failure ("Argument name too long: " ^ argname))
				) f.formals
		) functions;

	(* wfreturn statements can only be placed within functions *)
	let rec check_statements = function
		Block s1 -> ignore(List.map check_statements s1)
		| Return e -> raise (Failure ("wfreturn can only be used within functions"))
		| _ -> ignore() in
	let _ = List.map check_statements statements in
```

```
        (* Keep track of global variables used - global variables are all variables except function
arguments *)
        let global_vars = Hashtbl.create 100 in

        (*************************************************************************)
  (* Converts a binary operator to the equivalent compiler method *)
  let operator_to_cs = function
                | Add -> "Add"
                | Sub -> "Sub"
                | Mult -> "Mult"
                | Div -> "Div"
                | Modulus -> "Modulus"
                | Concatenate -> "Concatenate"
                | Equal -> "Equal"
                | Neq -> "Neq"
                | Less -> "Less"
                | Leq -> "Leq"
                | Greater -> "Greater"
                | Geq -> "Geq"
                | And -> "And"
                | Or -> "Or"

        (*************************************************************************)
  (* Translate an expression *)
  in let rec expr = function
        (* For numbers - A number can include a decimal portion *)
                (* WarmFusion has dynamic/loose typing so internally numbers are treated as strings and
conversions are done when necessary *)
        Literal i ->
                        (* Simple check for numbers - the string cannot be longer than 25 characters.  We
can actually handle numbers that big. *)
                        if String.length i > 25 then
                                raise (Failure ("Numeric constant too long: " ^ i))
                        else
                                "\"" ^ i ^ "\""
                (* For strings *)
        | String s -> "\"" ^ s ^ "\""
                (* To get the negative value of a number *)
        | Negative e -> "WF.Negative(" ^ (expr e) ^ ")"
                (* To negate a boolean expression *)
        | Negate e -> "WF.Negate(" ^ (expr e) ^ ")"
        (* For variables - All variables are internally prefixed with udv_ to prevent naming collisions *)
        | Id s -> Hashtbl.replace global_vars s 0;
                                                        "udv_" ^ s
        (* Binary operators *)
    | Binop (e1, op, e2) -> "WF." ^ (operator_to_cs op) ^ "(" ^ (expr e1) ^ ", " ^ (expr e2) ^ ")"
        (* Call a function *)
        | Call (fname, actuals) ->
                                (* Check if the function is a built-in function *)
                        if Hashtbl.mem builtin_functions fname then
                                        (* Number of arguments must match *)
                                        if (List.length actuals) != (Hashtbl.find builtin_functions
fname) then
                                                raise (Failure ("Call of function " ^ fname ^ " does not
have the correct number of arguments."))
                                        else
                                        "WF." ^ fname ^ "(" ^ String.concat ", " (List.map expr actuals)
^ ")"
                                (* Check if the function is a user-defined function *)
                                else if Hashtbl.mem user_defined_functions fname then
                                        (* Number of arguments must match *)
                                        if (List.length actuals) != (Hashtbl.find user_defined_functions
fname) then
                                                raise (Failure ("Call of function " ^ fname ^ " does not
have the correct number of arguments."))
                                        else
                                                (* All user-defined functions are internally prefixed
with udf_ to prevent naming collisions *)
```

```
                                                "udf_" ^ fname ^ "(" ^ String.concat ", " (List.map expr
actuals) ^ ")"
                                (* If the function is not defined *)
                        else
                                raise (Failure ("Undefined function " ^ fname))
        (* *No expression *)
        | Noexpr -> ""

        (*****************************************************************************)
  (* Handle wfoutput with variable names inside *)
  in let rec wfoutput = function
        | Output s -> "Console.Write(@\"" ^ (Str.global_replace (Str.regexp_string "\"") "\"\"" s) ^
"\");\n"
        | OutputVariable v ->
                        Hashtbl.replace global_vars v 0;
                        "Console.Write(udv_" ^ v ^ ");\n"

