Report on CHAD

Designed, Implemented and Documented by:
Haronil Estevez, Diana Jackson, Catherine MacInnes, Adam Rosenzweig

Table of Contents

WHITE PAPER

Language Overview
The instruction of computer science is often hampered by the abstract nature of programming. Data and data structures are represented in programs with formal language and some students have difficulty visualizing what is actually going on. This problem is made worse when the data structures are manipulated in algorithms, even ones as simple a searches and sorts. Instructors are often reduced to drawing crude illustrations on the board or even calling on students to act as array elements. It is our belief that many instructors would appreciate a way to easily illustrate these data structures and algorithms in the context of a program which could then be included as part of a class demonstration. We hope to implement some of the more common complex data structures in a language that is syntactically and semantically easy to extend to other data structures

Goal
The goal of our language is to allow instructors to quickly and easily write algorithms pertaining to the arrays, queues and stacks which are then illustrated through the interpreting process. To this end, CHAD provides a set of basic functions for each of the native data types which allow them to be easily manipulated in the context of the language and which the interpreter then knows how to illustrate.

Easy to Use
One of the main goals of this language is to make the illustration of data structures and algorithms easy. To this end we will implement a syntactically simple language sacrificing power for simplicity. The language will be able to express certain data structures and algorithms very succinctly at the expense of not being able to express other data structures at all. We will also maintain a syntactic style that will be familiar to users of other common programming languages.

Complex Native Data Types
Programming in our language will be oriented around a small number of native types which include the complex data types arrays, queues, and stacks. In the future the language could easily be extended to include other complex types such as linked lists, or trees.

Portable
One of the things which is most important in terms of actually making CHAD useful is that it will be extremely portable. Instructors will be able to run CHAD programs anywhere and can even make them available for students online. This is possible because the CHAD interpreter will be written in Java, and will be the only thing necessary to run a CHAD program.

Language Tutorial

CHAD Tutorial

Section I (a). Basic Operators

Provided below is a table listing all of the supported CHAD operators along with a description.

Operator	Description	Examples
Assignment (=)	Assigns a value to a variable	int i = 0; string s1 = “string”;
Addition (+)	Adds integers/Concatenates strings	int j = 1; string s2 = “s”; int k = j + i; (k = 1) string s3 = s1 + s2; (s3 = “strings”)

Subtraction (-)	Subtracts integers/Denotes Negation	k = k – j; (k = 0)
		k = -1;

Multiplication (*)	Multiplies integers	k = 3 * 4; (k = 12)
Division (/)	Divides integers	k = 12 / 3; (k = 4)
Increment (++)	Increases integer variable by 1	k++; (k = 5)
Decrement (--)	Decreases integer variable by 1	k--; (k = 4)
Parentheses (())	Specifies which operations to evaluate first	k = (5 + 4) * 2; (k = 18 because the “5+4” is evaluated first, then multiplied by 2)

Comparison Operators	**Description**	Examples
*In each of these cases (as well as cases to follow in the Logical Conjunction Operators section), 0 is returned if the comparison yields a false result. 1 is returned if it yields a true one. Typically, these comparison operators will be used in the context of an **if-statement**, but for the purpose of demonstrated its use, that notion is not demonstrated here.		
IsEqualTo (==)	Compares both sides of operator for equality	k == 18 (returns 1)
IsGreaterThan (>)	Compares both sides of operator to see if the right is greater than the left	k > 19 (returns 0)
IsLessThan (<)	Compares both sides of operator to see if the right is less than the right	k < 19 (returns 1)
IsGreaterThanOrEqualTo (>=)	Compares both sides of operator to see if the right is greater than or equal to the left	k >= 15 (returns 1)
IsLessThanOrEqualTo (<=)	Compares both sides of operator to see if the right is lessgreater than or equal to the left	k <= 15 (returns 0)
Logical Conjunction Operators	**Description**	**Examples**
AND	Signifies logical conjunction of two expressions	(K > 15) AND (K < 20) (true)
OR	Signifies logical disjunction of two expressions	(K > 15) OR (K < 1) (true)
NOT	Signifies logical negation of an expression	NOT (K == 15) (true)

Section I (b). Control Statements

For-Loops allow a block of code to be run multiple times. The syntax is as follows:

```
for (int i=0; i<10; i++)
        statements
endfor
```

The first parameter takes an integer for which the iteration should begin. The second parameter specifies when the iteration will end. In this case, the range will be from zero to 9. The third parameter specifies that the integer variable will increment with each iteration. Next, "statements" represent whatever action you want to occur with each iteration of this loop. Finally, a for-loop must be ended with the keyword **endfor**.

If-Then-Else Statements allow statements to be executed provided that a test expression proves true. Let's say for example that you have the numbers five and six. You would like to assign the variable, i, to be equal to the greater of the two. Here is the code:

```
int int5 = 5;
int int6 = 6;
int i;
```

```
if (int5 > int6)
        i = int5;
endif
else
        i = int6;
endelse
```

Additionally, you can use an **elseif** statement if you want to check multiple expression prior to resulting to the execution of the statements in the body of the **else** block. Finally, an important point to mention is that regardless of which type you use, each must be ended with the appropriate keyword: **if** -> **endif**, **elseif** -> **endelseif**, **else** -> **endelse**

Section II (a). Integers

Integers must be in the range of –999 to 999. The syntax along with the relevant operators has been demonstrated previously. (See Section on Operators) One function associated with integers is the **min()** and **max()** function. They both take two integer arguments and return the smaller or greater of the two, respectively. (Note: These two functions may also be called with strings)

Section II (b). Strings

Strings may contain up to 30 characters and be alphanumeric. The syntax along with the relevant operators has been demonstrated previously. (See Section on Operators)

Section II (c). Arrays

Arrays may hold integer or string values. It is declared with a fixed length by using the following syntax:

```
array[int, 4] myArray;
```

This creates an array named myArray that will hold up to 5 integer values. To assign values to the array, you can access each element via its index:

```
myArray[0] = 1;
```

Or, you could have specified it upon declaration:

```
array[int, 4] myArray = {1,2,3,4,5};
```

Array Length: You can find the length of the array by invoking the length method: myArray.length(); which in this case would return 5.

Swapping Indices: There is also a **swap(int index1, int index2)** method, which takes as parameters, two integers representing two indices in the array. By calling this method, the data contained in each of these cells are switched.

Sorting: Finally, arrays can be sorted via the sortAZ() and sortZA() methods. Both have two parameters, a beginning index and an ending index. The indices beginning at the specified begin index and ending at the specified ending index are sorted in ascending or descending order depending on the method evoked.

Section II (d). Queues

Queues can hold integer or string values. They are declared by using the following syntax:

```
queue[int] myQueue;
```

To add an element to the queue, use the enqueue() method with:

```
myQueue.enqueue(1);
```

To retrieve an element from the queue, use the dequeue() method:

```
int i = myQueue.dequeue();
```

Section II (e). Stacks

Stacks can hold integer or string values. They are declared by using the following syntax:

```
stack[int] myStack;
```

To add an element to the stack, use the push() method with:

```
myStack.push(1);
```

To retrieve an element from the stack, use the pop() method:

```
int i = myStack.pop();
```

Section III. Writing your own Functions

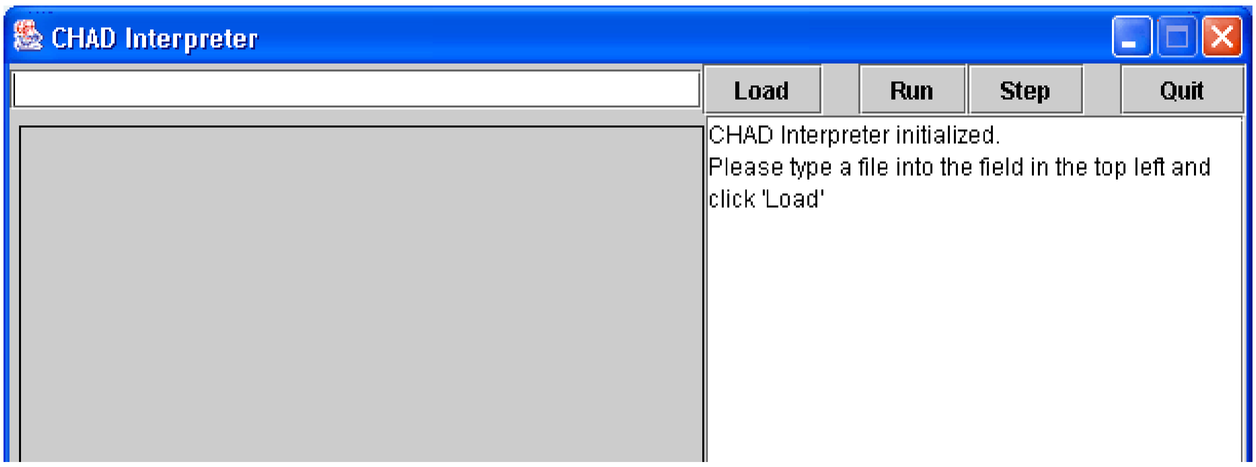
If you want to be adventurous, you can also write your own functions. Here is the syntax that you should follow:

```
function returnType name(parameters)
        body of statements
        return statement
endFunction
```

The return type of any CHAD function can be a string, integer, or array (array[int] or array[string]) Your function may take in as many parameters as you wish. Functions must have a unique name.

Section IV. Displaying with CHAD GUI

Here is a snapshot of the CHAD GUI:



CHAD Interpreter initialized.
Please type a file into the field in the top left and click 'Load'

The path of your CHAD file should be typed into the text box in the top left corner. After you load it, you may **run** the entire program or opt to **step** through it. Note: By default, variables are not included in the graphical display. If you want your program to be animated, you must use the method **show(variableName)** to make it appear. Likewise, to remove it, use the method **hide(variableName)** to make it disappear. Only arrays, queues, and stacks can be animated.

Language
Reference Manual

INTRODUCTION

The CHAD programming language is a limited purpose programming language designed to allow teachers and students to quickly code algorithms involving arrays, queues and stacks and see graphically how their data is being manipulated. CHAD is an interpreted language in which the interpreter outputs a graphical representation of the data structures used in the program as well as performing any calculations. The goal of CHAD is to make visualization of algorithms easier by providing a way to easily and compactly create graphical representations of them.

STRUCTURE OF THE PROGRAM

A CHAD language program consists of a series of variable declarations followed by a series of statements to be executed followed by a series of optional user-defined functions. User-defined functions consist of the function keyword followed a header and a body and are ended with the endfunction keyword. The header of the function is its return type followed by it's name and an optional set of comma delineated arguments along with their types enclosed in parenthesis. The body consists of a series of variable declarations followed by a series of statements.
e.g

```
function string echo(string x)
return x;
end function
```

Note, within the main body, or a function block, any declarations must precede all statements.

VARIABLES AND DECLARATION

The CHAD language has 5 native data-types. They are integers, strings, arrays, queues and stacks. All variables must be declared before they are used and all declarations are to be located at the beginning of a program or function definition. All variables are local to the block in which they are defined. Variables defined within a function can be used only within that function and variable defined in the main body of the program (outside of any function declaration) cannot be used by any functions in the program. Any parameters needed by a function must be explicitly passed to that function and functions can not have side effects which alter the value of variables outside their scope. The only exception to this is some built in functions which can be called on a variable using a variable.function() syntax which gets the variable as a reference and is able to modify the variable it is given as well as return another value. This will be discussed further when we discuss function calls. When variables are declared they can be declared with or without initializing them. However, initialization is required before a variable can be used. The syntax for initialization of a variable is exactly the same as the assignment syntax. Variable and function names are identifiers (see later definition of an identifier).

