BELL

Final Report
Programming Languages and Translators
Professor Stephen Edwards
December 18, 2007

Alicia Bozyk	amb2129@columbia.edu
Yousry ElMallah	yae2103@columbia.edu
Robert Lin	rcl2106@columbia.edu
Carlene Liriano	cl2294@columbia.edu

Table of Contents

Introduction

BELL is a compiled language created for the purposes of generating HTML code. Primarily BELL was conceived in order to provide control flow structures to otherwise static HTML code. In addition, the BELL language allows for the creation and execution of user-defined functions. And finally, BELL is unique because it is able to call certain java-specific code at its IR stage. Notice that BELL was initially heavily inspired by the PHP language. However, because BELL is compiled instead of interpreted, their purposes are actually very different.

The architectural summary of BELL is simple: The language accepts a source file coded in the BELL language, "file_name.bell" and creates the BELL IR file, "belljava.java." This IR file is then compiled through the JVM to create the final "file_name.html."

Notice that BELL is particularly special because it allows the user to call Java, via static methods, at the IR stage. BELL does not have Java's notion of instances of classes (all method calls must be static and return an int or string), but Bell is still powerful nonetheless. The ability to access Java at the IR stage empowers the user with the ability to do many useful things.

Possible use cases include the following: Every time a BELL file is generated, a static Java method could create a separate log file, call a user-defined class, insert into a database, return an int or string, or do anything else that Java can do. Notice that BELL alone is already useful and quite powerful because it handles control-flow-structures to simplify html coding that might otherwise be tedious to write (the repetition of table rows in a table for instance). But BELL is truly special because of its tight-knit connectivity to Java. Because all BELL code is compiled into Java first, it allows for tight integration with any static user-defined Java classes that the user may already wish to use.

Language Tutorial

BELL Functionality

- Comments
- Types – int, string, Boolean
- Declaration of Variables
- Expressions – arithmetic, logical, relational, unary
- Statements – if, else, for, while
- Built in Functions
- User Defined Functions

Useful Tips

- To compile BELL code, call bell.java and specify the input file and the output file
  java bell input.bell output.html
- When compiling BELL code, bell.java is first translated to belljava.java, which is compiled and executed by Java to generate the final output file.
- Comments
  (: this text is commented out :)
- Functions must begin with fxn
  fxn int functionName(type var1, type var2)

Hello World Example

HelloWorld.bell

```
<?
     string h;
?>
<html>
<html>
<head>
<title>Hello World Example</title>
</head>
<body>
<?
     h = "Hello World!";
     print(h);
?>
</body>
</html>
</html>
```

HelloWorld.html

```
<html>
<head>
<title>
Hello World Example
</title>
</head>
<body>
Hello World!
</body>
</html>
```

User Defined Functions

User defined functions can appear in the first BELL block before any HMTL code appears.

MailTo.bell

```
<?
fxn int createMailTo(string email, string name) {
print("<a href=mailto:\"" + email + "\"> " + name + " </a>");
return 1;
}
?>
<html>
<html>
<head>
<title> MailTo Example </title>
</head>
<body>
Send an Email
<?
createMailTo("alicia@gmail.com", "Alicia");
createMailTo("carlene@gmail.com", "Carlene");
createMailTo("robert@gmail.com", "Robert");
createMailTo("yousry@gmail.com", "Yousry");
?>
</body>
</html>
</html>
```

MailTo.html

```
<html>
<head>
<title> MailTo Example </title>
</head>
<body>
Send an Email
<a href=mailto:"alicia@gmail.com"> Alicia </a>
<a href=mailto:"carlene@gmail.com"> Carlene </a>
<a href=mailto:"robert@gmail.com"> Robert </a>
<a href=mailto:"yousry@gmail.com"> Yousry </a>
</body>
```

Language Reference Manual

Lexical Conventions

Comments

Comments are supported in BELL. A comment is introduced by a ":(" and is terminated by a "):". They can be placed anywhere within the (: :) block.

Identifiers

Identifiers are sequences of letters, digits, and/or underscores where the first character can only be a letter or an underscore. Identifiers are case sensitive.

Keywords

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

int	boolean	if
char	string	else
float	array	while
double	NULL	for

Constants

There are several kinds of constants, as follows:

a. Integer constants
An integer constant is a sequence of digits. The associated keyword is int.

String Literals

A string is a sequence of characters surrounded by double quotes " " ". Strings may consist of any characters except for the double quote, which must be escaped using a back slash (\"). Strings are associated with the keyword string.

Other Tokens

There are a set of single characters that must be used correctly according to BELL's syntax. They include:

```
{    }    (    )    +    -
*    /    $    :    ;    _
!    %    |    \    ,    .
```

These are also a set of character pairs which must be used correctly according to BELL's syntax. They include:

```
&&   ||   ==   <=   >=   !=
```

Data Types

Boolean

A Boolean expresses a truth value. It can be either TRUE or FALSE.

Int

An int is a number of the set Z = {…, - 2, - 1, 0, 1, 2, …}.

String

A string is a series of characters. In BELL, a character is the same as a byte, that is, there are exactly 256 different characters possible.

Functions

User-Defined Functions

A BELL function can be defined with the following syntax:

```
Fxn _type function_name ( parameter_type parameter_name, ... ) {
     (: Any BELL code can be inserted here :)
     return value;
}
```

BELL allows users to create their own functions to implement any functionally desired by the user. These functions can be defined using the syntax above.

Internal Functions

BELL comes standard with many functions and constructs, providing a lot of flexibility to the developer. Every function will have a manual page that describes function parameters, behaviors, and return values. The following are some examples of some basic BELL functions:

- Print() (prints a string)
- D.print() (prints a string)
- Print_r() (prints the contents of an array)
- Unique() (determines if the values in a data structure are unique)
- Substr() (returns a substring given a string)
- Count() (returns the # of elements in any data structure)
- Include()
- Import() (imports data from an external source)
- Popup()
- Isset() (checks if a values is set)

Expressions

Function Call

A function call in BELL takes the form:

```
Function_identifier(arg1, arg2, …);
```

The function call contains the function name followed by parentheses that may or may not include function arguments, depending on the function definition. When arguments are passed, as in the above example, they are passed as a comma - separated list of expressions.

Unary Expressions

Unary expressions in BELL are those that employ the following operators i: !, ++, - -, and (type). These operators are described below.

```
$variable = expression++
```

Post-fix incrementation operators in BELL assign a value to the variable and then increments the expression. In the above example, the value of $variable would be expression. After the assignment is made, expression is incremented by 1.

```
$variable = expression--
```

Post-fix decrementation operators in BELL assign a value to the variable and then decrements the expression. In the above example, the value of $variable would be expression. After the assignment is made, expression is incremented by 1.

```
!expression
```

The above example is a logical negation expression. Expression must either return a 0 or 1. If expression has a value of 0, the result is 1. If expression has a value of 1, the result is a 0.

Relational/Comparison Expressions

The following table summarizes the relational operators supported in BELL.

Example Expression	Result
$a < $b	**TRUE** if $a is strictly less than $b.
$a > $b	**TRUE** if $a is strictly greater than $b.
$a <= $b	**TRUE** if $a is less than or equal $b.
$a >= $b	**TRUE** if $a is greater than or equal to $b.
$a == $b	**TRUE** if $a is equal to $b.
$a != $b	**TRUE** if $a is not equal to $b.

Relational operators group from left - to - right. Relational expressions that evaluate to true, return a result of type int with a value of 1. Similarly, those that evaluate to false, return a result also of type int with a value of 0.

The equality operators == and != follow the same rules as the relational operators. However, they have lower precedence than the other relational operators.

Logical

The table below summarizes the functionality of the logical operators in BELL.

Example Expression	Result
$a && $b	**TRUE, False** otherwise.
$a || $b	**TRUE** if either $a or $b is **TRUE**.

In BELL, the logical - and operator takes precedence over the logical - or operator. These logical expressions are typically applied to other logical expressions or relational/comparison expressions. They return either true or false, where true is denoted by 1, and false by 0.

Assignment

Assignment expressions in BELL group right-to-left. Assignment expressions take the form of: Identifier -assignment operator-expression;

Assignment operators may be any of those listed in the Operators section. After this assignment expression executes, identifier will contain the value returned by expression.

For arithmetic assignment expressions, either both operands can be of the same type or the operand on the right side of the equation is converted to the type of the operand on the left.

Statements

Statements in BELL are executed in sequence. A statement in BELL is simply a line of code representing an instruction to the compiler. Much like some of the more popular programming languages that exist today, statements in BELL must be terminated with a ';'. Also, because BELL is geared at web-based functionality, a standard BELL program will have html code interspersed throughout. In order to tell the compiler when to start interpreting the code and when to skip over HTML, statements must be surrounded by opening and closing brackets similar to those found in PHP. There are several types of statements, including assignment, function call, and return statements. BELL also features iterative and selection statements made up of the standard control-flow structures.

Opening/Closing Tags

Opening and closing tags will delimit where BELL code begins and where BELL code ends and is followed by HTML code. A standard BELL enclosed statement resembles the following:

```
<?
(: BELL code goes here; :)
?>
<p> HTML stuff can follow </p>
<?
(: More BELL stuff to come :)
?>
```

Assignment Statements

Assignment statements are simply Assignment expressions terminated by a semi - colon and enclosed in BELL brackets. Please se the Assignment Expression example in the previous section.

Selection Statements

Selection statements are simply conditionals (if/else) that select whether or not a particular statement gets executed. The if - statement may take two forms:

(1)
```
if (expr) {
      statement
}
```

- or -

(2)
```
if (expr) {
      Statement
}
else {
      Statement
}
```

In example 1, if the expression evaluates to true then the statement will get executed, otherwise the compiler will ignore it and move on with the rest of the program.

If-statements may be followed by an else statement, as shown in example 2. Here if the expression evaluates to true the statement directly following the if is executed. Otherwise, the statement following the else is executing and then the program returns to a normal flow of control.

It is important to note that an "else" will always be matched to the nearest if - statement to handle the "dangling else problem."

Iterative Statements

The simplest iterative statement in BELL is the while loop. It functions pretty much the same as it does in other common programming languages such as C and Java. A while loop in BELL looks like the following:

```
while(Boolean expr) {
      Statement
}
```

The while loop executes until the Boolean expression is no longer true. Multiple statements can be inside a while loop by enclosing in the scope of the loop (delimited by brackets).

Note also, like any other block of BELL code, iterative statements must be enclosed in the BELL tags.

Another control structure is the for loop

```
for (expr1; expr2; expr3) {
      statement
}
```

In this example, expr1 is an initialization for the loop. Expr2 is a testing condition for the loop to

determine whether or not the loop should terminate. This condition is checked before each iteration. Finally, expr3 alters the value of expr1 to eventually cause the loop to reach a termination condition. A programmer may drop any or all of the expressions in the for loop, i.e. a programmer may use a for loop of the form:

Jump Statement

BELL only supports one kind of jump statement - the return statement. The return statement will unconditionally transfer control flow. Typically, return statements will appear at the end of a function definition, so that after a method has executed, the program will to return to its caller along with the value of the expression it is returning. A return - statement should be used when a function returns some value after its completion to its caller. Otherwise, it should be left out.