        (*****************************************************************************)
  (* Translate a statement *)
  in let rec stmt = function
        (* For a block of statements, do one statement after the other *)
        Block sl     ->  "\n{\n" ^ String.concat "" (List.map stmt sl) ^ "}\n"
                (* wfset *)
        | Assign (v, e) ->
                        Hashtbl.replace global_vars v 0;
                        "udv_" ^ v ^ " = " ^ (expr e) ^ ";\n"
                (* End of line *)
    | Expr e       -> expr e ^ ";\n"
                (* wfreturn *)
        | Return e -> "return " ^ (expr e) ^ ";\n"
                (* wfif/wfelse *)
        | If (p, t, f) -> "if (WF.ConvertStringToBool(" ^ (expr p) ^ "))" ^ (stmt t) ^ "else" ^ (stmt f)
                (* wfloop with index *)
        | For (v, e2, e3, b) ->
                        Hashtbl.replace global_vars v 0;
                        "for (udv_" ^ v ^ " = (" ^ (expr e2) ^ "); " ^
                        "WF.ConvertStringToBool(WF.Leq(udv_" ^ v ^ ", " ^ (expr e3) ^ ")); " ^
                        "udv_" ^ v ^ " = WF.Add(udv_" ^ v ^ ", \"1\"))" ^
                        (stmt b)
                (* wfloop with condition *)
        | While (e, b) -> "while (WF.ConvertStringToBool(" ^ (expr e) ^ ")) " ^ (stmt b)
                (* wfoutput - We need to do separate scanning for stuff inside the wfoutput tag.  This
parses #variable# output *)
        | RawOutput (s) ->      let lexbuf = Lexing.from_string s in
                                                                        let wfoutput_ast
= Parser.wfoutput_parser Scanner.wfoutput_scanner lexbuf in
                                                                        String.concat ""
(List.map wfoutput (List.rev wfoutput_ast))

        (*****************************************************************************)
        (* Translate a function *)
  in let func f =
                (* Make sure the user isn't trying to re-define a system function *)
                if Hashtbl.mem builtin_functions f.fname then
                        raise (Failure ("Redefinition of a built-in system function " ^ f.fname))
                else
        let arguments = List.map (function s -> "string udv_" ^ s) (List.rev f.formals) in
        "public static string udf_" ^ f.fname ^ "(" ^ (String.concat ", " arguments) ^ ") {\n" ^
                (String.concat "" (List.map stmt f.body)) ^ "\nreturn \"\";}\n\n"

        (*****************************************************************************)
  (* Generate code for all of the functions *)
  in
                (* Generate code for all functions and statements that aren't in functions *)
                let strFunctions = String.concat "" (List.map func (List.rev functions)) in
                let strStatements = String.concat "" (List.map stmt (List.rev statements)) in
                (* WarmFusion doesn't need variable declarations.  You can just start using the variables
without declaring them.  This does some processing to prepare for that. *)
                let varDeclarations = Hashtbl.fold (
```

```
                                fun key _ str ->
                                        (* Make sure variable names are 100 characters or less *)
                                        if String.length key > 100 then
                                                raise (Failure ("Variable name too long: " ^ key))
                                        else
                                                str ^ "public static string udv_" ^ key ^ " = \"\";\n"
                        ) global_vars "" in
                "public class Program
                {
                " ^
                varDeclarations ^
                strFunctions ^
        (* Main header *)
        "public static void Main()
        {
                try
                {
                        Console.WriteLine(@\"Content-type: text/html\n\n\");
                " ^
    strStatements ^
                (* Handle runtime errors *)
        "       } catch (Exception e) {
          Console.WriteLine(\"WarmFusion Error: \" + e.Message);
        }
        }
                }"
```

8.6 stub.cs

```
using System;
using System.Collections.Specialized;
using System.Web;

#pragma warning disable 0162

public class WF
{
    private static Random rand = new Random();

    //WarmFusion isn't strongly typed, so we store everything as strings, and convert to a number when
necessary
    public static decimal ConvertToNumber(string s)
    {
        if (s.Length == 0)
            return 0;

        decimal d;
        bool success = decimal.TryParse(s, out d);
        if (!success)
            throw new Exception("Could not convert \"" + s + "\" to a number.");

        return d;
    }

    //The number 0, an empty string or the string "false" all evaluate to false.  Everything else
evaluates to true
    public static bool ConvertStringToBool(string s)
    {
        bool boolValue;
        try
        {
            decimal d = ConvertToNumber(s);
            boolValue = (d != 0);
        }
        catch (Exception)
        {
```