Integers

The integer data type in CHAD supports integers values from –999 to 999

the declaration of an integer variable consists of the keyword int followed by the name of the variable.

Strings

The string data type in CHAD supports alphanumeric strings up to 30 characters in length. For a list of characters to be interpreted as a string literal by CHAD it must be enclosed in double quotes and cannot include any non-alphanumeric characters. The declaration of a string variable consists of the keyword string followed by the name of the variable.

Arrays

The array data type in CHAD supports arrays of either string or integer types with a fixed length chosen at declaration. The declaration of an array consists of a statement such as array(int, 10) where the first argument is the type of the values to be stored in the array and the second argument is the length of the array. The index of the array begins at 0 and goes to n-1 where n is the length of the array. The length of an array is fixed as soon as it is created. Trying to access an element of the array that does not exist causes an error by the compiler.

Queues

The queue data type supports queues of either string or integer types. Queues are dynamically assigned memory by the interpreter and therefore have no fixed length. However the programmer should use caution to avoid overflowing available memory.
The declaration of a queue consists of the queue keyword followed by the type of the queue in parenthesis followed by the name of the queue. e.g. queue(int) myQueue

Stacks

The stack data type supports support stacks of either string or integer types. Stacks are dynamically assigned memory as queues are and therefore have no fixed size. The declaration of a stack consists of the stack keyword followed by the type of the stack in parenthesis and the name of the stack. e.g. stack(int) myStack.

BUILT IN OPERATORS

ASSIGNMENT (=)

The assignment operator,. = is used to give a particular value to a variable after that variable has been declared. The syntax is variable = value where value can be another variable of the same type. There are no conversion operators in CHAD, assignments must be of the same type of variable or value.

Integers and Strings

These use only basic assignments as shown above. The value of an integer must be a digit between -999 and 999. The value of a string must be a series of alphanumeric characters enclosed in double quotes to distinguish it from an identifier.

Arrays.

There are two types of assignments to arrays. The first treats each cell in the array individually. The syntax to extract a single cell from an array is the name of the array followed by the index of the desired value in square brackets e.g. myArray[5] accesses the 6th element of myArray. This syntax is treated as an integer or string depending on the type of the array. If an attempt is made to access an index which is beyond the length of the array an error will be raised. The second method of assignment to an array handles multiple values at the same time. This syntax is variable = {vals} where vals is a comma separated list of n values to be put into the first n cells of the array. Any values currently in these cells will be discarded in the process of the assignment. If you attempt to assign more values than there are cells in the array an error will be raised. An array can also be assigned to another array of the same type. However if an array is assigned to an array which is shorter in length it will raise an error.

Queues and Stacks

There is no assignment syntax for queues and stacks. Values are added and subtracted through special access methods which will be defined later.

ADDITION (+)

The addition operator can be applied to integers and strings with the syntax value + value. Where values can be either laterals variable names. The results can then be assigned to a variable as defined above. The two values must be of the same type and if they are assigned to a variable that variable must also be of the same type. If integers are added, they are arithmetically added. If two strings are added they are concatenated.

SUBTRACTION (-)

Is the usual arithmetic operation and can only be applied to integers. Can also be applied to a single integer value in prefix notation e.g. -8 in which case it changes the sign of the integer to which it is applied.

DIVISION (/)

Is the usual arithmetic operation and can only be applied to integers.

MULTIPLICATION (*)

Is the usual arithmetic operation and can only be applied to integers.

INCREMENT (++)

Can be applied in postfix notation to an integer variable to increase the value by 1.

DECREMENT (–)

Can be applied in postfix notation to an integer variable to decrease the value by 1.

PARENTHESIS (())

Expressions that are placed within parenthesis should be evaluated before any outside operations are applied to them.

The precedence of the above operators is negation and increment and decrement, multiplication and division, addition and subtraction, and then assignment. If another order is desired parenthesis should be used. Within any level of precedence operators are applied from left to right.

COMPARISON OPERATORS

Is Equal to (==)

e.g variable == value compares the values on the two sides of the operator. Returns 0 if the variable are not equal and 1 if the variables are equal. Can only be used on integers and strings. If the types of the two values are different the function will always return 0. For integers the variables are considered equal if they have the same numerical value. In the case of strings, the variables are considered equal if they consist of the same series alphanumeric characters.

Is Greater Than (>)

Can be used on either integers or strings but the behavior is undefined if applied to unmatched types. The syntax is var1 > var2 and compares var1 and var2. Returns 1 if var1 is larger and 0 otherwise. Integers are compared using their numeric values. Strings are compared lexically according to the ASCII values.

Is Less Than (<)

Behaves exactly like greater than except return the opposite value in all cases except where the values are equal.

Greater Than or Equal To (>=)

Behaves like greater than but returns 1 in the case that the two values are equal.

Less Than or Equal To (<=)

Behaves like less than but returns 1 in the case that the two values are equal.

LOGICAL CONJUNCTION OPERATORS

AND

The keyword AND is used to signify the logical conjunction of two expressions. Returns 1 if the value of both expressions are 1 and 0 otherwise.

OR

The keyword OR is used to signify the logical disjunction of two expressions. Returns 0 if the value of both expressions are 0 and 1 otherwise.

NOT

The keyword NOT is used to signify the logical negation of an expression. Returns1 if the value of the expression is 0 and 0 if the value of the expression is 1.

In general all expressions in a complex logical expression are evaluated regardless of whether the outcome could be known at an earlier point. The order of operations for logical expressions is the comparison operators evaluated from left to right followed by the conjunction operators evaluated from left to right. Parenthesis should be used to enforce any other desired precedence relationships.

CONTROL STATEMENTS

for loops

The for loop syntax allows a block of code to be run multiple times. The syntax for the for loop is

```
for(var; test; change;)
statements
endfor
```

where end and endfor are keywords, var represents the loop control variable, test represents the test which determines whether or not the body of the loop will be executed. If the test evaluates to true the body is executed otherwise it is skipped. Change represents a statement which changes the value of the control variable each time the loop is executed. The endfor keyword indicates the end of the for loop body. When the test evaluates to zero execution of the program picks up at the statement following the endor keyword.

IF-THEN-ELSE

The IF-THEN-ELSE syntax allows statements to be executed or not depending on upon the value of a test statement. The syntax is:

```
if(test)
statements
endif
elseif(test)
statements
endelseif
else
statements
endelse
```

if, endif, elseif, endelseif ,else, endelse are keywords of the language. Test represents a conditional statement. If test evaluates to true the statements between if and endif are executed, otherwise every subsequent elseif conditional statetement is evaluated. The first true elseif statement is executed. If no elseif statement evalutes to true or if no elseif statement exist, then the statements between else and endelse are executed. After these statements are executed the program continues at the statement following the endelse keyword. The block:

```
else
statements
endelse
```

along with the block:

```
elseif(test)
statements
endelseif
```

are optional. If any one of these blocks is left off and the test evaluates to false the program begins execution at the statement following the endif.
Control statements can be arbitrarily nested within each other. If this is the case the endfor, endif, endelseif, and endelse keywords match the closest for, if or else

respectively.

BUILT IN FUNCTIONS

There are two kinds of built in functions for CHAD. The first are standard function which take arguments and return based on them. The second are specific to a data type and are called on a particular variable by using the syntax of variable.function(args) where function is the name of the function and args is a comma separated list of arguments which are passed to the function. The built in functions are defined to work on specific data types.

min

min(var1, var2) returns the argument with the least value. Min can be called with either integers or strings. It causes an error if called with one of each. The comparisons are done as if the < operator were called.

max

max(var1, var2) returns the argument with the greater value. Max can be called with either integers or strings but causes an error if called with one of each. The comparisons are done as if the > operator wrer called.

show

Takes as an argument any variable and makes that variable part of the display.

hide

Takes as an argument any variable already part of the display and removes it from the display.

print

Takes as an argument a string variable or literal and types it in the comments part of the display. Allows an instructor to make notes about an algorithm while it is executing.

Arrays

myArray.swap(index1, index2) swaps the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘,’, Has no return value. myArray.sortAZ(index1, index2) puts the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘.’ in increasing order in the locations indicated. Returns 1 if a switch was made and 0 otherwise. myArray.sortZA(index2, index1) puts the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘.’ in decreasing order in the locations indicated. Returns 1 if a switch was made and 0 otherwise. myArray.length() returns the integer length of the array.

Stacks

myStack.pop() removes the top value from the given stack and returns it. The return type is the same as the type of the stack. myStack.push(arg) adds the value represented by arg to the given stack. Returns 1 if successful, 0 otherwise. Arg must be of the same type as the stack.

Queues

myQueue.enqueue(arg) adds the value given by arg to the given queue. Returns 1 if successful, 0 otherwise. Arg must be of the same type as the queue. myQueue.dequeue() takes the value from the front of the queue, removes it and returns it. The return type is the same as the type of the queue.

DEFINING YOUR OWN FUNCTIONS

User defined functions can be added to the end of a program.
The structure of a user defined function is:

```
function returnType myFunction(args)
statements
return val;
endfunction
```

where function and endfunction are keywords, returnType is the type that the function returns myFunction represents the name of the function and args represents a comma separated list of arguments each preceded by their types. Return is also a keyword which must be the last statement in the functiond definition and val represents the value to be returned by the function. The return statement must be the last statement in the function definition and is required. If the return type of the function is void no value follows the return statement.

e.g.

```
function int min(int x, int y)
int temp;
if(x < y)
temp = x;
endif
else
temp = y;
endelse
return temp;
endfunction
```

Assignments and function calls must be ended with semi-colons. For, if and else statements must be ended with their appropriate endfor endif or endelse.

PROJECT PLAN

The project will be implemented in parts with different people being responsible for different parts of the project. The first part of the project for which Catherine is primarily responsible is the determination of what principles the language will embody and the documentation of those decisions. This part of the project will be done in large part when the White Paper is completed on September 23rd and will be at least tentatively finished by the completion of the language reference manual on October 28th. These decisions will be made in consultation with the entire groups but especially Haronil who will be responsible for the syntactic and semantic analysis of input programs up to the creation of the abstract syntax tree. This part of the project should be close to completion by the middle of November. Adam will be responsible for the front end of the interpreter including the production of the graphical interface and output. This should be done in large part by Thanksgiving. At this point Diana will start the main portion of the testing which will continue up until the time that the source code is handed in the project considered completed on December 17th . Constant revisions will be made to all parts of the project as things progress and especially during the testing phase when all kinds of unforseen errors can come up. Of course this process continues as long as anyone is using the language if that is beyond the scope of this project.

Architectural Design

The CHAD Interpreter provides a visualization of basic data structures and algorithms. First, a user loads a CHAD script by entering the path to the file and clicking the "Load" button. This triggers the java.io functions that will locate and read in the file data. They pass the character stream onto the CHADLexer, which was created using ANTLR.