A return - statement will take the form:

```
return expr;
```

Bell Specific Functions

We wanted to create a language that would provide a lot of useful functionality in terms of generating useful HTML code for programmers. However, due to time constraints, the only functions we were able to set up were one that enclosed a string in title tags and one that executes a java function that is allowed in the BELL command package.

Title

The Title(String arg) function accepts one parameter and is the only Bell-specific function at the moment, but demonstrates the program's ability to wrap the user's input text in the title html tags, <title></title>.

callJava

The callJava(String arg) function is the workhorse of the BELL program that allows the user to bring in foreign Java code that is required to both be static and in the bell.command package. Also, BELL has no notion of instances and objects, so and foreign imported static methods must return either an int or String.

Project Plan

Processes

There were many processes involved in the creation of BELL. These processes included planning our language, deciding its specifications, development, and testing. We met weekly in order to keep the project moving along, and the time we spent was invaluable. The next sections detail our processes further.

Planning

One of the ideas that most members in the group liked early on was to make a language that would make it easier for programmers to create web pages based on HTML. Our original idea was to make a widget generation language. We did poor planning, as none of our team members were very familiar with widgets and we did not know how it would work in the end. As a result, we decided to change our project to make BELL a language that would imitate a subset of PHP, with some extra functionality. The goal was to make it simpler to generate HTML by providing programmers with the tools to generate HTML code quickly, which was at the heart of our original proposal. This taught us that we had to clearly understand and think our ideas through completely.

Specification

We planned out the specifications for BELL in the first Language Reference Manual. We had to decide what language features we would be able to provide. We included most of the features basic to all languages, and defined BELL's lexical tokens and grammar. Our specifications changed as the project developed. BELL provides some built in functions, but is powerful because it allows a user to define his own functions.

Development

Development happened in multiple stages. First, the lexer and parser was developed and tested to ensure that it was able to parse all of the code that met our BELL specifications, and put into an abstract syntax tree properly. The tree walker was developed separately, and when the three were completed, they did not integrate well with each other. In the meantime, we built the IR classes, and worked simultaneously on getting the tree walker functioning properly. When the tree walker was completed, we implemented the final step of compiling and executing the bell code so that the final output is printed to the file specified by the user.

Testing

Testing happened at multiple stages. We tested the lexer and the parser, and the walker. These Test scripts were developed in Perl to test the functionality of the language. We also did regression testing creating a test suite that tested BELL functionality by validating output based on meaningful input. These tests were developed as we created functionality, but they could not be used until the end because they test the final output of the BELL compiler.

Project Timeline

- 9/11 Form Group
- 9/17 Decide on Language
- 9/25 Proposal
- 9/26 Begin Lexer
- 10/6 Begin Parser
- 10/18 Language Reference Manual
- 10/19 Begin Tree Walker
- 11/1 Begin IR Classes
- 12/18/07 Regression Testing
- 12/15 Project Completion
- 12/17 Final Presentation
- 12/18/07 Final Report

Roles and Responsibilities

- Alicia Bozyk: Documentation, Interface, Presentation, Regression Testing
- Yousry El-Mallah: Lexer, Parser, Tree Walker, IR Classes
- Robert Lin: Documentation, Tree Walker, IR Classes
- Carlene Liriano:: Lexer, Parser, IR Classes

Software Development Environment

IDE

The team wrote the compiler for BELL through the IDE Eclipse and utilized an ANTLR plugin to generate code for the Lexer, Parser, and Walker. Working in an integrated developed environment was a major advantage because it helped us manage our code and took care of certain things that would be tedious to do on the command-line, such as compiling, without us having to automate the process ourselves. It also proved very useful in debugging.

Version Control System

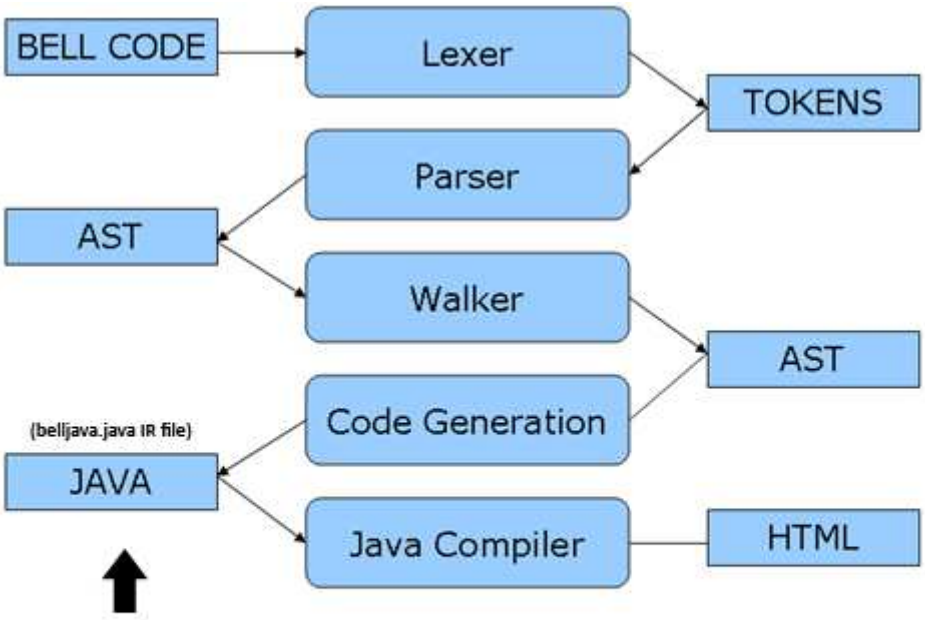
The team used Subclipse, a Subversion plugin for Eclipse to manage code versioning. Subclipse makes it extremely easy to use Subversion, and we were able to quickly check out and commit code to our repositories, through the built in features for Eclipse.

Project Log

- 9/11 Formed Group
- 9/17 Decided on Language
- 9/25 Proposal
- 10/15 Initial Lexer Completed
- 10/18 Initial Parser Completed
- 10/18 Language Reference Manual
- 10/19 Completion of Most IR Classes
- 12/8 Final Tree Walker Completed
- 12/9 Interface for BELL Compilation Completed
- 12/12 Regression Testing Completed
- 12/15 Project Completion
- 12/17 Final Presentation
- 12/18 Final Report

Architectural Design

A BELL file is read by calling java bell, and the end result is the output of an html file. BELL code is read by the Lexer and converted into tokens. These tokens are sent to the parser, which places them in an abstract syntax tree based on the BELL grammar. The Walker walks the abstract syntax tree, and then intermediate java code is then generated as a file called belljava.java. When our BELL compiler executes this code, the output is the final result, which is written to a file for the user.



(outside java classes can be called from the belljava.java IR file that is generated at this point. see the test.bell for an example.)

Test Plan

Test Methods

We used a couple of methods of testing throughout our entire project. We had simple test scripts to help us display what was happening as we parsed a file and walk the abstract syntax tree. The bulk of out testing was focused on regression testing, testing Bell functionality by validating output based on meaningful input.

Parser Test

We used a java class called TestParser.java to display the output of the abstract syntax tree that was generated by the parser.

Walker Test

The test we used to evaluate the tree walker, TestWalker.java, was used to display the processes happening in the Tree Walker.

Regression Testing

We did regression testing by creating a Java class to run a series of test BELL files, and check if they are run correctly. The only thing necessary to run the test suite is to run the file TestBell.java, which contains instructions for executing all of the test cases and checking to see if they are valid.

Test Cases

There were two types of test cases. Cases are included in the code appendix.

1. test.testname -> The test case to be run
   bell.testname -> The output that should be generated after compiling test.testname

Once the test is compiled, its output is checked against bell.testname, which contains the expected output. If the two files match, then the test has passed, and if not the test fails.

2. test.testname -> The test case to be run

The test cases are marked accepted or failed when executed.

Example output of the test cases follows:

```
Test: opeq
      1. 1 == 2          -> false     accepted
      2. 2 == 2          -> true      accepted
      3. "hi" == "hello"       -> false     accepted
      4. "hi" == "hi"    -> true      accepted
      5. true == false   -> false     accepted
      6. true == true    -> true      accepted
      7. 1 != 2          -> true      accepted
      8. 2 != 2          -> false     accepted
      9. "hi" != "hello"       -> true      accepted
      10. "hi" != "hi"   -> false     accepted
      11. true != false        -> true      accepted
      12. true != true   -> false     accepted
...accepted
Test: dop
      1. 1++             -> 2         accepted
      2. 1--             -> 0         accepted
...accepted
Test: arith
      1. 3 + 1           -> 4         accepted
      2. 3 - 1           -> 2         accepted
      3. 3 * 1           -> 3         accepted
      4. 3 / 1           -> 3         accepted
      5. 3 + 3 / 1 - 1         -> 5         accepted
...accepted
```

Lessons Learned

Individual Reflections

Alicia Bozyk

The creation of a language was a challenging process, but was fun because we learned a lot along the way. We had difficulty in the beginning since none of us had ever created a compiler before. Once we got through the initial phases, things became clearer and we were better able to appreciate the scope of our project and the fact that we were in control of the language. I now have a better understanding of what goes into the creation of a language, and would like to attempt to make another one in the future.

Our project also provided a good learning experience in terms of working in groups. It was tough trying to work around the schedules of four busy students. Creating BELL helped me learn more about project management, and the importance of good system of communication on many levels. Since the creation of a compiler was a new task for all of us, it was helpful to have a team to work things out and help each other better understand each step along the way. We did not have a good version management system in place, which was extremely necessary. We finally got a Subversion plugin for Eclipse set up, and once this happened we were able to make progress in shorter amounts of time. Without it, it was difficult to see and learn from the progress made by each member.

BELL was also one of the largest projects I worked on. You have to do a lot of planning, and understand what is happening in every piece of code in order to create a good language. It was also satisfying to work on such a large project and utilize the testing methods associated with software engineering. I have definitely learned a lot about managing a large project and building a compiler for a language, and will be able to use these lessons in the future.

Yousry El-Mallah

The final stages of developing BELL's compiler over the past 6 weeks have been a challenge but a very valuable and enriching one. The experience and technical skills I gained from developing the BELL compiler were one of a kind. This project gave me the opportunity to learn the importance of collaborating and working as a team to meet requirements, demands, and due dates. I also discovered the impact that communication, future planning and time management can have on the performance and throughput of a team. Developing BELL's compiler was a very enjoyable and pleasing experience since it furthered my interest and abilities in the design and implementation of programming languages. One of the biggest mistakes I made during the development process was coding the parser without complete knowledge of the structure of the abstract syntax tree and its walker which was very difficult to figure out later in the process. So my advice to everybody is to never underestimate the tree walker and always believe in the power of recursion.