```
            boolValue = (s.ToLower() != "false");
        }

        return boolValue;
    }

    public static string ConvertBoolToString(bool b)
    {
        if (b)
            return "1";
        else
            return "0";
    }

    public static string Negate(string s)
    {
        if (ConvertStringToBool(s))
            return "0";
        else
            return "1";
    }

    public static string Negative(string s)
    {
        return (-ConvertToNumber(s)).ToString();
    }

    public static string Add(string s1, string s2)
    {
        return (ConvertToNumber(s1) + ConvertToNumber(s2)).ToString();
    }

    public static string Sub(string s1, string s2)
    {
        return (ConvertToNumber(s1) - ConvertToNumber(s2)).ToString();
    }

    public static string Mult(string s1, string s2)
    {
        return (ConvertToNumber(s1) * ConvertToNumber(s2)).ToString();
    }

    public static string Div(string s1, string s2)
    {
        return (ConvertToNumber(s1) / ConvertToNumber(s2)).ToString();
    }

    public static string Modulus(string s1, string s2)
    {
        return (ConvertToNumber(s1) % ConvertToNumber(s2)).ToString();
    }

    public static string Concatenate(string s1, string s2)
    {
        return s1 + s2;
    }

    public static string Equal(string s1, string s2)
    {
        bool isEqual;
        try
        {
            //If they're both numbers, do a numeric comparison
            decimal d1 = ConvertToNumber(s1);
            decimal d2 = ConvertToNumber(s2);
            isEqual = (d1 == d2);
        }
        catch (Exception)
        {
```

```
            //Otherwise do a string comparison
            isEqual = (s1 == s2);
        }

        return ConvertBoolToString(isEqual);
    }

    public static string Neq(string s1, string s2)
    {
        return Negate(Equal(s1, s2));
    }

    public static string Less(string s1, string s2)
    {
        return ConvertBoolToString(ConvertToNumber(s1) < ConvertToNumber(s2));
    }

    public static string Leq(string s1, string s2)
    {
        return ConvertBoolToString(ConvertToNumber(s1) <= ConvertToNumber(s2));
    }

    public static string Greater(string s1, string s2)
    {
        return ConvertBoolToString(ConvertToNumber(s1) > ConvertToNumber(s2));
    }

    public static string Geq(string s1, string s2)
    {
        return ConvertBoolToString(ConvertToNumber(s1) >= ConvertToNumber(s2));
    }

    public static string And(string s1, string s2)
    {
        return ConvertBoolToString(ConvertStringToBool(s1) && ConvertStringToBool(s2));
    }

    public static string Or(string s1, string s2)
    {
        return ConvertBoolToString(ConvertStringToBool(s1) || ConvertStringToBool(s2));
    }

    ////////////////////////////////////////////////////////////////////////////////
    // Built-in functions
    ////////////////////////////////////////////////////////////////////////////////

    public static string GetTickCount()
    {
        return Environment.TickCount.ToString();
    }

    public static string Sqr(string s)
    {
        decimal dec = ConvertToNumber(s);
        double dbl = Math.Sqrt((double)dec);
        return dbl.ToString();
    }

    public static string LCase(string s)
    {
        return s.ToLower();
    }

    public static string UCase(string s)
    {
        return s.ToUpper();
    }

    public static string Len(string s)
```

```
    {
        return s.Length.ToString();
    }

    public static string Left(string s, string len)
    {
        try
        {
            return s.Substring(0, (int)ConvertToNumber(len));
        }
        catch (Exception)
        {
            throw new Exception("Invalid len " + len + " in " + s);
        }
    }

    // String Mid function.  In WarmFusion, the index starts at 1 instead of 0.
    public static string Mid(string s, string start, string count)
    {
        try
        {
            return s.Substring((int)ConvertToNumber(start) - 1, (int)ConvertToNumber(count));
        }
        catch (Exception)
        {
            throw new Exception("Invalid start " + start + " or count " + count + " in " + s);
        }
    }

    public static string Right(string s, string len)
    {
        try
        {
            return s.Substring(s.Length - (int)ConvertToNumber(len));
        }
        catch (Exception)
        {
            throw new Exception("Invalid len " + len + " in " + s);
        }
    }