CHADLexer transforms the character stream into tokens, and will report an error in the GUI if anywhere within the character stream is a sequence of characters that cannot form a valid token (for example, a character that is not allowed in the language, such as '$'). If it is successful, CHADLexer passes the tokens on to CHADParser.

CHADParser, which was also created using ANTLR, checks the syntax of the script. It makes sure that the tokens come in a valid order - that assignments are made to variables rather than to expressions, and so on. If it encounters an invalid sequence of tokens, it will report an error in the GUI. If it is successful, it will generate an Ab stract Syntax Tree and hand that off to the StaticCHADWalker.

StaticCHADWalker, another ANTLR creation, is the static semantic checker. It checks the Abstract Syntax Tree generated by CHADParser for errors such as types that do not match - comparing an int and a string, for example. It also checks that variables have been declared, functions defined, and that scopes are valid. If it encounters any errors in the semantics of the program, then it reports the errors in the GUI. If it does not encounter errors, then it reports in the GUI that the source is valid.

At this point, it is up to the user to press either the "Run" button or the "Stop" button.Regardless of which button is pressed, the Interpreter will pass the checked AbstractSyntax Tree to CHADWalker, the final ANTLR creation. CHADWalker walks the Abstract Syntax Tree, but unlike StaticCHADWalker, it interprets the script as it walks the tree. That is to say, it executes the code.As it executes the script, CHADWalker builds up a Java Vector containing one item for each change that the script causes in the GUI. This means that print statements, show statements, hide statements, and all statements that modify a variable that is currently being shown all add to the Vector.

When CHADWalker has finished executing the script, the GUI is given access to the Vector. If the "Run" button was pressed, then the GUI will step through the changes at a delay of one second between each graphical update. If the "Step" button was pressed, the GUI will make the first graphical update, and make one more update for each subsequent press of the "Step" button, until all elements of the Vector have been displayed.

Due to the nature of this architecture, if the script has a non-terminating loop, the CHADWalker will not terminate, and thus the GUI will never begin to perform graphical updates. Unfortunately, it is not possible for the Interpreter to identify situations such as this to avoid infinite looping.

If a run-time error is encountered (for example, a manipulation on an integer that forces it to exceed the bounds of [-999, 999], a string manipulation that extends it beyond the 30 character limit, a push or enqueue onto a full stack or queue, or a pop or dequeue off of an empty stack or queue), the error is reported in the GUI, and the CHADWalker either ignores the offending statement (for push/enqueue onto full structures, and the removal part of pop/dequeue off of empty structures), or sets the variable to its default (0 for integers, the empty string "" for strings). The value assigned on a pop/dequeue from an empty structure is likewise the default value of the appropriate type.

Test Plan

Introduction

In order to conduct effective testing, the LRM was re-examined to determine the exact capabilities and functionality of our language. A separate test script was created for each of our five native data types (integers, strings, arrays, queues, and stacks), user-defined functions, as well as a separate file testing the recognition of comments. Within these scripts were embedded tests that checked for proper usage of built-in operators and control statements. To generate the scripts that contained errors, typical errors were first added such as missing semicolons, incorrect spelling of keywords, etc. All other errors were derived by purposely defying the rules mentioned in the LRM

Part I. ANTLR Testing In order to test the parser and lexer, two separate test scripts were generated for each main component/data structure of our system as mentioned in the introduction. Input files without errors resulted in no output, while files containing errors specified the error and the line it originated from. The purpose of these tests were simply for the purpose of detecting errors within the grammar and making sure the errors detected errors in the program.

Part II. Static Walker Testing StaticWalker.java is a static semantic version of Walker.java. Testing was done on this class to ensure that the static semantics were properly checked. StaticWalker does the minimal interpretation needed to accomplish this. It prints out static semantic errors if there are any, but prints nothing out if the program is semantically correct. The test scripts used for this portion and the testing of the walker did not contain errors. Once these scripts ran successfully, we then added errors to check for proper functionality.

Part III. Walker Testing Walker.java prints what the parser sees, the AST tree, and what is inside of the function declarations. The same test cases were used. This testing phase was more along the lines of regression testing.

Conclusion Each of the test scripts was run by loading the file into CHAD. The same scripts were used in each testing phase with modifications only being made to check errors on the spot without saving changes to the file. Included as a part of this report are two informal reports StaticWalkerTestResults and WalkerTestResults that list the specific cases tested as well as problems/errors found. Testing was an iterative process. We tested after a series of changes were made to the code to ensure that errors were indeed fixed as well as to make sure that previous functionality still existed. As found errors became less frequent, no revisions were made to the informal reports. Instead, status emails were sent to the developers notifying them of the errors that needed to be fixed. Subsequently, the revised code was sent out…and the testing process was carried out again.

See Appendix A for a partial listing of Test Code.

Lessons Learned

In the process of writing CHAD we learned a lot. First, designing and implementing a language is not as easy as we thought it would be when we started. We started with a much more ambitious plan and had to scale back when we realized how long it was going to take. Along those lines we also realized that there are a lot more details than you are going to think about at any one time. No matter where in the process you are, something will come up that you haven't thought of yet, and when you start to think about how you are going to answer that issue, three more present themselves to you. The design process is simply never ending. Even as we were doing final debugging on the code there were things that came up that we went, why did we do this, or why didn't we do that. Along those same lines, you should always leave more time for a project than you think you need. Starting early and trying to keep ahead means that when you fall behind your own deadlines, at least you aren't behind the hard deadlines. There are also two big time savers that you can employ to try and keep yourself on schedule. The first is to plan carefully before you start writing any code. This is particularly important for the language design issues like variable scope which have to be taken into account in a lot of places. The second big time saver is to have your code print out the most specific error messages you can come up with. One of the most frustrating feelings is to not be able to decipher the error message that you wrote two days ago.

Another major component of this challenge, was to work together effectively as a group. No single person could have done this project in one semester, but the group also causes some problems of it's own. The group helps when you are trying to come up with ideas, or trying to think of all the problems that you can, but it often does not help in making final decisions. We actually didn't have trouble where two or more people had really strong feelings on something and bumped heads, more often, everything was willing to defer to someone else's judgement, and that is almost as bad. Also, although splitting up jobs and knowing who is responsible for what pieces is important, so is communication. There were times when we didn't know where other people were, or what decisions were being made. This is particularly important for when someone is not directly involved in one part of the work. The people who are involved in that part should make absolutely certain that everyone else knows what is going on. We had a lot of trouble with communication because we found it very difficult to all meet due to very different and busy schedules.

APPENDIX A

ANTLER TEST SCRIPTS

ERROR FREE SCRIPTS

1. Integers
2. Strings
3. Arrays
4. Stacks
5. Queues
6. Functions
7. Comments

SCRIPTS WITH ERRORS

1. Integers
2. Strings
3. Arrays
4. Stacks
5. Queues
6. Functions
7. Comments

STATIC WALKER TEST SCRIPTS

1. Integers
2. Strings
3. Arrays
4. Stacks
5. Queues
6. Functions
7. Comments
8. The Result of these test scripts in the static walker as of Dec. 6th by way of demonstration of the testing process.
8. The Result of these test scripts in the walker as of Dec. 6th by way of demonstration of the testing process.

Testing Integers

```
int a = 3;
int b = 4
int ab = min(a,b);
int ba = max(b,a);
int a = 3;
a = 5;
int c;
integer test = 4;
Int test2 = 4;
C = a + b - 0;
c=((a + b) - 0) % 7;
c += 2;
a++;
b--;
if (a > b) b = 10;
else if (a < b) a = 10;
else b = 0;
int d = c * a / 1;
int string = "Diana";
int d = 3.1;
int e = -1;
```

Testing Strings

```
string s1 = ”Diana”;
string s2 = "Jackson";
string s3 = "Catherine";
string s4;
string s5 = s1;
if (s1 == s3) s4 = ”girls”;
String concat = s1 + " " + s2;
```

Testing Arrays

```
array a[int, 10];
a.sortAZ(0,9);
array b[string, 2];
array f[string, 2] = {“one”,”two”};
a[0] = 0;
b[0] = ”string”;
if (f[0] == ”one” AND f[1] == ”two”) b[0] = "true";
a.swap(0,1);
array A[int, 5];
for (int i=0; i<5; i++)
        A[i] = i;
endfor
```

Testing Stacks

```
int a;
string s;
stack[int] myStack;
stack [string] myStack3;
myStack.push(1);
a = myStack.pop();
myStack.push(2);
myStack3.push(“string”);
s = myStack3.pop();
```

Testing Queues

```
int a;
queue[int] myQueue;
queue[string] myQueue3;
myQueue.enqueue(1);
myQueue. enqueue (2);
a = myQueue.dequeue();
a = myQueue. dequeue ();
myQueue3. enqueue (“Diana”);
myQueue3. dequeue ();
```

Testing Functions

```
int i;
int a = 3;
int b = 2;

function int min(int x, int y)
    int temp;
    if(x < y)
      temp = x;
    endif
    else
      temp = y;
    endelse
  return temp;
endfunction

I = min(a,b);
```

Testing Comments

```
//Will the program recognize that this line is a comment?
/*How about this one?*/
/*
   And multi
   line
   comments?
*/
```

Testing Integers

```
int a = 3;
int b = 4
int ab = min(a,b);
int ba = max(b,a);
int a = 3;
a = 5;
int c;
integer test = 4;
Int test2 = 4;
C = a + b - 0;
c=((a + b) - 0) % 7;
c += 2;
a++;
b--;
if (a > b) b = 10;
else if (a < b) a = 10;
else b = 0;
int d = c * a / 1;
int string = "Diana";
int d = 3.1;
int e = -1;
```

Testing Strings

```
string s1 = ”Diana”;
string s2 = 1;
string s3 = ’Diana’;
string s4 = ’s’;
string s5 = s;
if (s1 == s3) s4 = ”nothing”; endif
string s6 = ”text3”
String s = ”hello”;
String concat = s1 + s3;
```

Testing Arrays

```
array a[int, 10];
a.sortAZ(0,9);
a.sortAZ(0,10);
array b[string, 2];
array c[int, 20];
array d[int, 3];
array e[int, 3];
array f[string, 2] = {“one”,”two”};
array g[string, 2] = {“one”, “two”,”three”};
array h[int, 2] = {1,”two”};
a[0] = 0;
a[10] = 10;
b[0] = ”string”;
b[1] = 2;
if (g[0] == ”one” AND f[0] == ”one”) a[0] = 0;
c = a;
swap(0,1);

d = {1,2,3};
e = {1,2,3,4};
array A[int, 5];
for (int i = 0; i < 5; i++)
        A[i] = i;
endfor
for (int i = 0; i < 5; i++)
        A[i]=i;
```

Testing Stacks

```
int a;
string s;
stack[int] myStack;
stack myStack2[int];
stack [string] myStack3;
myStack.push(1);
a = myStack.pop();
myStack.push(2);
myStack.push(“string”);
a = myStack.pop();
myStack3.push(“string”);
s = myStack.pop;
```

Testing Queues

```
int a;
queue[int] myQueue;
queue myQueue2[int];
queue[string] myQueue3;
myQueue.enqueue(1);
myQueue. enqueue (2);
a = myQueue.dequeue();
a = myQueue. dequeue ();
a = myQueue. dequeue ();
myQueue3. enqueue (“Diana”);
myQueue3. enqueue (1);
```