Robert Lin

In this project I learned many things. Many were the usual suspects such as the collective importance of teamwork, communication, collaboration, and source control. And then of course, I also learned a lot about the inner-workings of ANTLR and what goes into creating a language.

But the one lesson I really learned through the experience of our project, a lesson I had never learned to this degree before, was how to suffer massive, epic, unprecedented failure. It was very heart-crushing for me to write an entire Walker that we eventually disposed of. And although that knowledge of failure (essentially what not to do) did help me and Yousry work on the final Walker, it was nonetheless a difficult experience. It put our team behind schedule and could have jeopardized the entire project were it not for the brilliance of my other teammates. Our group met frequently so it was not a question of communication – we communicated very well. But I essentially started the Walker at a time before I knew what I was really doing. And I was reluctant to throw away code. And that, is very dangerous. For future teams: Never become attached to your code, no matter how beautiful you think it might be. It could be the end of you. (Also, don't even *start* your Walker until you have a fully-functioning idea of what your Lexer and Parser look like.)

For all we've been through, I am infinitely grateful that our language is working. We did finish the Walker and our language did come together in the end. It has been an utmost rewarding experience.

Carlene Liriano

Making a language from scratch was a huge undertaking but the process taught me more than any other CS course I've taken about programming languages on a high level. The project itself was extremely interesting because it finally put us in complete control of what we were producing. There was no spec or any requirements. Some of the most exciting times during the development of Bell were when we would have debates about things our language should and should not allow the user to do. Working on Bell taught me a lot of valuable lessons both programming wise and about the software life cycle in general.

One of the most important things I learned early on was that we were most productive when we dedicated longer blocks of time to team meetings. Weekly meetings are important, but if they only go for an hour then not very much can be done because a lot of time is spent just discussing the current state of development and what should come next.

Also, while it was important to work independent of each other, I found that bugs in the program were resolved fastest when all members were present to help solve the problem. This is something future team should keep in mind because a huge portion of creating a language is debugging it and making sure it does what you claim it can.

If I could give future teams advice it would be that planning is very important and you have to start planning early, but it is ok if you do not have the full scope of your language planned from day 1. Our team originally started out with a very high level idea for a language and as the semester wore on we focused it a lot more. The programming language we described in our proposal was not the same as the end product of our project. This is something that is bound to happen as during the

development process all ideas and concepts must be refined to a certain extent. Its important to understand that refining your original idea and ending up with a slightly different language in the end is a good thing because it shows growth in the project.

Advice for Future Teams

- Start early and try to make progress
- Allow room for changes
- Divide up the work, but take advantage of collaboration with other team members
- Try to set aside regular blocks of time to meet as a group
- Set an agenda for each meeting
- Enjoy the project!

Code

BELL Grammar

grammar.g

```
class bellLexer extends Lexer; // Yousry and Carlene

	options {
		charVocabulary = '\0'..'\377';
		testLiterals = false;
		k = 2;
		exportVocab = BELL;
	}

	// Ignore all whitespace
	WS:
	    ( ' ' | '\t' | '\r' | '\n' { newline(); } { newline(); }
	    ) { $setType(Token.SKIP); } ;


	BELL_CMTS:  (
		OCMT
		( // multi-line comments
		options {greedy=false;}
		: ('\n' | '\r')
		| ~( '\n' | '\r' )
		)*
		CCMT
		)
		{ $setType(Token.SKIP); }
		;

	NOT:		'!';
	LPAREN:		'(';
	RPAREN:		')';
	LBRACE:		'{';
	RBRACE:		'}';
	LBRACKET:	'[';
	RBRACKET:	']';
	DOT:		'.';
	COLON:		':';
	COMMA:		',';
	SEMI:		';';
	OCMT:		"(:";
	CCMT:		":)";
	BSTART:		"<?";
	BEND:		"?>";
	ASSIGN:		'=';
	DOP:		("++"|"--");
	OPEQ:		("+=" | "-=" | "*=" | "/\\=");
	// Assignment operators
```

```
// Math operators
PLUS:             '+';
MINUS:            '-';
MULT:             '*';
DIV:              '/';

// Comparison operators
LEQ:              "<=";
LT:               '<';
GEQ:              ">=";
GT:               '>';
EQ:               "==";
NOT_EQUAL:        "!=";
QMARK:            '?';

POW:              '^';
COMPAND:          "&&";
COMPOR:           "||";
AT:               '@';
PERCENT:          '%';
USCORE:           '_';
QUOTE:            '"';

ID options{testLiterals=true;}
        : LETTER (LETTER|Int|USCORE)*
        ;

NUMBER options{testLiterals=true;}
        : (Int)+ ('.' (Int)*)?
        ;

STRING_LITERAL
        : '"' (Escapes|~'"')* '"'
        ;

protected
        LETTER
        : ('a'..'z'|'A'..'Z')
        ;

protected
        Int
        : ('0'..'9')
        ;

protected Escapes
        : '\\' ( 'n' | 'r' | 't' | 'b' | 'f' | '"' | '\'' | '\\' )
        ;
```

```
/* ############################## P A R S E R ############################## */
class bellParser extends Parser; // Yousry and Carlene

	options {
		buildAST = true;	// Automatically build the AST
		k=2;			// Need lookahead of two for props
		exportVocab = BELL;
	}

	//taken from online tutorial
	tokens {
		BELL;
		DECL;
		HTML;
		NEG;
		FUNC;
		FUNC_CALL;
		HTMLJUNK;
		START;
	}

	type
		:
		"int" | "string" | "boolean"
		;

	htmljunk
		:
		    (ID|symbols)+
		    { #htmljunk = #([HTMLJUNK, "HTMLJUNK"], #htmljunk); }
		;

	symbols
		:
		    (NOT|LPAREN|RPAREN|LBRACE|RBRACE|LBRACKET|
		    |RBRACKET|DOT|COLON|COMMA|SEMI||ASSIGN|PLUS
		    |MINUS|MULT|DIV|QMARK|POW|AT|PERCENT)
		;

	htmlblock
		:
		LT! (ID) GT! (bellblock|htmlblock|htmljunk)* LT! DIV! (ID!) GT!
		    { #htmlblock = #([HTML, "HTML"], #htmlblock);
		;

	bellblock
	      :
		    BSTART!(statement|func_def)* BEND!
		    { #bellblock = #([BELL, "BELL"], #bellblock); }
	      ;

	start
	      :
		    (htmlblock | bellblock)*
		    { #start = #([START, "START"], #start);
	      ;
```

```
func_def
      :
            "fxn" ("void" | type) ID argumentList functionBody
            { #func_def = #([FUNC, "FUNC"], #func_def); }
      ;

argumentList
      :
            LPAREN! (varDecl (COMMA! varDecl)*)? RPAREN!
      ;

functionBody
      :
            LBRACE^ (statement)* RBRACE!
      ;

func_call
      :
            ID LPAREN! expr_list RPAREN!
            { #func_call = #([FUNC, "FUNC"], #func_call); }
      ;

expr_list
      :
            expression (COMMA! expression)*
      ;

varDecl
      :
            type ID
            {#varDecl = #([DECL, "DECL"], #varDecl); }
      ;

statement
      :     varDecl SEMI!
            | assignment SEMI!
            | printFunc SEMI!
            | func_call SEMI!
            | titleFunc SEMI!
            | callJava SEMI!
            | return_stmt SEMI!
            | if_stmt
            | while_stmt
            | for_stmt
            | SEMI!
      ;

printFunc
      :
            "print"^ LPAREN! Expression RPAREN!
      ;

printFunc
      :"print"^ LPAREN! Expression PAREN! ;

titleFunc :"title"^ LPAREN!   expression  RPAREN! ;
```

```
assignment
	:
		ID ASSIGN^ expression
		| ID DOP^
		| ID OPEQ^ expression
	;
expression
	:
		logical (COMPOR^ logical)*
	;

logical
	:
		relation (COMPAND^ relation)*
	;

relation
	:
		addition
		(
			(GT^ | LT^ | EQ^ | NOT_EQUAL^ | GEQ^ | LEQ^)
			addition
		)?
	;

addition
	:
		multiply
		(
			(PLUS^ | MINUS^)
			multiply
		)*
	;

multiply
	:
		unary
		(
			(MULT^ | DIV^)
			unary
		)*
	;

unary
	:
		rValue
		| MINUS! unary { #unary = #([NEG, "NEG"], #unary); }
	;

rValue
	:
		NUMBER
		| STRING_LITERAL
		| ("true" | "false")
		| ID
		| LPAREN! expression RPAREN!
		| func_call
```

```
            ;

      return_stmt
            :
                  "return"^ (expression)
            ;
      if_stmt
            :
                  "if"^ LPAREN! expression RPAREN! (statement| functionBody)
                  (
                        options {greedy = true;}:
                        "else"! (statement|functionBody)
                  )?
            ;

      while_stmt
            :
                  "while"^ LPAREN! expression RPAREN!
                  (statement | functionBody)
            ;

      for_stmt
            :
                  "for"^ LPAREN! assignment SEMI! expression SEMI! assignment
                  RPAREN! (statement | functionBody)
            ;


//####################################Walker#######################################

class bellWalker extends TreeParser; // Yousry and Robert
{
      SymbolTable stack = null;
}


      bellprogram returns [Stmt s]
      {s=null; Stmt s1=null; Type t;}
      : #(START
            s=bellbell //(s1 = bellbell{s = new Seq(s, s1);})?
            )
      ;

      bellbell returns [Stmt s]
      {s=null; Stmt s1=null; Type t; stack = new SymbolTable(stack,
Type.Void);}
      : #(BELL
            s=bellstmts
            )
      #(HTML ID
            s1=begin
            {
                  Arglist args = new Arglist();
                  args.add(new Decl(Type.String, "args[]"));
                  s1 = new Func(Type.Void, "main", args, s1);
                  s = new Seq(s, s1);
            }
```

```
      )
;

begin returns [Stmt s]
{s = null; Stmt s1=null; Stmt s2=null;}
: #(HTML ID {s1 = new Htmlobject2(#ID.getText());}
      (s = begin2)?
      {
            s2 = new Htmlobject3(#ID.getText());
            if(s!=null){s = new Seq(s1, s);

            s = new Seq(s, s2);}
            else{s = new Seq(s1, s2);}
      }


      )
;

begin2 returns [Stmt s]
{s=null; Stmt s1=null;}
:     s = othernodes (s1 = begin2{s = new Seq(s, s1);})?
;

othernodes returns [Stmt s]
{s=null; Stmt s1=null; Stmt s2=null; Stmt s3=null;}
: #(HTML ID {s2 = new Htmlobject2(#ID.getText());}
            (s = begin2)? //(s1 = othernodes{s = new Seq(s, s1);})?
            {
                  s3 = new Htmlobject3(#ID.getText());
                  if(s !=null) { s = new Seq(s2, s);

                  s = new Seq(s, s3);}
                  else {s = new Seq(s2, s3);}
            }
      )
      | #(BELL
            (s=stmts)?
            )
      | #(HTMLJUNK
            (s=htmlstmts)?
      )
;

bellstmts returns [Stmt s]
{s=null; Stmt s1;}
:
      s=bellstmt (s1=bellstmts{s=new Seq(s, s1);})?
;

bellstmt returns [Stmt s]
{s=null;}
: s = decl
 |s = function
;
```

```
htmlstmts returns [Stmt s]
{s=null; Stmt s1;}
:
      s=htmlstmt (s1=htmlstmts{s=new Seq(s, s1);})?
;

htmlstmt returns [Stmt s]
{s=null;}
:
      #(ID
            {
                  s = new Htmlobject(#ID.getText());
            }
      )
;

function returns [Stmt s]
{ s = null; Type t; Stmt s1; Arglist a = new Arglist(); }
      : #(FUNC "fxn" (t=type) ID
      {
            SymbolTable oldstack = stack; stack = new
            SymbolTable(stack, t);
      }

      (a=args)?
      {
            oldstack.put(#ID.getText(), t, a);
      }
      s1=functionbody
      {
            if (t != Type.Void) {
                  if (!s1.hasReturn) {
                        System.err.println("Error: Function " +
                        #ID.getText() + " may not return a value");
                        System.exit(0);
                  }
            }
            s = new Func(t, #ID.getText(), a, s1);
            stack = oldstack;
        }
    )
;

type returns [Type t]
      { t= null;  }
      :     ( "int" {t = Type.Int; }
            | "string" {t = Type.String; }
            | "boolean" {t = Type.Boolean;}
            | "void" {t=Type.Void;}
            ) ;
args returns [Arglist a]
{ a = new Arglist(); Decl a1; Arglist a2; }
      :     a1=decl
            {
```

```
                a.add(a1);
            }
            (a2=args {a.copy(a2); })?
      ;

functionbody returns [Stmt s]
{s = null;}
      : #(LBRACE
             (s=stmts)?
            )
      ;

stmts returns [Stmt s]
{ s = null; Stmt s1; }
      : s=stmt (s1=stmts { s = new Seq(s, s1); }   )?
      ;


decl returns [Decl d]
{ d = null; Type t;}
      :     #(DECL t=type ID
              {
                  if(stack.get(#ID.getText()) != null)
                  {
                        D.print( "error" );
                        System.err.println("Variable already declared:"
                        + #ID.getText());
                        System.exit(0);
                  }
                  stack.put(#ID.getText(), t);
                  d = new Decl(t, #ID.getText());
              }
          )
      ;

funccall returns  [Expression e]
{ e= null; ExprList args = new ExprList(); }
      :     #(FUNC ID (args=exprs)?
            {
                  ID func = stack.get(#ID.getText());
                  if(func==null)
                  {
                        System.err.println("Error: Undeclared
                        Identifier: " + #ID.getText());
                        System.exit(0);
                  }
                  else
                  {
                        e = new FuncCall(func, args);
                  }
              }
         )
      ;
assign returns [Set s]
{ s = null; Expression e; }
      : #(ASSIGN ID e=expr
            {
```

```
                    ID var = stack.get(#ID.getText());
                    if (var == null) {
                          System.err.println("Error: Undeclared
                          Identifier: " + #ID.getText());
                          System.exit(0);
                    }
                    else s = new Set(#ASSIGN.getText(), var, e);
               }
          )
        | #(OPEQ ID e=expr
               {
                    ID var = stack.get(#ID.getText());
                    if (var == null) {
                    System.err.println("Error: Undeclared Identifier: " +
                    #ID.getText());
                    System.exit(0);
                    }
                    else s = new Set(#OPEQ.getText(), var, e);
               }
          )
        | #(DOP ID
               {
                    ID var = stack.get(#ID.getText());
                    if (var == null) {
                    System.err.println("Error: Undeclared Identifier: " +
                    #ID.getText());
                    System.exit(0);
                    }
                    else s = new Set(#DOP.getText(), var, null);
               }
        )
;

stmt returns [Stmt s]
{ s = null; Stmt s1 = null; Set f1; Set f2; Type t; Expression e =
  null; ExprList args = new ExprList();}
        : s=decl
        | #("print" e=expr
               { s = new printFunction(e); }
            )
        | s=assign
        | #("if" e=expr (s=stmt | s=functionbody) (s1=stmt |
               s1=functionbody)?
               { s = new Cond(e, s, s1); }
            )
        | #("while" e=expr
               (s=stmt | s=functionbody)
               { s = new While(e, s); }
               )
        | #("for" f1=assign e=expr f2=assign
               (s=stmt | s=functionbody)
               { s = new For(f1, e, f2, s); }
               )
        | e=funccall
               {
                    s = new FuncCallStmt((FuncCall)e);
               }
```

```
      | #("title" e=expr
            {
                  s = new titleFunction(e);
            }
          )
      | #("callJava" e=expr
            {
                  s = new callJava(e);
            }
          )
      | #("return" (e=expr)?
            {
                  s = new Return(stack.getReturnType(), e);
            }

            )
      ;

exprs returns [ExprList e]
{ e = new ExprList(); Expression e1; ExprList e2; }
      : e1=expr
        { e.add(e1); }
        ( e2=exprs { e.copy(e2); })?
      ;

expr returns [Expression e]
{ Expression a, b; e = null; Type t;}
      :
        #(COMPOR a=expr b=expr { e = new Relations("||", a, b); } )
      | #(COMPAND a=expr b=expr { e = new Relations("&&", a, b); } )
      | #(EQ a=expr b=expr { e = new Relations("==", a, b); } )
      | #(NOT_EQUAL a=expr b=expr { e = new Relations("!=", a, b); } )
      | #(LT a=expr b=expr { e = new Relations("<", a, b); } )
      | #(LEQ a=expr b=expr { e = new Relations("<=", a, b); } )
      | #(GT a=expr b=expr { e = new Relations(">", a, b); } )
      | #(GEQ a=expr b=expr { e = new Relations(">=", a, b); } )
      | e=funccall
      | #(PLUS a=expr b=expr {  e = new Arithmetic("+", a, b); } )
      | #(MINUS a=expr b=expr { e = new Arithmetic("-", a, b); } )
      | #(MULT a=expr b=expr { e = new Arithmetic("*", a, b); } )
      | #(DIV a=expr b=expr { e = new Arithmetic("/", a, b); } )
      | #(COMMA a=expr b=expr { e = new Arithmetic(",", a, b); } )
      | #(NOT a=expr { e = new Not(a); } )
      | #(NEG a=expr { e = new UnaryOp("-", a); } )
      | #(ID
            {
            e = stack.get(#ID.getText());
                  if (e == null)
                  {
                        System.err.println("Error: Undeclared
                        identifier: " + #ID.getText());
                        System.exit(0);
                  }
            }
       )
      | NUMBER { e = new Constant(#NUMBER.getText(), Type.Int); }
      | "true" { e = Constant.True; }
```

```
      | "false" { e = Constant.False; }
      | STRING_LITERAL {e = new Constant(#STRING_LITERAL.getText()); }
;
```