    //List functions below.  A list is basically a comma delimited string.
    //e.g str = "a,b,c,d"
    //In the example above, ListLen(str) is 4; ListGetAt(str, 2) is "b"; ListSetAt(str, 2, 'x') is
"a,x,c,d"; ListAppend(str, "e") is "a,b,c,d,e"; ListFind(str, "d") is 4
    public static string ListLen(string lst)
    {
        string[] splitted = lst.Split(',');
        return splitted.Length.ToString();
    }

    public static string ListGetAt(string lst, string num)
    {
        try
        {
            string[] splitted = lst.Split(',');
            return splitted[(int)ConvertToNumber(num) - 1];
        }
        catch (Exception)
        {
            throw new Exception("Invalid index " + num + " in " + lst);
        }
    }

    public static string ListSetAt(string lst, string num, string str)
    {
        try
        {
            string[] splitted = lst.Split(',');
```

```
            splitted[(int)ConvertToNumber(num) - 1] = str;
            return String.Join(",", splitted);
        }
        catch (Exception)
        {
            throw new Exception("Invalid index " + num + " in " + lst);
        }
    }

    public static string ListAppend(string lst, string str)
    {
        if (lst.Length == 0)
            return str;
        else
            return lst + "," + str;
    }

    //Returns 0 if the string could not be found, otherwise it returns the index.
    public static string ListFind(string lst, string str)
    {
        string[] splitted = lst.Split(',');
        int val = Array.IndexOf(splitted, str);
        return (val + 1).ToString();
    }

    public static string Rand()
    {
        return rand.NextDouble().ToString();
    }

    public static string Now()
    {
        return DateTime.Now.ToString();
    }

    public static string HTMLEditFormat(string str)
    {
        return HttpUtility.HtmlEncode(str);
    }

    public static string URLEncodedFormat(string str)
    {
        return HttpUtility.UrlEncode(str);
    }

    //This get up the URL query parameter if running this program as a CGI binary
    //e.g. http://localhost/warmfusion.exe?var=1234 The value "1234" can be accessed by calling
GetURLParameter("var")
    public static string GetQueryParameter(string name)
    {
        string queryString = System.Environment.GetEnvironmentVariable("QUERY_STRING");
        if (queryString != null)
        {
            NameValueCollection queryParameters = HttpUtility.ParseQueryString(queryString);
            string val = queryParameters[name];
            if (val != null)
                return val;
        }

        return "";
    }
}
```

8.7 warmfusionc.sh

```
#!/bin/sh
```

```
EXT=".wfm"
WARMFUSION="./warmfusion"
runprogram=0
deletefile=0

if [ $# -eq 0 ]
then
        echo "Usage: warmfusionc.sh [-r] [-d] <file.wfm>"
        echo "-d   Delete generated .cs file"
        echo "-r   Run the program after compiling"
        exit 1
fi

#Run the program if -r is specified
while getopts "rd" option
do
        case $option in
        d)
                deletefile=1
                ;;
        r)
                runprogram=1
                ;;
        esac
done
shift `expr $OPTIND - 1`

filename=$(basename "$1")
filenamewithoutext=${filename%$EXT}

#Concatenate the stub.cs with the generated output
cat stub.cs > "$filenamewithoutext.cs"
${WARMFUSION} < $1 >> "$filenamewithoutext.cs"

#Do the compilation using Mono (requires libmono-system-web2.0-cil to be installed)
#Alternatively, if you're using Microsoft's C# compiler, you could do something similar to:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /r:System.Web.dll $filenamewithoutext.cs
mcs /r:System.Web.dll "$filenamewithoutext.cs"

#Allow it to execute
chmod 755 "$filenamewithoutext.exe"

#Delete .cs file
if [ $runprogram -eq 1 ] ; then
        rm $filenamewithoutext.cs
fi

#Run the program
if [ $runprogram -eq 1 ] ; then
        ./$filenamewithoutext.exe
fi
```