Testing Functions

```
int i;
int a = 3;
int b = 2;

function int min(int x, int y)
    int temp;
    if(x < y)
      temp = x;
    endif
    else
      temp = y;
    endelse
  return temp;
endfunction

i = min(a,b);
```

Testing Comments

```
//Will the program recognize that this line is a comment?
/*How about this one?*/
/*
   And multi
   line
   comments?
*/
*/And improperly structured ones like this one?*/
```

```
// Testing Integers
//declarations
int a = 3;
int b = 4;
int c;
int d = c * a / 1;
int e = -1;

// expressions
int ab = min(a,b);
int ba = max(b,a);
a = 5;
c = a + b - 0;
c=((a + b) - 0) % 7;
c += 2;

if (a > b)
    b = 10;
endif
elseif (a < b)
    a = 10;
endelseif
else
    b = 0;
endelse
```

```
// Testing Strings
string s1 = "Diana";
string s2 = "Jackson";
string s3 = "Catherine";
string s4;
string concat = s1 + " " + s2;
string s5 = s1;
if (s1 == s3) s4 = "girls"; endif
if (s1==s3) s4="girls"; endif
```

```
// Testing Arrays
array[int, 10] a;
array[int, 5] A;
array[string, 3] b;
int i = 0;
int j=-1;
array[string, 2] f = {"one","two"};
for (i=9; i>=0; i--;)
        a[i]=10-i;
endfor
show(a);
a.sortAZ(0,9);
a.swap(4,5);
a.swap(7,8);
a[0] = 0;
b[0] = "string";
b[1] = "coat";
//show(b);
b.swap(0,1);
b.swap(1,0);
b.swap(0,1);
//if (f[0] == "one" AND f[1] == "two") b[0] = "true"; endif
for (i=0; i<4; i++;)
        A[i] = i;
endfor
```

```
// Testing Stacks
int a;
string s;
stack[int] myStack;
stack [string] myStack3;
show(myStack);
myStack.push(1);
a = myStack.pop();
myStack.push(2);
myStack3.push("string");
s = myStack3.pop();
```

```
// Testing Queues
int a;
string b;
queue[int] myQueue;
queue[string] myQueue3;
show(myQueue);
myQueue.enqueue(1);
myQueue. enqueue (2);
a = myQueue.dequeue();
a = myQueue.dequeue();
myQueue3. enqueue ("Diana");
b = myQueue3. dequeue ();
```

```
// Testing Functions

int i;
int a = 3;
int b = 2;
string matched;

string name1="Diana";
string name2="Jackson";
i = minimum(a,b);
matched=match(name1,name2);

function int minimum(int x, int y)
    int temp;
    if(x < y)
      temp = x;
    endif
    else
      temp = y;
    endelse
  return temp;
endfunction

function string match(string s1, string s2)
            string concat;
            string firstname="Diana";
            string lastname="Jackson";
            if (s1==firstname AND s2==lastname)
                        concat="true";
                        print(concat);
            endif
            else
                        concat="false";
            endelse
            return concat;
endfunction
```

```
// Testing Comments
//Will the program recognize that this line is a comment?
/* And these type of comments */
/*
        And
        multi
        line
        comments
*/


int i=0;
int j=i+1;
if (j==0) j=2; endif
```

Tested Cases for each Data Type

Listed below are each case that I tested for. If a problem was found, it is bolded.

Integers

- wrong type specification: int vs. Int
- missing semicolon at end of declaration
- operations on undeclared variable
- operations on declared variables with no definition (produced no error)
  - **If you try to perform an operation on several variables, and all have been declared but not necessarily defined, you get no error. For example: int a=3; int b=4; int c; int d=a*b+c; will not give an error.**
- missing parenthesis
- comparison operator found when assignment operator expected (a=b) vs. (a==b)
  - **Can we provide a meaningful error message when the user uses a comparative operator when an assignment operator is expected: (a=b) vs. (a==b)? Right now you just get an unexpected token/parsing error message**
- missing end(if/else/elseif)

Arrays

- sortAZ/ZA function for improper indices reference
  - **If you have an array[int, 10] and try to sortAZ with indices 0-10, there's no error, neither is there an error for 0-11**
- improper index reference
- assigning a string to an index of an integer array
  - **It does alert the user that they are not of the same type, but it says null and null are not of the same type.**
- adding elements of arrays
- swapping indices that don't exist
  - **For an array of 10 elements, you can swap 10 and 11 or 11 and 12, it doesn't check to make sure that the indices are actually in bound**

Comments - PROBLEMS!

- **I don't know why I kept getting errors with the comment file. I've attached it for you guys to look at. I got an unexpected character error**

Queues

- dequeueing more times that allowed

- enqueueing a string into an integer queue and vice versa

Stacks

- popping more times that allowed
- pushing a string into an integer queue and vice versa

Strings

- assigning integer to a string

Tested Cases for each Data Type

Listed below are each case that I tested for. If a problem was found, it is bolded.

Integers

- wrong type specification: int vs. Int
- missing semicolon at end of declaration
- operations on undeclared variable
- operations on declared variables with no definition (produced no error)
    - **If you try to perform an operation on several variables, and all have been declared but not necessarily defined, you get no error. For example: int a=3; int b=4; int c; int d=a*b+c; will not give an error.**
- missing parenthesis
- comparison operator found when assignment operator expected (a=b) vs. (a==b)
- missing end(if/else/elseif)
- **New Error:** assign a negative value to an int
    - **INTEGER ID: e**
    - **<AST>: unexpected AST node: UMINUS**
    - **e and null are not of the same type!!!**

Arrays – Keep getting null pointer exceptions, I can't figure out where the errors are

- can't reference array with a variable (if you have i defined as an int and arr as an array, you can't use arr[I] or you get an error
    - 14: line 1:283: expecting INTEGER, found 'i'
    - I changed it to be arr[0] and I got a java null pointer exception error
- sortAZ/ZA function for improper indices reference
    - **If you have an array[int, 10] and try to sortAZ with indices 0-10, there's no error, neither is there an error for 0-11**
- improper index reference
- assigning a string to an index of an integer array
    - **It does alert the user that they are not of the same type, but it says null and null are not of the same type.**
- adding elements of arrays
- swapping indices that don't exist
    - **For an array of 10 elements, you can swap 10 and 11 or 11 and 12, it doesn't check to make sure that the indices are actually in bound**

Comments - PROBLEMS!

- **Before: I don't know why I kept getting errors with the comment file. I've attached it for you guys to look at. I got an unexpected character error**
- Now, I get a parsing error

Queues

- dequeueing more times that allowed
- enqueueing a string into an integer queue and vice versa

Stacks

- popping more times that allowed
- pushing a string into an integer queue and vice versa

Strings

- assigning integer to a string

APPENDIX B

chad.g

```
/*****************************************************************************
** chad.g
** Lexer and Parser for CHAD
** @author Haronil Estevez
** @author Adam M. Rosenzweig
*****************************************************************************
class CHADLexer extends Lexer;

options
{
  k = 2;
  charVocabulary = '\3'..'\377';
  testLiterals = false;
  exportVocab = chadAntlr;


}


{
  int numLine = 1;
  int nr_error = 0;
  String error = "";

  public void reportError( String s )
  {
    error = s;
    //super.reportError( s );
    nr_error++;
  }

  public void reportError( RecognitionException e )
  {
    error = e.getMessage();
    //super.reportError( e );
    nr_error++;
  }
}



WS: (' ' | '\t' | '\n' { numLine++;  newline(); } | '\r')+ { $setType(Token.SKIP);

protected
ALPHA : 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT : '0'..'9';

INTEGER
    : (DIGIT)+;
```

```
LPAREN : '(';
RPAREN : ')';
MULT : '*';
PLUS : '+';
MINUS : '-';
DIV : '/';
MOD : '%';
SEMI : (';')+;
LBRACE : '{';
RBRACE : '}';
LBRK : '[';
RBRK : ']';
ASGN : '=';
COMMA : ',';
PLUSEQ : "+=";
MINUSEQ : "-=";
MULTEQ : "*=";
RDVEQ : "/=";
MODEQ : "%=";
GE : ">=";
LE : "<=";
GT : '>';
LT : '<';
EQ : "==";
NEQ : "!=";
DOT : ".";


COMMENT : ( "/*" (
  options {greedy=false;} :
  ('\n' | '\r')
  | ~( '\n' | '\r' )
     )* "*/"
| "//" (~( '\n' | '\r' ))* ('\n' | '\r')
)     { $setType(Token.SKIP); }
    { }
;

ID options { testLiterals = true; }
: ALPHA (ALPHA|DIGIT)*
;



STRING : '"'!
   (~('"' | '\n'))*
  '"'!
;



class CHADParser extends Parser;


options{

  k = 2;
  buildAST = true;
  exportVocab = chadAntlr;
  }

tokens { // fill in later
PROG;
STATEMENT;
EXPRESSION;
UMINUS;
FUNC_CALL;
EXPR_LIST;
FUNC_DECL;
SQA_DECL;
STRING_DECL;
RETURN;
DECLARATION;
ASSIGNMENT;
ARRAY_ENTRY;
FUNC_ARGS;
ARRAY_LIST;
INCREMENT;
DECREMENT;
IFBLOCK;
IF_STMT;
ELSEIF_STMT;
ELSE_STMT;
FOR_STMT;
COLLECTION_FUNC_CALL;
ARRAY_RETURN;
ARRAY_ARGUMENT;
}

{
  //int numLine = 1;
  int nr_error = 0;
  String error = "";

  public void reportError( String s )
  {
    error = s;
    //super.reportError( s );
    nr_error++;
  }

  public void reportError( RecognitionException e )
  {
    error = e.getMessage();
    //super.reportError( e );
    nr_error++;
  }
}


program : ((declaration)+ (statement)+ (func_decl)*) EOF!
  { #program = #([STATEMENT,"PROG"], program);  }
  ;

statement
   : assignment
   | func_call SEMI!
   | for_stmt
   | if_block
   {#statement = #([STATEMENT,"STATEMENT"], statement);}
   ;

declaration: ("int"^ ID (ASGN^ expression)? SEMI!) {  }
       | ("string"^ ID (ASGN^ expression)? SEMI!) {  }
       | ("array"^ LBRK! ("int" | "string") COMMA! INTEGER RBRK! ID
         (ASGN^ LBRACE! (expression)
         (COMMA! expression)* RBRACE!)? SEMI!)
         { #declaration = #([SQA_DECL,"ARRAY_DECL"],declaration);}

       | ("stack"^ LBRK! ("int" | "string") RBRK! ID SEMI!) {  #declaration = #([SQA
       | ("queue"^ LBRK! ("int" | "string") RBRK! ID SEMI!) { #declaration = #([SQA_

       ;

expression
   : logic_term ("OR"^ logic_term)*;

logic_term
   : logic_factor ("AND"^ logic_factor)*;

logic_factor
   : ("NOT"^)? relat_expr;

relat_expr
   : arith_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arith_expr)?;

arith_expr
   : arith_term ( (PLUS^ | MINUS^) arith_term)*;

arith_term
   : arith_factor ( (MULT^ | MOD^ | DIV^) arith_factor)*;

arith_factor
   : MINUS! r_value
         {#arith_factor = #([UMINUS,"UMINUS"],arith_factor); }
         | r_value ;

r_value
      : l_value
      | func_call
      | INTEGER {  }
      | STRING { }
      | LPAREN! expression RPAREN!
      | LBRACE! array_list RBRACE!
      ;

l_value
      : ID^ (LBRK! (expression) RBRK! {#l_value = #([ARRAY_ENTRY,"ARRAY_ENTRY"],l_va

func_decl
     : "function"^ func_args(declaration)* (statement)* return_stmt "endfunction"
     {#func_decl = #([FUNC_DECL, "FUNC_DECL"], func_decl); }
     ;


func_args
```

```
  : ("string" | array_return | "int" | "void") ID LPAREN! (("int" | "string" | array
      (COMMA! ("int" | "string" | array_argument) ID)* RPAREN! {#func_args = #([FUNC

array_return
  : "array"! LBRK! ("int" | "string") RBRK!
  {
    #array_return = #([ARRAY_RETURN,"ARRAY_RETURN"],array_return);
  };

array_argument
  : "array"! LBRK! ("int" | "string") COMMA! INTEGER RBRK!
  {
    #array_argument = #([ARRAY_ARGUMENT,"ARRAY_ARGUMENT"],array_argument);
  };


return_stmt:  "return"^ (e2:expression)? SEMI! { #return_stmt = #([RETURN,"RETURN"],

func_call
      : ID (dt:DOT ID)? LPAREN! expr_list RPAREN!
          {
            if(null == dt) // array, queue, and stack function calls
              #func_call = #([FUNC_CALL,"FUNC_CALL"], func_call);
    else // user-defined function calls
      #func_call = #([COLLECTION_FUNC_CALL,"COLLECTION_FUNC_CALL"], func_call);
  };

expr_list
      : expression ( COMMA! expression )*
          { #expr_list = #([EXPR_LIST,"EXPR_LIST"],expr_list); }
          | /* empty node */
            {}
            ;

array_list
      : (expression) (COMMA! expression)*
  { #array_list = #([ARRAY_LIST,"ARRAY_LIST"],array_list); }
  ;

assignment
      : l_value (ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^
                         | RDVEQ^ | MODEQ^) expression SEMI!

      | increment SEMI!

      | decrement SEMI!
        {}
       ;

for_stmt: ("for"!) LPAREN! f1:assignment f2:expression SEMI! f3:assignment RPAREN! (
"endfor"! {
  #for_stmt= #([FOR_STMT, "FOR_STMT"],for_stmt);
  };

if_block: if_stmt (else_if_stmt)* (else_stmt)? { #if_block = #([IFBLOCK, "IFBLOCK"],

if_stmt:  "if"^ LPAREN! e1:expression RPAREN!
          (statement)*
          "endif"!
          {
            #if_stmt = #([IF_STMT, "IF_STMT"],e1);
          }
          ;

else_if_stmt: "elseif"^ LPAREN! e2:expression RPAREN! (statement)* "endelseif"!
 { #else_if_stmt = #([ELSEIF_STMT,"ELSEIF_STMT"],e2);};

else_stmt: "else"! (statement)* "endelse"! {#else_stmt = #([ELSE_STMT,"ELSE_STMT"],e

increment: ID PLUS! PLUS! { #increment = #([INCREMENT, "INCREMENT"], increment); };

decrement: ID MINUS! MINUS! { #decrement = #([DECREMENT, "DECREMENT"], decrement); }
```