IR Classes

Arglist.java - Yousry

```
import java.util.*;

public class Arglist extends Decl {

	public Vector decls;

	// Constructor to create a new empty vector
	public Arglist() {
		decls = new Vector();
	}

	// Method to add a declaration to the end of the vector
	public void add(Decl d) {
		decls.add(d);
	}

	/*
	 * Copy the elements in the vector of the Arglist a1 to
	 * the Vector Decl of this instance
	 */
	public void copy (Arglist a1) {
		for(int i=0; i<a1.decls.size(); i++) {
			decls.add(a1.decls.get(i));
		}
	}

	// Returns the size of the vector decls
	public int size() {
		return decls.size();
	}

	/* Returns the type of a declaration given the element number
	 * of the vector
	 */
	public Type getType(int i) {
		return ((Decl)(decls.get(i))).type;
	}

	// Prints out a list of all the arguments with commas seperating them
	public void gen() {
		if(decls.size() > 0) {
			((Decl)(decls.get(0))).gen2();
			for(int i=1; i<decls.size(); i++) {
				System.out.print(", ");
				((Decl)(decls.get(i))).gen2();
			}
		}
	}
}
```

Arithmetic.java - Carlene

```
public class Arithmetic extends Expression {
	public Expression ex1;
	public Expression ex2;

	/* Determines what the actual type of the expression should be
	 * i.e. if its a concatenation operation with a String an int the
	 * type of the expression should be String
	 */
	public Arithmetic(String op, Expression a, Expression b) {
		super(op, null);
		ex1 = a;
		ex2 = b;

		// Check for equal types
		if(ex1.type.typeClass == ex2.type.typeClass) {
			if(ex1.type == Type.String ) {
				if(op != "+") {
					error("ERROR: Strings can not be used with
					arithmetic operators");
				}
				else { //the type of the expression is String
					type= Type.String;
				}
			}
			else if (ex1.type == Type.Boolean) {
				error("ERROR: Boolean types can not be used with
				arithmetic operators");
			}
			else {
				type=ex1.type;
			}
		}
		else { //Check for different types
			if(ex1.type == Type.String) {
				if(op!= "+") {
					error("ERROR: Strings can not be used with
						arithmetic operators");
				}
				else if (ex2.type ==Type.Int ||
						ex2.type == Type.Boolean) {//concatenate
					type = Type.String;
					ex2.type = Type.String;
				}
			}
			else if(ex2.type == Type.String) {
				if(op!= "+") {
					error("ERROR: Strings can not be used with
						arithmetic operators");
				}
				else if (ex2.type ==Type.Int || ex2.type ==
						Type.Boolean) {
					type = Type.String;
					ex2.type = Type.String;
				}
			}
```

```
                else {
                        type= Type.Int;
                }
        }
    }

    /*
     * Makes the actual call to apply the numeric operator
     */
    public Expression evaluateConst() {

        //str comes from the Expression class
        if (ex1 instanceof Constant && ex2 instanceof Constant) {
                Constant c = new Constant(0);
                switch(str.charAt(0)) {
                case '+':
                        c = new Constant(Integer.parseInt(ex1.str) +
                                        Integer.parseInt(ex2.str));
                        break;
                case '-':
                        c = new Constant(Integer.parseInt(ex1.str) -
                                        Integer.parseInt(ex2.str));
                        break;
                case '*':
                        c = new Constant(Integer.parseInt(ex1.str) *
                                        Integer.parseInt(ex2.str));
                        break;
                case '/':
                        c = new Constant(Integer.parseInt(ex1.str) /
                                        Integer.parseInt(ex2.str));
                        break;
                }
                return c;
        }
        return this;
    }

    /*
     * Keep recursively calling Arithmetic until the expression
     * is reduced to constants
     */
    public Expression reduce() {
        return (new Arithmetic(str, ex1.reduce(),
                        ex2.reduce())).evaluateConst();
    }

    public void gen() {
        System.out.print(reduce().toString());
    }

    public String toString() {
        String result;
        return ex1.toString() + " " + str + " " + ex2.toString();
    }
}
```