CHADEnvironment.java

```
/*********************************************************************************
 ** CHADEnvironment.java
 ** Environment class used for scoping
 ** @author Adam M. Rosenzweig
 *********************************************************************************
import DataStructures.*;
import java.lang.*;
import java.util.*;


public class CHADEnvironment
{
    private Stack symbolTables;
    private CHADSymbolTable currentScope; // this is the top of the stack
    private CHADSymbolTable newScope; // this is a newly created scope that is not c

    public CHADEnvironment()
    {
        symbolTables = new Stack();
        currentScope = new CHADSymbolTable();
        newScope = null;
    }

    public void createNewScope()
    {
        newScope = new CHADSymbolTable();

        Enumeration e = currentScope.getEntries();

        while (e.hasMoreElements())
        {
            CHADType t = (CHADType) e.nextElement();
            addToNew(t);
        }
    }

    public void enterScope()
    {
        currentScope = new CHADSymbolTable();

        symbolTables.push(newScope);
        newScope = null;
    }

    public void exitScope()
    {
        currentScope = (CHADSymbolTable) symbolTables.pop();
    }

    public void addToCurrent(CHADType t)
    {
        currentScope.insert(t);
    }

    public void removeFromCurrent(String t)
    {
        currentScope.remove(t);
    }

    public void addToNew(CHADType t)
    {
        newScope.insert(t);
    }

    public CHADType find(String name)
    {
        return currentScope.find(name);
    }

    public boolean isNewScopeEmpty()
    {
        if (newScope == null)
        {
            return true;
        }

        return false;
    }

    public int numOfEntries()
    {
        return currentScope.numOfEntries();
    }

    public void print()
    {
        Enumeration e = currentScope.getEntries();

        while (e.hasMoreElements())
        {
            CHADType t = (CHADType) e.nextElement();
            System.out.println("ENVIRONMENT: " + t.getName());
        }
    }
}
```

CHADException.java

```
/*********************************************************************************
** CHADException.java
** General exception class
** @author Haronil Estevez
*********************************************************************************
public class CHADException extends Exception
{
    CHADException (String message)
    {
        super("ERROR: " + message);
    }
}
```

CHADGraphics.java

```
/*****************************************************************************
** CHADGraphics.java
** Graphics panel used by CHADGUI
** @author Adam M. Rosenzweig
*****************************************************************************
import DataStructures.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;


public class CHADGraphics extends JPanel {
    private static final int CHAR_WIDTH = 10;
    public CHADType array;
    public CHADType singleton;
    private int boxWidth;
    private int primaryFocus;
    private int secondaryFocus;

    public CHADGraphics() {
        super();
        array = null;
        singleton = null;
        primaryFocus = -1;
        secondaryFocus = -1;
    }

    public void paintComponent(Graphics page) {
        page.clearRect(0, 0, 500, 900);

        int x;
        int y;
        page.drawRect(5, 5, 350, 420);

        if (array != null) {
            // determine size needed
            if (array.getStorageType() == CHADType.CHADTYPE_INT) {
                boxWidth = (4 * CHAR_WIDTH) + 10;
            } else {
                boxWidth = (16 * CHAR_WIDTH) + 10;
            }

            switch (array.getType()) {
            case CHADType.CHADTYPE_ARRAY:

                for (x = 0; x < array.getLength(); x++) {
                    page.setColor(new Color(0.0f, 0.0f, 1.0f));

                    if ((x == primaryFocus) || (x == array.primaryFocus)) {
                        page.setColor(new Color(1.0f, 0.0f, 0.0f));
                    } else if ((x == secondaryFocus) ||
                            (x == array.secondaryFocus)) {
                        page.setColor(new Color(0.0f, 1.0f, 0.0f));
                    }

                    page.fillRect(50, 50 + (20 * x), boxWidth, 20);
                    page.setColor(new Color(0.0f, 0.0f, 0.0f));
                    page.drawRect(50, 50 + (20 * x), boxWidth, 20);
                    page.drawString(Integer.toString(x), 30, 65 + (20 * x));

                    if (array.getStorageType() == CHADType.CHADTYPE_INT) {
                        page.drawString(Integer.toString(array.getIntData(x)),
                            55, 65 + (20 * x));
                    } else {
                        page.drawString(array.getStringData(x), 55,
                            65 + (20 * x));
                    }
                }

                break;

            case CHADType.CHADTYPE_QUEUE:
                x = array.queueData.front;
                y = 0;

                while (y < array.queueData.currentSize) {
                    // draw
                    page.setColor(new Color(0.0f, 0.0f, 1.0f));
                    page.fillRect(50,
                        370 - (20 * (array.queueData.currentSize - y)),
                        boxWidth, 20);
                    page.setColor(new Color(0.0f, 0.0f, 0.0f));
                    page.drawRect(50,
                        370 - (20 * (array.queueData.currentSize - y)),
                        boxWidth, 20);

                    if (x == array.queueData.front) {
                        page.drawString("Head", 10,
                            385 - (20 * (array.queueData.currentSize - y)));
                    }

                    if (array.getStorageType() == CHADType.CHADTYPE_INT) {
                        page.drawString(Integer.toString(((CHADType) array.queueData
                                    0)), 55,
                            385 - (20 * (array.queueData.currentSize - y)));
                    } else {
                        page.drawString(((CHADType) array.queueData.theArray[x]).get
                                0), 55,
                            385 - (20 * (array.queueData.currentSize - y)));
                    }

                    x = array.queueData.increment(x);
                    y += 1;
                }

                if (array.queueData.currentSize > 1) {
                    y -= 1;
                }

                if (!(array.queueData.isEmpty())) {
```

```
                page.drawString("Tail", 10,
                    385 - (20 * (array.queueData.currentSize - y)));
            }

            break;

        case CHADType.CHADTYPE_STACK:

            if (!(array.stackData.isEmpty())) {
                for (x = 0; x < array.stackData.topOfStack; x++) {
                    page.setColor(new Color(0.0f, 0.0f, 0.75f));
                    page.fillRect(50, 370 - (20 * x), boxWidth, 20);
                    page.setColor(new Color(0.0f, 0.0f, 0.0f));
                    page.drawRect(50, 370 - (20 * x), boxWidth, 20);

                    if (array.getStorageType() == CHADType.CHADTYPE_INT) {
                        page.drawString(Integer.toString(((CHADType) array.stack
                                0)), 55, 385 - (20 * x));
                    } else {
                        page.drawString(((CHADType) array.stackData.theArray[x])
                            0), 55, 385 - (20 * x));
                    }
                }

                page.setColor(new Color(0.0f, 0.0f, 1.0f));
                page.fillRect(50, 370 - (20 * x), boxWidth, 20);
                page.setColor(new Color(0.0f, 0.0f, 0.0f));
                page.drawRect(50, 370 - (20 * x), boxWidth, 20);
                page.drawString("Top", 15, 385 - (20 * x));

                if (array.getStorageType() == CHADType.CHADTYPE_INT) {
                    page.drawString(Integer.toString(((CHADType) array.stackData
                            0)), 55, 385 - (20 * x));
                } else {
                    page.drawString((((CHADType) array.stackData.theArray[x]).ge
                        0)), 55, 385 - (20 * x));
                }
            }

            break;

        default:
            break;
        }
    }

    if (singleton != null) {
        if (singleton.getType() == CHADType.CHADTYPE_INT) {
            page.drawString(singleton.getName() + ":  " +
                singleton.getIntData(0), 150, 20);
        } else if (singleton.getType() == CHADType.CHADTYPE_STRING) {
            page.drawString(singleton.getName() + ":  " +
                singleton.getStringData(0), 150, 20);
        }
    }
}

/* Used to set the array that will be displayed */
public void showArray(CHADType theArray) {
        array = theArray;

        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            ;
        }
    }

    public void hideArray() {
        array = null;
    }

    /* Used to set the single variable that will be displayed */
    public void showSingle(CHADType theSingle) {
        singleton = theSingle;
    }

    public void hideSingle() {
        singleton = null;
    }

    public void setPrimary(int p) {
        primaryFocus = p;
    }

    public void setSecondary(int s) {
        secondaryFocus = s;
    }
}
```

CHADGUI.java

```
/************************************************************************************
** CHADGUI.java
** Graphical User Interface class used to load and run CHAD source files
** @author Adam M. Rosenzweig
************************************************************************************
import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;
import antlr.collections.ASTEnumeration;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;


public class CHADGUI extends JFrame implements ActionListener {
    // contains everything
    private Container content;
    // top of the display, contains the buttons
    private JPanel topPanel;
    // left panel, where array/stack/queue
    // is shown and manipulated (change to CHADGraphics)
    public CHADGraphics viewPanel;
    // panel used to hold the left & right panels
    private JPanel holderPanel;
```

```
// button used to load (& compile) a file
private JButton loadButton;
// button used to run the compiled code
private JButton runButton;
// button used to step through compiled code GUI update by GUI update
private JButton stepButton;
// button used to exit the program
private JButton quitButton;
private JLabel spacer;
private JLabel spacer2;
private JScrollPane scroller;
// text field used to open the file to run
private JTextField filenameField;
// the text area in which error output
// or user-specified strings will be displayed
private JTextArea mainTextArea;
private FileInputStream inFile;
CHADLexer lexer = null;
CHADParser parser = null;
int numSteps = 0;

// used to interpret tree step by step
CHADWalker StepWalker = null;
CommonAST st = null;
boolean doneExecuting = false;
boolean step = false;
boolean error = false;
CHADEnvironment env = null; // used to store function names

/* Default constructor.  Simply assembles the GUI */
public CHADGUI() {
    super("CHAD Interpreter");
    setSize(640, 480);
    setResizable(false);

    GridBagConstraints constraints;

    content = getContentPane();
    content.setLayout(new BorderLayout());

    topPanel = new JPanel();
    topPanel.setLayout(new GridBagLayout());
    viewPanel = new CHADGraphics();
    holderPanel = new JPanel();
    holderPanel.setLayout(new GridBagLayout());

    loadButton = new JButton("Load");
    loadButton.addActionListener(this);
    runButton = new JButton("Run");
    runButton.addActionListener(this);
    stepButton = new JButton("Step");
    stepButton.addActionListener(this);
    quitButton = new JButton("Quit");
    quitButton.addActionListener(this);

    filenameField = new JTextField(10);
    filenameField.addActionListener(this);

    spacer = new JLabel("      ");
```

```
spacer2 = new JLabel("      ");

mainTextArea = new JTextArea(
        "CHAD Interpreter initialized.\nPlease type a file into the field ir
mainTextArea.setEditable(false);
mainTextArea.setLineWrap(true);
mainTextArea.setWrapStyleWord(true);

scroller = new JScrollPane(mainTextArea,
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

//scroller.setSize (200, 500);
constraints = new GridBagConstraints();
constraints.gridx = 0;
constraints.gridy = 0;
constraints.gridwidth = 6;
constraints.gridheight = 1;
constraints.weightx = 10;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.ipadx = 8;
topPanel.add(filenameField, constraints);

constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
constraints.gridwidth = 4;
constraints.gridheight = 1;
topPanel.add(loadButton, constraints);

constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
constraints.gridwidth = 3;
constraints.gridheight = 1;
topPanel.add(spacer, constraints);

constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
constraints.gridwidth = 3;
constraints.gridheight = 1;
topPanel.add(runButton, constraints);

constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
constraints.gridwidth = 4;
constraints.gridheight = 1;
topPanel.add(stepButton, constraints);

constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
constraints.gridy = 0;
constraints.gridwidth = 3;
constraints.gridheight = 1;
topPanel.add(spacer2, constraints);

constraints = new GridBagConstraints();
```

```
        constraints.gridx = GridBagConstraints.RELATIVE;
        constraints.gridy = 0;
        constraints.gridwidth = 4;
        constraints.gridheight = 1;
        constraints.ipadx = 8;
        topPanel.add(quitButton, constraints);

        constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.gridwidth = 30;
        constraints.gridheight = 30;
        constraints.weightx = 2;
        constraints.fill = GridBagConstraints.BOTH;
        holderPanel.add(viewPanel, constraints);

        constraints = new GridBagConstraints();
        constraints.gridx = GridBagConstraints.RELATIVE;
        constraints.gridy = 0;
        constraints.gridwidth = 10;
        constraints.gridheight = 30;
        constraints.weightx = 1;
        constraints.weighty = 1;
        constraints.fill = GridBagConstraints.BOTH;
        constraints.anchor = GridBagConstraints.EAST;
        holderPanel.add(scroller, constraints);

        content.add(topPanel, BorderLayout.NORTH);
        content.add(holderPanel, BorderLayout.CENTER);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == quitButton) {
            System.exit(0);
        } else if ((event.getSource() == loadButton) ||
                (event.getSource() == filenameField)) {
            viewPanel.hideArray();
            viewPanel.hideSingle();
            viewPanel.setPrimary(-1);
            viewPanel.setSecondary(-1);
            update(getGraphics());

            updateText("Attempting to load file\n", true);
            CHADType.static_checking = true;
            numSteps = 0;

            try {
                inFile = new FileInputStream(filenameField.getText());
                updateText("Loaded.  Checking that this is a valid CHAD program...\r
                    true);
                lexer = new CHADLexer(inFile);
                parser = new CHADParser(lexer);

                // Parse the input file
                parser.program();

                if ((lexer.nr_error > 0) || (parser.nr_error > 0)) {
                    error = true;

                    if (lexer.nr_error > 0) {
                        throw new CHADException(lexer.error);
                    } else {
                        throw new CHADException(parser.error);
                    }
                }

                // retrieve AST from parser
                CommonAST t = (CommonAST) parser.getAST();
                //System.out.println("-------------------------------");
               //System.out.println(t.toStringList() + "\n");
               // System.out.println("-------------------------------");

                // check static semantics
                StaticCHADWalker walker = new StaticCHADWalker();

                walker.root = t;

                AST ftree = t.getFirstChild();

                while (ftree != null) {
                    walker.functions(ftree);

                    ftree = ftree.getNextSibling();
                }

        // store function names
        env = walker.e2;

                // set root variable to be used by userDefined()
                // in walker
                walker.root = t;
                walker.checkingFunctions = false;

                // get the first child
                AST tree = t.getFirstChild();
                int steps = 0;

                // Traverse the tree created by the parser
                while (tree != null) {
                    walker.expr(tree);
                    tree = tree.getNextSibling();
                }

        error = false;

                updateText("CHAD source is valid. \n", true);
            } catch (IOException exc) {
                updateText(exc.getMessage() + "\n", true);
                error = true;
            } catch (RecognitionException e) {
                error = true;
            } catch (TokenStreamException e) {
              error = true;
                if (lexer != null) {
                    updateText("ERROR: LINE " + lexer.numLine + ": " +
                        e.getMessage() + "\n", true);
                } else {
                    updateText(e.getMessage() + "\n", true);
                }

                error = true;
            } catch (DataStructures.Overflow e) {
                error = true;
                ;
            } catch (CHADException e) {
                error = true;
                updateText(e.getMessage() + "\n", true);
            }

        } else if (event.getSource() == runButton) {
            step = false;

            // verify that a source file has been loaded
            if ((lexer == null) || (parser == null) || error) {
                updateText("ERROR: No valid source file has been loaded!!!" +
                    "\n", true);
            }
            else {
                CHADType.static_checking = false;

                updateText("Running " + filenameField.getText() + "....\n", true);

                try {
                    // retrieve AST from parser
                    CommonAST t = (CommonAST) parser.getAST();

                    CHADWalker walker = new CHADWalker();

                    // set symbol table holding function names
                    walker.e2 = env;

                    // set root variable to be used by userDefined()
                    // in walker
                    walker.root = t;
                    walker.checkingFunctions = false;
                    walker.changes = new Vector();

                    // get the first child
                    AST tree = t.getFirstChild();
                    int steps = 0;

                    // walk the tree
                    walker.expr(t);

                   //System.out.println("NUM CHANGES: " + walker.changes.size());

                    // iterate through vector and update corresponding
                    // visible CHADTypes
                    for (int i = 0; i < walker.changes.size(); i++) {
                        CHADType change = (CHADType) walker.changes.elementAt(i);

                        if (change.getName().equals("print")) {
                            updateText(change.getStringData(0) + "\n", true);

                            update(getGraphics());
                }
                else if (!change.getVisible()) {
                    if (viewPanel.array != null) {
                        if (viewPanel.array.getName().equals(change.getName(
                            viewPanel.hideArray();
                            update(getGraphics());
                        }
                    }

                    if (viewPanel.singleton != null) {
                        if (viewPanel.singleton.getName().equals(change.getN
                            viewPanel.hideSingle();
                            update(getGraphics());
                        }
                    }
                }
                else {
                    update(change, walker);
                }

                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    ;
                }
            }

            updateText("Program has finished executing\n\n", true);
            doneExecuting = true;
        }
         catch (RecognitionException e) {
            updateText(e.getMessage() + "\n", true);
        }
         catch (CHADException e) {
            updateText(e.getMessage() + "\n", true);
        }
    }
} else if (event.getSource() == stepButton) {
    step = true;

    if (doneExecuting) {
        numSteps = 0;
        doneExecuting = false;
    }

    // verify that a source file has been loaded
    if ((lexer == null) || (parser == null) || error) {
        updateText("ERROR: No valid source file has been loaded!!!" +
            "\n", true);
    }
    else {
        numSteps++;

        CHADType.static_checking = false;

        try {
            if (numSteps == 1) {
                StepWalker = new CHADWalker();
                st = (CommonAST) parser.getAST();
```

```
                    // set symbol table holding function names
                    StepWalker.e2 = env;

                    // set root variable to be used by userDefined()
                    // in walker
                    StepWalker.root = st;
                    StepWalker.checkingFunctions = false;
                    StepWalker.changes = new Vector();

                    // walk the tree
                    StepWalker.expr(st);
                }

                // get next member of change vector and update corresponding
                // visible CHADTypes
                if (StepWalker.changes.size() > 0) {
                    CHADType change = (CHADType) StepWalker.changes.elementAt(0)

                    if (change.getName().equals("print")) {
                        updateText(change.getStringData(0) + "\n", true);
                    }
                    else if (!change.getVisible()) {
                        if (viewPanel.array != null) {
                            if (viewPanel.array.getName().equals(change.getName(
                                viewPanel.hideArray();
                                update(getGraphics());
                            }
                        }

                        if (viewPanel.singleton != null) {
                            if (viewPanel.singleton.getName().equals(change.getN
                                viewPanel.hideSingle();
                                update(getGraphics());
                            }
                        }
                    }
                    else {
                        update(change, StepWalker);
                    }

                    StepWalker.changes.removeElementAt(0);

                    if (StepWalker.changes.size() == 0) {
                        updateText("Program has finished executing\n\n",
                            true);
                        numSteps = 0;
                    }
                }
            }
             catch (RecognitionException e) {
            }
             catch (CHADException e) {
                updateText(e.getMessage() + "\n", true);
            }
        }
    }
}
```

```
    public void updateText(String theText, boolean append) {
        if (!append) {
            mainTextArea.setText("");
        }

        mainTextArea.append(theText);
    }

    public static void main(String[] args) {
        new CHADGUI();
    }

    public void update(CHADType c, CHADWalker walker) {
        if ((c.getType() != CHADType.CHADTYPE_INT) &&
                (c.getType() != CHADType.CHADTYPE_STRING)) {
            viewPanel.showArray(c);
        } else {
            viewPanel.showSingle(c);
        }

        //viewPanel.grabFocus();
        update(getGraphics());
    }
}
```