Cast.java - Yousry

```
public class Cast extends Expression {
	Expression expr;

	public Cast(Type t, Expression e) {
		super("CAST", t);
		if(t == Type.String) {
			if(!e.type.isNumeric()) {
				expr = e;
			}
			else {
				expr = e;
			}
		}
		else if (!(t.isNumeric() && e.type.isNumeric())) {
			error("Error: No valid cast defined between " + t + " and "
+ e.type);
		}
		else {
			e.type = Type.Int;
			expr = e;
			expr.type = t;
		}
	}

	public Expression reduce() {
		return expr.reduce();
	}

	public String toString() {
		return expr.toString();
	}

	public void gen() {
		expr.gen();
	}
}
```

Cond.java - Yousry

```
public class Cond extends Stmt {
	Expression expr;
	Stmt ifstmt, elsestmt;

	public Cond(Expression e, Stmt s1, Stmt s2) {
		if (e.type != Type.Boolean) {
			error("Error: Argument to if statment must be a boolean
expression");
		}
		expr = e;
		ifstmt = s1;
		elsestmt = s2;
	}

	public void gen() {
		System.out.print("if (");
		expr.gen();
		System.out.println(") {");
		if (ifstmt != null) {
			ifstmt.gen();
		}
		System.out.println("}");
		if (elsestmt != null) {
			System.out.println("else {");
			if (elsestmt != null) {
				elsestmt.gen();
			}
			System.out.println("}");
		}
	}
}
```

Constant.java - Yousry

```
public class Constant extends Expression {
	public Constant(String tok, Type p) {
		super(tok, p);
	}

	public Constant (int i) {
		super(String.valueOf(i), Type.Int);
	}

	public Constant (String s) {
		super(s, Type.String);
	}

	public static final Constant True = new Constant("true", Type.Boolean),
		False = new Constant("false", Type.Boolean);
}
```

D.java - Robert

```
/*
 * D.java
 *
 * Diagnostic Print
 * When D is true, prints all statements D.print in code
 */
public class D {

	public static boolean D = false;

	public static void print( String a ) {
		//if( D )
		System.out.println( a );
	}

	public static void print (Boolean a) {
		System.out.println( a );
	}
}
```

Decl.java - Yousry

```
/* Decl.java
 * Decl represents a declaration and extends statement
 * Datastructure Decl consists of an ID of type string,
 * type which is the type of the ID and javatype which is
 * also the ID type but as a string.
 */
public class Decl extends Stmt {
	public String ident;
	public Type type;
	String javatype;

	//Empty constructor because Arglist class extends it
	public Decl() {}

	//Constructor for a declaration object
	public Decl(Type t, String id) {
		ident = id;
		type = t;

		if(t.typeClass == Type.BOOLEAN) {
			javatype = "boolean";
		}
		else if (t.typeClass == Type.INT) {
			javatype = "int";
		}
		else if (t.typeClass == Type.STRING) {
			javatype = "String";
		}
	}

	//Prints out the declarations initialized
	public void gen() {
		System.out.print("public static "+ javatype + " " + ident);
		if(type.isNumeric()) {
			System.out.println(" = 0;");
		}
		else if (type == Type.String) {
			System.out.println(" = \"\";");
		}
		else if (type == Type.Boolean) {
			System.out.println(" = false;");
		}
		else {
			System.out.println(";");
		}
	}

	/* Also prints the declaration type and its ID but doesnt add a
	 * semi colon at the end since ArgList class is going to use it to
	 * print out a list of the arguments
	 */
	public void gen2() {
		System.out.print(javatype + " " + ident);
	}
}
```

doSomething.java - Robert

```
public class doSomething {

	public static String sayHi() {
		return ("hi works");
	}

	public static void testOutsidePrint() {
		System.out.println( "test outside print" );
	}

	public static int addOne(int a) {
		return a+1;
	}
}
```

Expression.java - Carlene

```
public class Expression extends Node {

	public String str;
	public Type type;

	Expression () {}

	Expression (String s, Type t) {
		str=s;
		type=t;
	}

	public void  gen() {
		System.out.print(reduce().toString());
	}

	public Expression reduce() {
		return this;
	}

	public String toString() {
		return str;
	}
}
```

ExprList.java - Carlene

```
import java.util.*;

public class ExprList extends Expression {
	public Vector exprs;

	public ExprList() {
		super();
		exprs= new Vector();
	}

	/*
	 * Add an expression to a list of Expressions
	 */
	public void add(Expression expr) {
		exprs.add(expr);
	}

	/*
	 * Returns the size of the expression list
	 */
	public int size() {
		return exprs.size();
	}

	/*
	 * Get the type of an expression in the list
	 */
	public Type getType(int index) {
		Expression tmp= (Expression)exprs.get(index);
		Type t= tmp.type;

		return t;
	}

	/*
	 * Copies one expression list into another
	 */
	public void copy(ExprList lst) {
		for (int i = 0; i < lst.exprs.size(); i++) {
			exprs.add(lst.exprs.get(i));
		}
	}

	/*
	 * Print out a comma separated list of expressions
	 */
	public void gen() {
		if (exprs.size() > 0) {
			((Expression)(exprs.get(0))).gen();
			for (int i = 1; i < exprs.size(); i++) {
				System.out.print(", ");
				((Expression)(exprs.get(i))).gen();
			}
		}
	}
```

F.java - Robert

```
import java.io.BufferedWriter;
import java.io.FileWriter;

public class F {

	public static void write( String a ) {
		try {
			// begin BELL file
			FileWriter fstream = new FileWriter("bellIR.java", true);
			BufferedWriter out = new BufferedWriter(fstream);
			out.write( a );
			//Close the output stream
			out.close();
		} catch (Exception e){//Catch exception if any
			System.err.println("Error: " + e.getMessage());
		}
	}

	public static void write( int a ) {
		try {
			// begin BELL file
			FileWriter fstream = new FileWriter("bellIR.java", true);
			BufferedWriter out = new BufferedWriter(fstream);
			out.write( Integer.valueOf( a ).toString() );
			//Close the output stream
			out.close();
		} catch (Exception e){//Catch exception if any
			System.err.println("Error: " + e.getMessage());
		}
	}
}
```

For.java - Yousry

```
public class For extends Stmt {
	Set init, incr;
	Expression expr;
	Stmt body;

	// Creates a For object and checks the validity of the loop
	public For(Set s1, Expression e, Set s2, Stmt b) {
		init = s1;
		incr = s2;
		if(e.type != Type.Boolean) {
			error("Error: Second arg of for loop must be a boolean
				expression");
		}
		expr = e;
		body = b;
	}

	// Prints out the for loop with all of its elements
	public void gen() {
		System.out.print("for(");
		init.gen2();
		System.out.print("; ");
		expr.gen();
		System.out.print("; ");
		incr.gen2();
		System.out.println(") {");
		if (body != null) {
			body.gen();
		}
		System.out.println("}");
	}
}
```

Func.java - Yousry

```
//A data structure to hold all the contents of a function
public class Func extends Stmt  {
	String ident, type;
	Arglist args;
	Stmt body;
	int typeClass;

	// Creates a Func object, checks type, and sets it to the right value
	public Func(Type t, String id, Arglist a, Stmt b) {
		ident = id;
		if (t.typeClass == Type.BOOLEAN) {
			type = "boolean";
		}
		else if (t.typeClass == Type.INT) {
			type = "int";
		}
		else if (t.typeClass == Type.STRING) {
			type = "String";
		}
		else if (t.typeClass == Type.PUBLICS) {
			type = "void";
		}
		else if (t.typeClass == Type.VOID){
			type = "void";
		}
		typeClass = t.typeClass;
		args = a;
		body = b;
	}

	//prints out the function in the correct java format
	public void gen() {
		System.out.print("public static " + type + " " + ident + "(");
		if (args.size() != 0) {
			args.gen();
		}
		System.out.println(") {");
		if (body != null) {
			body.gen();
		}
		if (type == "int") {
			System.out.println("return 0;");
		}
		else if (type == "boolean") {
			System.out.println("return true;");
		}
		else if (type == "String") {
			System.out.println("return \"\";");
		}
		else {
			System.out.println("return;");
		}
		System.out.println("}");
	}
}
```

FuncCall.java - Yousry

```
public class FuncCall extends Expression {

	String ident;
	String javatype;
	ExprList args;

	public FuncCall(ID func, ExprList a) {

		ident = func.str;
		type = func.type;

		if (type.typeClass == Type.BOOLEAN) {
			javatype = "boolean";
		}
		else if (type.typeClass == Type.INT) {
			javatype = "int";
		}
		else if (type.typeClass == Type.STRING) {
			javatype = "String";
		}

		args = a;
		if (args.size() != func.args.size()) {
			error("Error: Wrong number of arguments to function " +
				func);
		}
		for (int i = 0; i < args.size(); i++) {
			if (args.getType(i) == Type.Void) {
				error("Error: Passing void type as function
					argument");
			}
			if ( args.getType(i).isNumeric() !=
			     func.args.getType(i).isNumeric()
			     && func.args.getType(i) != Type.String) {

				error("Error: Passing incompatible type to function "
					+ func);
			}
		}
	}

	public void gen() {
		System.out.print(ident + "(");
		if (args.size() != 0) {
			args.gen();
		}
		System.out.print(")");
	}
}
```

FuncCallStmt.java - Yousry

```
public class FuncCallStmt extends Stmt {
	FuncCall func;

	public FuncCallStmt(FuncCall f) {
		func = f;
	}

	public FuncCallStmt(ID f, ExprList a) {
		func = new FuncCall(f, a);
	}

	public void gen() {
		func.gen();
		System.out.println(";");
	}
}
```

Htmlblock.java - Yousry

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

public class Htmlblock {

    private String name;
    private List bellblocks = new ArrayList();

    public Htmlblock(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void addBlock(Bellblock bellblock) {
        this.bellblocks.add(bellblock);
    }

    public List getBlocks() {
        return this.bellblocks;
    }

    public String toString() {
        StringBuffer buffer = new StringBuffer(Tools.quoteString(this.name));
        buffer.append(" {\n");


        Iterator blockIter = this.bellblocks.iterator();
        while(blockIter.hasNext()) {
            Bellblock bellblock = (Bellblock) blockIter.next();
            buffer.append( bellblock );
        }
        buffer.append("}\n\n");
        return buffer.toString();
    }
}
```

Htmlobject.java - Yousry

```
public class Htmlobject extends Stmt {
	Stmt x;

	public Htmlobject(String s) {
		x = new Stmt(s);
	}

	public void gen() {
		System.out.println("System.out.println(\"" + x.str + "\"); ");
	}
}
```

Htmlobject2.java - Yousry