CHADSymbolTable.java

```
/***********************************************************************************
 ** CHADSymbolTable.java
 ** Symbol table class used to store CHADTypes
 ** @author Adam M. Rosenzweig
 **********************************************************************************
import DataStructures.*;
import java.lang.*;
import java.util.*;


public class CHADSymbolTable
{
    private Hashtable table;

    public CHADSymbolTable()
    {
        table = new Hashtable();
    }

    public void insert(CHADType t)
    {
        table.put(t.getName(), t);
    }

    public void remove(String t)
    {
        table.remove(t);
    }

    public CHADType find(String name)
    {
        return (CHADType) table.get(name);
    }
```

```java
    public void edit(String name, CHADType val)
    {
        CHADType entry = find(name);

        if (entry == null)
        {
            return;
        }

        entry = val;
    }

    public int numOfEntries()
    {
        return table.size();
    }

    public Enumeration getEntries()
    {
        return table.elements();
    }

    public boolean isEmpty()
    {
        return table.isEmpty();
    }
}
```

CHADType.java

```java
/***********************************************************************************
** CHADType.java
** Tree Walker for CHAD - Used as the interpreter
** @author Haronil Estevez
** @author Adam M. Rosenzweig
** IMPORTANT NOTE:  getIntData and getStringData functions will dequeue from queues.
************************************************************************************
import DataStructures.*;
import antlr.collections.AST;
import java.lang.*;


public class CHADType implements Hashable, java.io.Serializable
{
    public static final int CHADTYPE_VOID = 0;
    public static final int CHADTYPE_INT = 1;
    public static final int CHADTYPE_STRING = 2;
    public static final int CHADTYPE_ARRAY = 3;
    public static final int CHADTYPE_QUEUE = 4;
    public static final int CHADTYPE_STACK = 5;
    public static final int CHADTYPE_FUNCTION = 6;
    public static boolean static_checking = true;
    private String name;
    private int type;
    private int storageType;
    public CHADType[] arrayData; // for array
    public QueueAr queueData; // for queue
    public StackAr stackData; // for stack
    private int intValue;
    private String stringValue;
    private boolean visible;
    public int primaryFocus = -1;
    public int secondaryFocus = -1;
    public int return_storage_type = 0;
    public int return_array_length = 0;

    /* Default Constructor */
    public CHADType()
    {
        name = null;
        type = CHADTYPE_VOID;
        storageType = CHADTYPE_VOID;
        arrayData = null;
        queueData = null;
        stackData = null;
        visible = false;
    }

    /* Pure name constructor */
    public CHADType(String n)
    {
        name = new String(n);
        type = CHADTYPE_VOID;
        storageType = CHADTYPE_VOID;
        arrayData = null;
        queueData = null;
        stackData = null;
        visible = false;
    }

    public CHADType(String n, int t)
    {
        name = new String(n);
        type = t;
        visible = false;
    }

    /* Constructor for ints and strings with a name */
    public CHADType(String n, int t, int vi, String vs)
    {
        name = new String(n);
        type = t;

        if (type == CHADTYPE_INT)
        {
            intValue = vi;
            stringValue = null;
        }
        else if (type == CHADTYPE_STRING)
        {
            intValue = 0;
            stringValue = new String(vs);
        }

        arrayData = null;
        queueData = null;
        stackData = null;
        visible = false;
    }

    /* Constructor for ints and strings without a name */
    public CHADType(int t, int vi, String vs)
    {
        name = null;
        type = t;
        storageType = t;

        if (type == CHADTYPE_INT)
        {
            intValue = vi;
            stringValue = null;
        }
        else if (type == CHADTYPE_STRING)
        {
            intValue = 0;
            stringValue = new String(vs);
        }

        arrayData = null;
        queueData = null;
        stackData = null;
        visible = false;
    }

    /* Constructor for arrays, stacks, queues & functions */
    public CHADType(String n, int t, int s, int size)
    {
        name = new String(n);
        type = t;
        storageType = s;

        if ((type == CHADTYPE_ARRAY) || (type == CHADTYPE_FUNCTION))
        {
            arrayData = new CHADType[size];
            queueData = null;
            stackData = null;
        }
        else if (type == CHADTYPE_QUEUE)
        {
            queueData = new QueueAr(16);
            arrayData = null;
            stackData = null;
        }
        else
        {
            stackData = new StackAr(16);
            arrayData = null;
            queueData = null;
        }

        visible = false;
    }

    /* Used to insert a value into a stack/queue/array */
    /* Also used to insert argument type into function */
    public void insert(int index, int idata, String sdata)
        throws CHADException
```

```
{
    try
    {
        if (type == CHADTYPE_ARRAY)
        {
            arrayData[index] = new CHADType(storageType, idata, sdata);
        }
        else if (type == CHADTYPE_QUEUE)
        {
            queueData.enqueue(new CHADType(storageType, idata, sdata));
        }
        else if (type == CHADTYPE_STACK)
        {
            stackData.push(new CHADType(storageType, idata, sdata));
        }
        else
        {
            arrayData[index] = new CHADType(idata, idata, sdata);
        }
    }

    catch (Overflow o)
    {
        String errorMessage = name + " has overflowed";
        throw new CHADException(errorMessage);
    }
}

public int getType()
{
    return type;
}

public int getStorageType()
{
    return storageType;
}

public CHADType[] getArrayData()
{
    return arrayData;
}

public CHADType getArrayEntry(int index)
{
    //CHADType a = arrayData[index];
    return arrayData[index];
}

public void setArrayData(CHADType[] ad)
{
    for (int i = 0; i < ad.length; i++)
    {
        CHADType c = ad[i];

        if (c != null)
        {
            arrayData[i] = new CHADType(getName(), c.getType(),
                    c.getIntData(0), c.getStringData(0));
        }
    }
}

/* Used to get integer data out of the type.  Index only relevant for arrays. */
public int getIntData(int index)
{
    if ((type == CHADTYPE_ARRAY) && (storageType == CHADTYPE_INT))
    {
        return arrayData[index].getIntData(index);
    }
    else if ((type == CHADTYPE_QUEUE) && (storageType == CHADTYPE_INT))
    {
        return ((CHADType) (queueData.dequeue())).getIntData(index);
    }
    else if ((type == CHADTYPE_STACK) && (storageType == CHADTYPE_INT))
    {
        return ((CHADType) (stackData.top())).getIntData(index);
    }
    else if (type == CHADTYPE_INT)
    {
        return intValue;
    }
    else if (type == CHADTYPE_FUNCTION)
    {
        return arrayData[index].getIntData(index);
    }

    return 0;
}

public String getStringData(int index)
{
    if (((type == CHADTYPE_ARRAY) || (type == CHADTYPE_FUNCTION)) &&
            (storageType == CHADTYPE_STRING))
    {
        return arrayData[index].getStringData(index);
    }
    else if ((type == CHADTYPE_QUEUE) && (storageType == CHADTYPE_STRING))
    {
        return ((CHADType) (queueData.dequeue())).getStringData(index);
    }
    else if ((type == CHADTYPE_STACK) && (storageType == CHADTYPE_STRING))
    {
        return ((CHADType) (stackData.top())).getStringData(index);
    }
    else if (type == CHADTYPE_STRING)
    {
        return new String(stringValue);
    }

    return null;
}

public int popInt()
{
    CHADType c = null;

    c = (CHADType) stackData.topAndPop();
    return c.getIntData(0);

    //return ((CHADType)(stackData.pop ())).getIntData (0);
}

public int topInt()
{
    CHADType c = null;

    c = (CHADType) stackData.top();

    return c.getIntData(0);

    //return ((CHADType)(stackData.pop ())).getIntData (0);
}

public String popString()
{
    CHADType c = null;

    c = (CHADType) stackData.topAndPop();

    return c.getStringData(0);

    //return ((CHADType) (stackData.topAndPop ())).getStringData (0);
}

public String topString()
{
    CHADType c = null;

    c = (CHADType) stackData.top();

    return c.getStringData(0);

    //return ((CHADType) (stackData.topAndPop ())).getStringData (0);
}

public void setIntValue(int v)
{
    intValue = v;
}

// for arrays only
public void setIntValue(int index, int v)
{
    arrayData[index].setIntValue(v);
}

public void setStringValue(String v)
{
    stringValue = new String(v);
}

// for arrays only
public void setStringValue(int index, String v)
{
    arrayData[index].setStringValue(v);
}

int getLength()
{
    return arrayData.length;
}

public String getName()
{
    return name;
}

public void setName(String n)
{
    name = n;
}

public void show()
{
    visible = true;
}

public void hide()
{
    visible = false;
}

public boolean getVisible()
{
    return visible;
}

public int hash(int tablesize)
{
    int hashVal = 0;
    int i;

    for (i = 0; i < name.length(); i++)
    {
        hashVal = (37 * hashVal) + name.charAt(i);
    }

    hashVal %= tablesize;

    if (hashVal < 0)
    {
        hashVal += tablesize;
    }

    return hashVal;
}

// prints data for int and string types,
// name for array, queue, and stack types
public String toString()
{
    if ((getName() != null) && getName().equals(""))
    {
        return getName();
    }

    else if (type == CHADTYPE_INT)
    {
        return Integer.toString(getIntData(0));
    }
    else if (type == CHADTYPE_STRING)
    {
        return getStringData(0);
    }
    else if ((type == CHADTYPE_ARRAY) || (type == CHADTYPE_QUEUE) ||
            (type == CHADTYPE_STACK) || (type == CHADTYPE_FUNCTION))
    {
        return getName();
    }

    return "";
}

// return type name
public String typeName()
{
    String t = null;

    switch (type)
    {
    case CHADTYPE_VOID:
        t = "void";

        break;

    case CHADTYPE_INT:
        t = "integer";

        break;

    case CHADTYPE_STRING:
        t = "string";

        break;

    case CHADTYPE_ARRAY:
        t = "array";

        break;

    case CHADTYPE_QUEUE:
        t = "queue";

        break;

    case CHADTYPE_STACK:
        t = "stack";

        break;

    case CHADTYPE_FUNCTION:
        t = "function";
```

```java
            break;
        }

        return t;
    }

    // return storage type name
    public String storageTypeName()
    {
        String t = null;

        switch (storageType)
        {
        case CHADTYPE_VOID:
            t = "void";

            break;

        case CHADTYPE_INT:
            t = "integer";

            break;

        case CHADTYPE_STRING:
            t = "string";

            break;

        case CHADTYPE_ARRAY:
            t = "array";

            break;

        case CHADTYPE_QUEUE:
            t = "stack";

            break;

        case CHADTYPE_STACK:
            t = "stack";

            break;
        }

        return t;
    }

    //--------------------------------------------------------------------------
    // Built in functions for arrays, stacks, and queues
    //--------------------------------------------------------------------------
    public CHADType pop() throws CHADException
    {
        CHADType c = null;

        if (getType() != CHADTYPE_STACK)
        {
            String errorMessage = "Pop function can only be used by stacks!";
            throw new CHADException(errorMessage);
        }
        if (stackData.isEmpty())
        {
            String errorMessage = getName() + " is an empty stack!!!";
            throw new CHADException(errorMessage);
        }

        if (getStorageType() == CHADTYPE_INT)
        {
            int data = popInt();
            c = new CHADType(CHADTYPE_INT, data, "");
            c.setName(getName());
        }

        if (getStorageType() == CHADTYPE_STRING)
        {
            String data = popString();
            c = new CHADType(CHADTYPE_STRING, 0, data);
            c.setName(getName());
        }

        return c;
    }

    public CHADType top() throws CHADException
    {
        CHADType c = null;

        if (getType() != CHADTYPE_STACK)
        {
            String errorMessage = "Pop function can only be used by stacks!";
            throw new CHADException(errorMessage);