```
public class Htmlobject2 extends Stmt {
	Stmt x;

	public Htmlobject2(String s) {
		x = new Stmt(s);
	}

	public void gen() {
		System.out.println("System.out.println(\"<" + x.str + ">\"); ");
	}
}
```

Htmlobject3.java - Yousry

```
public class Htmlobject3 extends Stmt {
	Stmt x;

	public Htmlobject3(String s) {
		x = new Stmt(s);
	}

	public void gen() {
		System.out.println("System.out.println(\"</" + x.str + ">\"); ");
	}
}
```

ID.java - Carlene

```
public class ID extends Expression {

	public Arglist args= new Arglist();

	public ID(String str, Type t) {
		super(str, t);
	}


	public ID(String id, Type p, Arglist a) {
		super(id, p);
		args = a;
	}

	/*
	 * Prints out the string in the ID
	 * and ID has a type followed by the actual 'string'
	 * str comes from the public string variable in the Expressions class
	 */
	public void gen() {
		System.out.print(str);
	}
}
```

Node.java - Yousry

```
/*
 * Node.java
 * Node object is the class that all other classes inherit from
 * and its two subclasses are Expression and Stmt that represent
 * expressions and statements respectively.
 */
public class Node {
	// Prints any error messages and exits the program
	void error(String s) {
		System.err.println(s);
		System.exit(0);
	}
}
```

Not.java - Carlene

```
public class Not extends Expression {

	Expression ex;
	public Not(Expression e) {
		//the not operator can only work on boolean expressions
		super("!", Type.Boolea);
		ex=e;
		if( e.type != Type.Boolean ) {
			error("ERROR The Not operator can only be applied to
			boolean types");
		}
	}

	/*
	 * Returns an expression of the form !(expr) where expr is of
	 * type boolean
	 */
	public String toString() {
		return str + " "+ ex.toString();
	}
}
```

printFunction.java - Robert

```
public class printFunction extends Stmt {
	Expression line;

	public printFunction(Expression e) {
		//line= e;
		line = new Cast(Type.String, e);
	}

	//When this call actually gets made in "java land" System.out
	//will know to implicitly call the toString method of line
	//which is an expression...so it boils down to a string
	public void gen() {
		System.out.print("System.out.println(");
		System.out.println(line + ");");
	}
}
```

Relations.java - Carlene

```
/*
 * Relation.java
 * Handles relational operators "<", ">", "<=", ">="
 * Relational operators return expression of type boolean
 * All relational operators work on ints, but NOT for Strings and booleans
 */
public class Relations extends Expression {
	Expression expr1, expr2;

	public Relations(String op, Expression e1, Expression e2) {
		super(op, Type.Boolean);

		//Check for the operators that only accept boolean "&&" and "||"
		if (op == "&&" || op == "||") {
			if(!(e1.type == Type.Boolean && e2.type == Type.Boolean)) {
				error("Error: && and || operators must take boolean "
					+ "expressions as operands");
			}
		}

		/* if both expressions are numbers then all operations are fine
		 * if either or both are string or booleans "<",">",">=","<="
		 * won't work
		 * /
		else if (!(e1.type.isNumeric() && e2.type.isNumeric())) {
			if (!(e1.type == e2.type && (op == "==" || op == "!="))) {
				error("Error: Incompatible types");
			}
		}
		expr1 = e1;
		expr2 = e2;
		if (expr1.type.typeClass != expr2.type.typeClass) {
			if (expr1.type == Type.Int) {
				expr1.type = expr2.type;
			}
			else if (expr2.type == Type.Int) {
				expr2.type = expr1.type;
			}
		}
	}

	public void gen() {
		expr1.gen();
		System.out.print(" " + str + " ");
		expr2.gen();
	}
}
```

Return.java - Yousry

```
public class Return extends Stmt {
	Expression returnValue;
	Type returnType;

	public Return(Type t, Expression e) {
		//Check to make sure return value is consistent with function
type
		if (t == Type.Void) {
			if (e != null) {
				error("Error: Void function cannot return value");
			}
		}
		else {
			if (e == null) {
				error("Error: Null value in return statement of
function of type " + t);
			}
			else if (t != Type.String) {
				if (t.isNumeric() != e.type.isNumeric()) {
					error("Error: Return type not consistent with
function type");
				}
			}
		}
		returnType = t;
		returnValue = e;
		hasReturn = true;
	}

	public void gen() {
		System.out.print("if (true) return ");
		//System.out.print("return ");
		if (returnValue != null) {
			returnValue.gen();
		}
		System.out.println(";");
	}
}
```

Seq.java - Yousry

```
/*
 * Seq.java
 * Class to ensure statements are read in the correct order
 * (left child followed by right child)
 */
public class Seq extends Stmt {
      Stmt stmt1;
      Stmt stmt2;

      public Seq(Stmt s1, Stmt s2) {
            stmt1 = s1;
            stmt2 = s2;
            if( s1.hasReturn || s2.hasReturn ) {
                  hasReturn = true;
            }
      }

      //prints both statements in the correct order
      public void gen() {
            stmt1.gen();
            stmt2.gen();
      }
}
```

Set.java - Yousry

```
public class Set extends Stmt {
	public ID var;
	public Expression e;
	public String op;

	public Set(String o, ID lhs, Expression rhs) {
		op = o;
		var = lhs;
		e = rhs;
		if(e == null) {
			if(op.charAt(1) == '+') { //To handle ++ operator
				op = "+=";
				e = new Constant(1);
			}
			else if(op.charAt(1) == '-') { //To handle -- operator
				op = "-=";
				e = new Constant(1);
			}
		}
		if(op.charAt(0)== '^') { //To handle the power operator
			e = new Arithmetic(",", var, e);
			op = "=";
		}

		/*
		 * The expression and the variable has to be of a numeric type
		 * or else an error message is printed it also makes that a
		 * expression of type string cannot be assigned to a variable of
		 *  a numeric type
		 */

		if( ( ( var.type == Type.Int)&& ((e.type == Type.String)
			 || (e.type == Type.Boolean))) |||((var.type == Type.String)
			 && ((e.type == Type.Int) || (e.type == Type.Boolean)) )) {
			error("Error: Assignment of incompatible types");
		}
	}

	// Prints the set statement with a semicolon at the end
	public void gen() {
		gen2();
		System.out.println(";");
	}

	// Prints the set statement without a semicolon
	public void gen2() {
		var.gen();
		System.out.print(" " + op + " ");
		e.gen();
	}
}
```

Start.java - Yousry

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

public class Start {
	private List htmlblocks = new ArrayList();

	public void addBlock(Htmlblock htmlblock) {
		D.print( "--added and Htmlblock!");
		this.htmlblocks.add(htmlblock);
	}

	public List getBlocks() {
		return this.htmlblocks;
	}

	public String toString() {
		StringBuffer buffer = new StringBuffer();
		Iterator blockIter = htmlblocks.iterator();

		while(blockIter.hasNext()) {
			Htmlblock block = (Htmlblock) blockIter.next();
			buffer.append(block);
		}
		return buffer.toString();
	}
}
```

Stmt.java - Yousry

```
import java.io.*;

/* Stmt.java
 * A Stmt object contains a string that contains the actual statement
 * and a boolean that indicates whether this statement is a return statement
 */
public class Stmt extends Node {
	public boolean hasReturn = false;
	String str;

	// Empty Constructor so that subclasses can use it
	public Stmt() {}

	// Sets the input string s to variable str
	public Stmt(String s) {
		str = s;
	}

	// Prints the statement followed by a semi colon
	public void gen() {
		System.out.println(str + ";");
	}
}
```

SymbolTable.java - Carlene

```
import java.util.*;

public class SymbolTable {
	private Hashtable table;
	protected SymbolTable parent;
	public Type returnType = null;

	public SymbolTable (SymbolTable t) {
		table = new Hashtable();
		parent = t;
	}

	public SymbolTable (SymbolTable t, Type s) {
		table = new Hashtable();
		parent = t;
		returnType = s;
	}

	/* Given a key and its type, check the table if it exists, thrown
	 * an error otherwise, add it to the tabel with its ID */
	public void put(String key, Type t) {
		if( get(key) != null ) {
			D.print("Variable "+ key+ " already defined");
		}
		else { table.put(key, new ID(key, t)); }
	}

	public void put (String token, Type t, Arglist a) {
		table.put(token, new ID(token, t, a));
	}

	/* Given a key, check the current table to see if that key already
	 * exists. If it is not in that table, check its parent table. Keep
	 * doing so until you reach the outermost table. If the key is in any
	 * one of the tables return it. Otherwise return null.
	 */
	public ID get(String key) {
		for (SymbolTable thisTable = this; thisTable != null;
		thisTable = thisTable.parent) {
			ID id = (ID) (thisTable.table.get(key));
			if (id != null) {
				ID result = new ID(id.str, id.type);
				result.args = id.args;
				return result;
			}
		}
		return null;
	}

	public Type getReturnType() {
		SymbolTable temp = this;
		while (temp.returnType == null) { temp = this.parent; }
		return temp.returnType;
	}
}
```

Tools.java - Robert

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;


public class Tools {

    public static String quoteString(String value) {
        if(value != null) {
            for(int i=0; i<value.length(); i++) {
                char c = value.charAt(i);
                if(! Character.isLetterOrDigit(c) && c != '_') {
                    return "" + value + "";
                }
            }
        }
        return value;
    }
}
```

Type.java - Carlene

```
/*
 * Type.java
 * Represents the four basic types of Bell. Includes a method to determine
 * whether or not a type is an int. Bell does not support Type promotions.
 */
public class Type {

	//Four main types supported by BELL
	public static final int INT = 1;
	public static final int BOOLEAN = 2;
	public static final int STRING = 3;
	public static final int VOID = -1;
	public static final int PUBLICS = 4;

	public String name = "";
	public int typeClass = 0;

	public Type(String s, int cls) {
		name = s;
		typeClass = cls;
	}

	// Types
	public static final Type
	Void = new Type ("void", VOID),
	Int = new Type("int", INT),
	Boolean = new Type("boolean", BOOLEAN),
	String = new Type("string", STRING),
	Publics = new Type("void", PUBLICS);

	//Returns true only if it is an INT
	public boolean isNumeric() {
		return typeClass == Type.INT;
	}

	public String toString() {
		return name;
	}
}
```

UnaryOp.java - Carlene

```
/*
 * UnaryOp.java
 * Handles unary operators "-", "++", "--"
 * These unary operators only work with Ints so if something other
 * than an int is passed an error is thrown
 */

public class UnaryOp extends Expression {

	Expression ex;

	public UnaryOp(String op, Expression e) {
		super(op, null);

		type=ex.type;

		if(ex.type!= Type.Int) {
			error("ERROR Unary operators are defined for " +
					"expressions of Type " + ex.type);
		}

		/*
		 * If the the unary operator isn't one of the ones we defined
		 * in our lexer than throw an error because we won't know what
		 * to do with it
		 */
		ex=e;
	}

	/*
	 * Basic too string method of the form "op""expr" for negations and
	 * expr"op" for increment and decrement functions
	 */
	public void gen() {

		System.out.print("(");
		if (str == "-") {
			System.out.print(str);
		}
		System.out.print("(");
		ex.gen();
		System.out.print(")");
		if (str == "++" || str == "--") {
			System.out.print(str);
		}
		System.out.print(")");
	}
}
```

While.java - Yousry

```
public class While extends Stmt {
	Expression expr;
	Stmt body;

	// Checks whiel has correct types and create the object
	public While(Expression e, Stmt s) {
		if(e.type != Type.Boolean) {
			error("Error: Argument to while statement must be a " +
					"boolean expression");
		}
		expr = e;
		body = s;
	}

	// Prints while statement out in correct java format
	public void gen() {
		System.out.print("while (");
		expr.gen();
		System.out.println(") {");
		if(body != null) {
			body.gen();
		}
		System.out.println("}");
	}
}
```

Compiling BELL code

bell.java - Alicia

```
import java.io.*;

public class bell {
      public static void main (String args[]) throws IOException, Exception {

            String belloutput_file = "a.html";

            if ( args.length != 1 && args.length != 2 ) {
                  System.err.println("Usage: bell code.bell
(optional)output.html");
                  System.exit(-1);
            }
            if ( args.length == 2 ) {
                  belloutput_file = args[1];
            }

            String bellcode = args[0];

            bell_out.bell_out(bellcode, belloutput_file);
      }
}
```

bell_out.java - Alicia

```
import java.io.*;

import antlr.CommonAST;

public class bell_out {

      public static void bell_out(String bellcode, String belloutput_file)
            throws IOException, Exception {

            Process p_compile = null;
            Process p_exec = null;
            String in = null;

            // Run code through, lexer, parser, and walker
            Reader reader = new BufferedReader(new FileReader(bellcode));
            bellLexer lexer = new bellLexer(reader);
            bellParser parser = new bellParser(lexer);
            parser.start();
            CommonAST t = (CommonAST)parser.getAST();
            bellWalker walker = new bellWalker();
            Stmt stmt = walker.bellprogram(t);

            // Generate intermediate java code and write it to belljava.java
            BufferedWriter out = new BufferedWriter(new
                              FileWriter("belljava.java"));
```

```
            out.write("import java.io.*;\n"
                        + "public class belljava { \n");
            out.close();

            // Redirect stdout to file, since that's how code is generated
            PrintStream ps = System.out; //backup
            System.setOut(new PrintStream(new
                              FileOutputStream("belljava.java", true)));
            stmt.gen();
            System.setOut(ps);

            out = new BufferedWriter(new FileWriter("belljava.java", true));
            out.write("}\n");
            out.close();

            try {
                  // Compile belljava.java through system process
                  p_compile = Runtime.getRuntime().exec("javac
                               belljava.java");
                  BufferedReader comp_stdErr = new BufferedReader(new
                              InputStreamReader(p_compile.getErrorStream()));
                  // If compilation failed, quit
                  while ((in = comp_stdErr.readLine()) != null) {
                        System.err.println("failed - compiling");
                        System.exit(-1);
                  }

                  // Generate output of belljava.java
                  p_exec = Runtime.getRuntime().exec("java belljava");
                  BufferedReader exec_stdIn = new BufferedReader(new
                              InputStreamReader(p_exec.getInputStream()));
                  BufferedReader exec_stdErr = new BufferedReader(new
                              InputStreamReader(p_exec.getErrorStream()));

                  BufferedWriter fout = new BufferedWriter(new
                        FileWriter(belloutput_file));
                  // Redirect java output to final output file
                  while ((in = exec_stdIn.readLine()) != null) {
                        fout.write(in + "\n");
                  }
                  fout.close();
                  while ((in = exec_stdErr.readLine()) != null) {
                        System.err.println("Quitting, error compiling file");
                        System.err.println(in);
                        System.exit(-1);
                  }
            }
            catch (IOException e) {
                  System.err.println("Exception Happened: ");
                  e.printStackTrace();
                  System.exit(-1);
            }
      }
}
```

Test Classes

TestParser.java - Carlene

```
import java.io.*;

import antlr.CommonAST;
import antlr.DumpASTVisitor;

public class TestParser {

    public static void main(String[] args) throws Exception {
        System.out.println("Parsing: " + args[0]);

        Reader reader = new BufferedReader(new FileReader(args[0]));
        bellLexer lexer = new bellLexer(reader);
        bellParser parser = new bellParser(lexer);
        parser.start();

        CommonAST ast = (CommonAST)parser.getAST();
        DumpASTVisitor visitor = new DumpASTVisitor();
        visitor.visit(ast);
    }
}
```

TestWalker.java - Yousry

```
public class TestWalker {

      public static void main(String args[]) throws Exception {

            System.out.println( "public class bellIR{\n ");

            Reader reader = new BufferedReader(new FileReader(args[0]));
            bellLexer lexer = new bellLexer(reader);
            bellParser parser = new bellParser(lexer);

            parser.start();
            CommonAST t = (CommonAST)parser.getAST();

            //ASTFrame frame = new ASTFrame("AST", t);
            //frame.setVisible(true);

            // Traverse the tree created by the parser
            bellWalker walker = new bellWalker();

            Stmt stmt = walker.bellprogram(t);
            stmt.gen();

            System.out.println( "\n}");
      }
}
```

TestBell.java - Alicia

```
import java.io.*;
import java.util.regex.*;

public class TestBell {

	public static void main(String[] args) throws Exception {

		test("print", 1);
		test("assign", 2);
		test("comment", 2);
		test("int", 2);
		test("string", 2);
		test("boolean", 2);
		test("arith", 2);
		test("dop", 2);
		test("rel", 2);
		test("opeq", 2);
		test("comp", 2);
		test("if", 2);
		test("for", 1);
		test("while", 2);
		test("return", 2);

	}

	public static void test( String testCase, int type) throws Exception {

		String testFile = "bell/tests/test." + testCase;
		String outFile = "bell/tests/out." + testCase;

		bell_out.bell_out(testFile, outFile);

		System.out.println("Test: " + testCase);

		// Accepts or fails a test based on matching two files
		if ( type == 1 ) {
			String bell = readTextFile("bell/tests/bell." + testCase);
			String out = readTextFile(outFile);

			Pattern r = Pattern.compile("\r|\n|\t|\b|\f| ");
			Matcher m = r.matcher(bell);
			bell = m.replaceAll("");
			m = r.matcher(out);
			out = m.replaceAll("");

			if ( bell.equals(out) ) {
				System.out.println("...accepted");
			}
			else {
				System.err.println("...failed");
			}
		}
```

```
            // Accepts or fails a test based on info resulting from test
            else {
                  DataInputStream in = new DataInputStream(new
                        FileInputStream(outFile));
                  BufferedReader br = new BufferedReader(new
                        InputStreamReader(in));

                  String line;
                  String test_result = "accepted";
                  Pattern a = Pattern.compile("^\\d.*accepted$");
                  Pattern f = Pattern.compile("^\\d.*failed$");
                  Matcher m = null;
                  while ((line = br.readLine()) != null)   {
                        // Print the content on the console
                        m = a.matcher(line);
                        if ( m.matches() ) {
                              System.out.println("\t" + line);
                        }
                        m = f.matcher(line);
                        if ( m.matches() ) {
                              System.out.println("\t" + line);
                              test_result = "failed";
                        }
                  }
                  in.close();
                  System.out.println("..." + test_result);
            }
      }

      /*
       * Reads in a text file to a string
       */
      public static String readTextFile(String file) throws IOException {
            FileInputStream f= new FileInputStream(file);
            int len= f.available();
            String s= "";

            for( int i = 1; i <= len; i++) {
                  s = s+(char)f.read();
            }
            return s;
      }
}
```

Test Cases - Alicia

test.arith

```
<?
      int a;
      int b;
      int c;
?>
<html>
<html>
<body>
<?
      a = 3;
      b = 1;

      (: 1. a + b :)
      c = a + b;
      if ( c == 4 ) {   print ("1. 3 + 1  \t\t-> 4 \t\t accepted");     }
      else { print ("1. 3 + 1  \t\t-> " + c + " \t\t failed"); }

      (: 2. a - b :)
      c = a - b;
      if ( c == 2 ) {   print ("2. 3 - 1  \t\t-> 2 \t\t accepted");     }
      else { print ("2. 3 - 1  \t\t-> " + c + " \t\t failed"); }

      (: 3. a * b :)
      c = a * b;
      if ( c == 3 ) {   print ("3. 3 * 1  \t\t-> 3 \t\t accepted");     }
      else { print ("3. 3 * 1  \t\t-> " + c + " \t\t failed"); }

      (: 4. a / b :)
      c = a / b;
      if ( c == 3 ) {   print ("4. 3 / 1  \t\t-> 3 \t\t accepted");     }
      else { print ("4. 3 / 1  \t\t-> " + c + " \t\t failed"); }

      (: 5. b + a / b - a :)
      c = a + a / b - b;
      if ( c == 5 ) { print ("5. 3 + 3 / 1 - 1  \t-> 5 \t\t accepted"); }
      else { print ("5. 3 + 3 / 1 - 1 \t-> " + c + " \t\t failed"); }

?>
</body>
</html>
</html>
```

test.assign

```
<?
      int i;
      string s;
      boolean b1;
      boolean b2;
?>
<html>
<html>
<body>
<?
      (: 1. Assign integer :)
      i = 1;
      if ( i == 1 ) { print ("1. Assign i = 1 \t-> 1 \t\t accepted");   }
      else { print ("1. Assign i = 1 \t-> " + i + " \t\t failed"); }

      (: 2. Assign string :)
      s = "Hello";
      if (s == "Hello") { print("2. Assign s = \"Hello\"\t->\"Hello\"\t
            accepted"); }
      else { print ("2. Assign s = \"Hello\"\t-> \"" + s + "\" \t failed"); }

      (: 3. Assign Boolean  :)
      b1 = true;
      if (b1 == true) { print("3. Assign b1 = true\t-> true \t accepted"); }
      else { print ("3. Assign b1 = true \t-> " + b1 + " \t failed"); }
?>
</body>
</html>
</html>
```

test.boolean

```
<?
      boolean b1;
      boolean b2;
?>
<html>
<html>
<body>
<?
      (: 1. b1 = true :)
      b1 = true;
      if ( b1 == true ) { print ("1. b1 = true  \t\t-> true \t accepted"); }
      else { print ("1. b1 = true \t\t-> " + b1 + " \t failed"); }

      (: 2. b2 = false :)
      b2 = false;
      if ( b2 == false ) { print("2. b2 = false \t\t-> false \t accepted");}
      else { print ("2. b2 = false  \t\t-> " + b2 + " \t failed"); }
?>
</body>
</html>
</html>
```

test.comment

```
<?
      int a;
?>
<html>
<html>
<body>
<?
      (: 1. Comment Block :)
      (: print ("1. Commented Block \t-> visible \t failed"); :)
      print ("1. Commented Block  \t-> invisible \t accepted");
?>
</body>
</html>
</html>
```

test.comp

```
<?
      boolean b;
      int i;
      int j;
?>
<html>
<html>
<body>
<?
      (: 1. true && true :)
      if( true && true ) { print("1. true && true \t-> true \t accepted"); }
      else { print("1. true && true \t-> false\t failed"); }

      (: 2. true && false :)
      if( true && false ) { print("2. true && false \t-> true \t failed"); }
      else { print("2. true && false \t-> false\t accepted"); }

      (: 3. false && true :)
      if( false && true ) { print("3. false && true \t-> true \t failed"); }
      else { print("3. false && true \t-> false\t accepted"); }

      (: 4. false && false :)
      if( false && false) { print("4. false && false \t-> true \t failed"); }
      else { print("4. false && false \t-> false\t accepted"); }
?>
</body>
</html>
</html>
```

test.dop

```
<?
      int a;
      int b;
?>
<html>
<html>
<body>
<?
      (: 1. a++ :)
      a = 1;
      a++;
      if ( a == 2 ) {   print ("1. 1++  \t\t-> 2 \t\t accepted"); }
      else { print ("1. 1++  \t\t-> " + a + " \t\t failed"); }

      (: 2. b-- :)
      b = 1;
      b--;
      if ( b == 0 ) {   print ("2. 1--  \t\t-> 0 \t\t accepted"); }
      else { print ("2. 1--  \t\t-> " + a + " \t\t failed"); }
?>
</body>
</html>
</html>
```

test.for

```
<?
int i;
?>
<html>
<html>
<body>
<?
      for( i = 0; i <= 5; i++ ) {
            print(i);
      }
?>
</body>
</html>
</html>
```

bell.for

```
<html>
<body>
0
1
2
3
4
5
</body>
</html>
```

test.if

```
<?
      boolean b1;
      boolean b2;
?>
<html>
<html>
<body>
<?
      (: 1. if( true )  :)
      b1 = true;
      if ( b1 ) { print ("1. if(true) \t\t-> true \t accepted");  }
      else { print ("1. if(false) \t\t-> false \t failed"); }

      (: 2.if( false ) :)
      b2 = false;
      if ( b2) {print ("2. if(false) \t\t-> true \t failed");     }
      else { print ("2. if(false) \t\t-> false \t accepted"); }
?>
</body>
</html>
</html>
```

test.int

```
<?
      int a;
?>
<html>
<html>
<body>
<?
      (: 1. a = positive :)
      a = 1;
      if ( a == 1 ) {   print ("1. a = 1  \t\t-> 1 \t\t accepted");     }
      else { print ("1. a = 1  \t\t-> " + a + " \t\t failed"); }

      (: 2. a = zero :)
      a = 0;
      if ( a == 0 ) {   print ("2. a = 0  \t\t-> 0 \t\t accepted");     }
      else { print ("2. a = 0  \t\t-> " + a + " \t\t failed"); }

      (: 3. a = negative
      a = -1;
      if ( a == -1 ) {  print ("3. a = -1  \t\t-> -1 \t\t accepted");   }
      else { print ("3. a = -1  \t\t-> " + a + " \t\t failed"); }
      :)

      (: 4. a = double
      a = 1.1;
      if ( a == 1.1 ) { print ("4. a = 1.1 \t\t-> 1.1 \t\t accepted");  }
      else { print ("4. a = 1.1  \t\t-> " + a + " \t\t failed"); }
      :)
?>
</body>
</html>
</html>
```

test.opeq

```
<?
      int a;
?>
<html>
<html>
<body>
<?
      (: 1. 1 == 2 :)
      if ( 1 == 2 ) {   print ("1. 1 == 2 \t\t-> true \t failed"); }
      else { print ("1. 1 == 2 \t\t-> false \t accepted"); }

      (: 2. 2 == 2 :)
      if ( 2 == 2 ) {   print ("2. 2 == 2  \t\t-> true \t accepted"); }
      else { print ("2. 2 == 2  \t\t-> false \t failed"); }
```

```
        (: 3. "hi" == "hello" :)
        if ( "hi" == "hello" ) { print ("3. \"hi\" == \"hello\" \t-> true \t
                                failed"); }
        else { print ("3. \"hi\" == \"hello\" \t-> false \t accepted"); }

        (: 4. "hi" == "hi" :)
        if ( "hi" == "hi" ) { print ("4. \"hi\" == \"hi\" \t-> true \t
                                accepted"); }
        else { print ("4. \"hi\" == \"hi\" \t-> false \t failed"); }

        (: 5. true == false :)
        if ( true == false ) { print("5. true == false \t-> true \t failed"); }
        else { print ("5. true == false \t-> false \t accepted"); }

        (: 6. true == true :)
        if ( true == true ) { print("6. true == true \t-> true \t accepted"); }
        else { print ("6. true == true \t-> false \t failed"); }

        (: 7. 1 != 2 :)
        if ( 1 != 2 ) { print ("7. 1 != 2 \t\t-> true \t accepted"); }
        else { print ("7. 1 != 2 \t\t-> false \t failed"); }

        (: 8. 2 != 2 :)
        if ( 2 != 2 ) {   print ("8. 2 != 2  \t\t-> true \t failed"); }
        else { print ("8. 2 != 2  \t\t-> false \t accepted"); }

        (: 9. "hi" != "hello" :)
        if ( "hi" != "hello" ) { print ("9. \"hi\" != \"hello\" \t-> true \t
                                accepted"); }
        else { print ("9. \"hi\" != \"hello\" \t-> false \t failed"); }

        (: 10. "hi" != "hi" :)
        if ( "hi" != "hi" ) { print ("10. \"hi\" != \"hi\" \t-> true \t
                                failed"); }
        else { print ("10. \"hi\" != \"hi\" \t-> false \t accepted"); }

        (: 11. true != false :)
        if ( true != false ) {  print ("11. true != false \t-> true \t
                                accepted"); }
        else { print ("1. true != false \t-> false \t failed"); }

        (: 12. true != true :)
        if ( true != true ) { print ("12. true != true \t-> true \t failed"); }
        else { print ("12. true != true \t-> false \t accepted"); }

?>
</body>
</html>
</html>
```

test.print

```
<?
      int i;
      string s;
      boolean b;
?>
<html>
<html>
<body>
<?
      i = 1;
      s = "Hello";
      b = true;

      print(i);
      print(s);
      print(b);
      print("Hello");
?>
</body>
</html>
</html>
```

bell.print

```
<html>
<body>
1
Hello
true
Hello
</body>
</html>
```

test.rel

```
<?
      string s1;
      string s2;
      boolean b1;
      boolean b2;
?>
<html>
<html>
<body>
<?
      s1 = "hello";
      s2 = "hi";
      b1 = true;
      b2 = false;

      (: 1. 1 < 2 :)
      if ( 1 < 2 ) {    print ("1. 1 < 2 \t\t-> true \t accepted"); }
      else { print ("1. 1 < 2 \t\t-> false \t failed"); }
```

```
      (: 2. 2 < 1 :)
      if ( 2 < 1 ) {    print ("2. 2 < 1  \t\t-> true \t failed"); }
      else { print ("2. 2 < 1  \t\t-> false \t accepted"); }

      (: 3. 1 > 2 :)
      if ( 1 > 2 ) {    print ("3. 1 > 2  \t\t-> true \t failed"); }
      else { print ("3. 1 > 2 \t\t-> false \t accepted"); }

      (: 4. 2 > 1 :)
      if ( 2 > 1 ) {    print ("4. 2 > 1  \t\t-> true \t accepted"); }
      else { print ("4. 2 > 1  \t\t-> false \t failed"); }

      (: 5. 1 <= 2 :)
      if ( 1 <= 2 ) {   print ("5. 1 < =2 \t\t-> true \t accepted"); }
      else { print ("5. 1 <= 2 \t\t-> false \t failed"); }

      (: 6. 2 <= 1 :)
      if ( 2 <= 1 ) {   print ("6. 2 <= 1  \t\t-> true \t failed"); }
      else { print ("6. 2 <= 1  \t\t-> false \t accepted"); }

      (: 7. 2 <= 2 :)
      if ( 2 <= 2 ) {   print ("7. 2 <= 2  \t\t-> true \t accepted"); }
      else { print ("7. 2 <= 2 \t\t-> false \t failed"); }

      (: 8. 1 >= 2 :)
      if ( 1 >= 2 ) {   print ("8. 1 >= 2  \t\t-> true \t failed"); }
      else { print ("8. 1 >= 2 \t\t-> false \t accepted"); }

      (: 9. 2 >= 1 :)
      if ( 2 >= 1 ) {   print ("9. 2 >= 1  \t\t-> true \t accepted"); }
      else { print ("9. 2 >= 1  \t\t-> false \t failed"); }

      (: 10. 2 >= 2 :)
      if ( 2 >= 2 ) {   print ("10. 2 >= 2  \t\t-> true \t accepted"); }
      else { print ("10. 2 >= 2  \t\t-> false \t failed"); }
?>
</body>
</html>
</html>
```

test.return

```
<?
      int i;
      string s;
      boolean b;
      fxn int testReturnInt() {
            i = 3;
            return i;
      }
      fxn string testReturnString() {
            s = "Hi";
            return s;
      }
      fxn boolean testReturnBool() {
            b = true;
            return b;
      }
      int returni;
      string returns;
      boolean returnb;
?>
<html>
<html>
<head>
<title>Hello World Example</title>
</head>
<body>
<?
      (: 1. Return an int :)
      returni = testReturnInt();
      if ( returni == 3 ) { print("1. return int \t\t-> works \t accepted");}
      else { print ("1. return int \t\t-> " + returni + " \t failed"); }

      (: 2. Return a string :)
      returns = testReturnString();
      if ( returns == "Hi" ) { print ("2. return string \t-> works \t
                              accepted");      }
      else { print ("2. return string \t-> " + returns + " \t failed"); }

      (: 3. Return a boolean :)
      returnb = testReturnBool();
      if ( returnb == true ) { print ("3. return boolean \t-> works \t
                              accepted");      }
      else { print ("3. return boolean \t-> " + returnb + " \t failed"); }
?>
</body>
</html>
</html>
```

test.string

```
<?
      string s;
?>
<html>
<html>
<body>
<?
      (: 1. Assign string :)
      s = "Hello";
      if ( s == "Hello") {print ("1. s = \"Hello\"  \t-> \"Hello\" \t
accepted"); }
      else { print ("1. s = \"Hello\"  \t-> \"" + s + "\" \t failed"); }
?>
</body>
</html>
</html>
```

test.while

```
<?
      boolean b;
?>
<html>
<html>
<body>
<?
      (: 1. while(true) :)
      b = false;
      while(b) { print ("1. while(false) \t\t-> false \t accepted"); b =
                  false; }
      b = true;
      while(b) { print ("1. while(true) \t\t-> true \t accepted"); b = false;
}
?>
</body>
</html>
</html>
```