        }

        if (stackData.isEmpty())
        {
            String errorMessage = getName() + " is an empty stack!!!";
            throw new CHADException(errorMessage);
        }

        if (getStorageType() == CHADTYPE_INT)
        {
            int data = topInt();
            c = new CHADType(CHADTYPE_INT, data, "");
        }

        if (getStorageType() == CHADTYPE_STRING)
        {
            String data = topString();
            c = new CHADType(CHADTYPE_STRING, 0, data);
        }

        return c;
    }

    public int push(CHADType a) throws CHADException
    {
        try
        {
            CHADType c = new CHADType(getName(), a.getType(), a.getIntData(0),
                    a.getStringData(0));
            stackData.push(c);
        }
        catch (Overflow o)
        {
            String errorMessage = "stacks can hold a maximum of 16 elements!!!";
            throw new CHADException(errorMessage);
        }

        return 1;
    }

    public int enqueue(CHADType a) throws CHADException
    {
        try
        {
            CHADType c = new CHADType(getName(), a.getType(), a.getIntData(0),
                    a.getStringData(0));
            queueData.enqueue(c);
        }
        catch (Overflow o)
        {
            String errorMessage = "queues can hold a maximum of 16 elements!!!";
            throw new CHADException(errorMessage);
        }

        return 1;
    }

    public CHADType dequeue() throws CHADException
    {
        CHADType c = null;

        if (getType() != CHADTYPE_QUEUE)
        {
            String errorMessage = "dequeue function can only be used by queues!!!";
            throw new CHADException(errorMessage);
        }

        if (queueData.isEmpty())
        {
            String errorMessage = getName() + " is an empty queue!!!";
            throw new CHADException(errorMessage);
        }

        if (getStorageType() == CHADTYPE_INT)
        {
            int data = getIntData(0);
            c = new CHADType(CHADTYPE_INT, data, null);
        }

        if (getStorageType() == CHADTYPE_STRING)
        {
            String data = getStringData(0);
            c = new CHADType(CHADTYPE_STRING, 0, data);
        }

        return c;
    }

    public void swap(CHADType a, CHADType b) throws CHADException
    {
        if (getType() != CHADTYPE_ARRAY)
        {
            String errorMessage = "swap(a,b) can only be called by arrays!!!";
            throw new CHADException(errorMessage);
        }

        if ((a.getType() != CHADTYPE_INT) || (b.getType() != CHADTYPE_INT))
        {
            String errorMessage = "swap(a,b) takes integer types as arguments!";
            throw new CHADException(errorMessage);
        }

        int first_index = a.getIntData(0);
        int second_index = b.getIntData(0);

        if ((first_index >= getLength()) | (second_index >= getLength()))
        {
            String errorMessage = "swap(a,b) index values out of bounds!!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            if (getStorageType() == CHADTYPE_INT)
            {
                int first_value = getIntData(first_index);
                int second_value = getIntData(second_index);

                setIntValue(first_index, second_value);
                setIntValue(second_index, first_value);
            }
            else
            {
                String first_value = getStringData(first_index);
                String second_value = getStringData(second_index);

                setStringValue(first_index, second_value);
                setStringValue(second_index, first_value);
            }
        }
    }

    public void sortAZ(CHADType a, CHADType b) throws CHADException
    {
        if (getType() != CHADTYPE_ARRAY)
        {
            String errorMessage = "sortAZ(a,b) can only be called by arrays!!!";
            throw new CHADException(errorMessage);
        }

        if ((a.getType() != CHADTYPE_INT) || (b.getType() != CHADTYPE_INT))
        {
            String errorMessage = "sortAZ(a,b) takes integer types as arguments!";
            throw new CHADException(errorMessage);
        }
        int first_index = a.getIntData(0);
        int second_index = b.getIntData(0);

        if ((first_index >= getLength()) | (second_index >= getLength()))
        {
            String errorMessage = "sortAZ(a,b) index values out of bounds!!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            if (getStorageType() == CHADTYPE_INT)
            {
                int first_value = getIntData(first_index);
                int second_value = getIntData(second_index);

                if (first_value > second_value)
                {
                    setIntValue(first_index, second_value);
                    setIntValue(second_index, first_value);
                }
            }
            else
            {
                String first_value = getStringData(first_index);
                String second_value = getStringData(second_index);

                if (first_value.compareTo(second_value) > 0)
                {
                    setStringValue(first_index, second_value);
                    setStringValue(second_index, first_value);
                }
            }
        }
    }

    public void sortZA(CHADType a, CHADType b) throws CHADException
    {
        if (getType() != CHADTYPE_ARRAY)
        {
            String errorMessage = "sortZA(a,b) can only be called by arrays!!!";
            throw new CHADException(errorMessage);
        }

        if ((a.getType() != CHADTYPE_INT) || (b.getType() != CHADTYPE_INT))
        {
            String errorMessage = "sortZA(a,b) takes integer types as arguments!";
            throw new CHADException(errorMessage);
        }

        int first_index = a.getIntData(0);
        int second_index = b.getIntData(0);

        if ((first_index >= getLength()) | (second_index >= getLength()))
        {
            String errorMessage = "sortZA(a,b) index values out of bounds!!!";
            throw new CHADException(errorMessage);
        }
        if (!static_checking)
        {
            if (getStorageType() == CHADTYPE_INT)
            {
                int first_value = getIntData(first_index);
                int second_value = getIntData(second_index);

                if (first_value < second_value)
                {
                    setIntValue(first_index, second_value);
                    setIntValue(second_index, first_value);
                }
            }
            else
            {
                String first_value = getStringData(first_index);
                String second_value = getStringData(second_index);

                if (first_value.compareTo(second_value) < 0)
                {
                    setStringValue(first_index, second_value);
                    setStringValue(second_index, first_value);
                }
            }
        }
    }

    //----------------------------------------------------------------------------
    // MIN AND MAX built in functions
    //----------------------------------------------------------------------------
    public static CHADType min(CHADType a, CHADType b)
        throws CHADException
    {
        CHADType c = new CHADType(a.getType(), 0, "");

        if ((a.getType() != CHADTYPE_INT) && (a.getType() != CHADTYPE_STRING) &&
                (b.getType() != CHADTYPE_INT) &&
                (b.getType() != CHADTYPE_STRING))
        {
            String errorMessage = "min(a,b) can only be called on int and string typ
            throw new CHADException(errorMessage);
        }

        if (a.getType() != b.getType())
        {
            String errorMessage = "min(a,b) must have arguments with the same type!!
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            CHADType r = a.isLessThan(b);

            if (r.getIntData(0) == 1)
            {
                c = a;
            }
            else
            {
                c = b;
            }
        }

        return c;
    }

    public static CHADType max(CHADType a, CHADType b)
        throws CHADException
    {
        CHADType c = new CHADType(a.getType(), 0, "");

        if ((a.getType() != CHADTYPE_INT) && (a.getType() != CHADTYPE_STRING) &&
                (b.getType() != CHADTYPE_INT) &&
                (b.getType() != CHADTYPE_STRING))
        {
            String errorMessage = "max(a,b) can only be called on int and string typ
            throw new CHADException(errorMessage);
        }

        if (a.getType() != b.getType())
        {
            String errorMessage = "max(a,b) must have arguments with the same type!!
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            CHADType r = a.isGreaterThan(b);

            if (r.getIntData(0) == 1)
            {
                c = a;
            }
            else
            {
                c = b;
            }
        }

        return c;
    }

    //----------------------------------------------------------------------------
    // UNARY OPERATORS
    //----------------------------------------------------------------------------
    public CHADType uminus() throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if (getType() != CHADTYPE_INT)
        {
            String errorMessage = "UNARY - can only be applied to integer types!!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            c = new CHADType(CHADTYPE_INT, -getIntData(0), "");
        }

        return c;
    }

    //---------------------------------------------------------------------------
    // BINARY OPERATORS
    //---------------------------------------------------------------------------
    public CHADType plus(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(getType(), 0, "");

        if ((getType() == CHADTYPE_VOID) || (getType() == CHADTYPE_VOID) ||
                (getType() == CHADTYPE_QUEUE) ||
                (b.getType() == CHADTYPE_QUEUE) ||
                (getType() == CHADTYPE_STACK) ||
                (b.getType() == CHADTYPE_STACK))
        {
            String errorMessage = "+ operator must be used on integer or string type
            throw new CHADException(errorMessage);
        }

        if (getType() != b.getType())
        {
            String errorMessage = "+ operator requires same types!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            if (getType() == CHADTYPE_INT)
            {
                int sum = getIntData(0) + b.getIntData(0);

                c = new CHADType(CHADTYPE_INT, sum, "");
            }

            if (getType() == CHADTYPE_STRING)
            {
                String sum = getStringData(0) + b.getStringData(0);

                c = new CHADType(CHADTYPE_STRING, 0, sum);
            }
        }

        return c;
    }

    public CHADType minus(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() == CHADTYPE_VOID) || (getType() == CHADTYPE_VOID) ||
                (getType() == CHADTYPE_QUEUE) ||
                (b.getType() == CHADTYPE_QUEUE) ||
                (getType() == CHADTYPE_STACK) ||
                (b.getType() == CHADTYPE_STACK) ||
                (getType() == CHADTYPE_STRING) ||
                (b.getType() == CHADTYPE_STRING))
        {
            String errorMessage = "- operator must be used on integer types!!";
            throw new CHADException(errorMessage);
        }

        if (getType() != b.getType())
        {
            String errorMessage = "- operator requires same types!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            int difference = getIntData(0) - b.getIntData(0);
            c = new CHADType(CHADTYPE_INT, difference, "");
        }

        return c;
    }

    public CHADType mult(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() == CHADTYPE_VOID) || (getType() == CHADTYPE_VOID) ||
                (getType() == CHADTYPE_QUEUE) ||
                (b.getType() == CHADTYPE_QUEUE) ||
                (getType() == CHADTYPE_STACK) ||
                (b.getType() == CHADTYPE_STACK) ||
                (getType() == CHADTYPE_STRING) ||
                (b.getType() == CHADTYPE_STRING))
        {
            String errorMessage = "* operator must be used on integer types!!";
            throw new CHADException(errorMessage);
        }

        if (getType() != b.getType())
        {
            String errorMessage = "* operator requires same types!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            int product = getIntData(0) * b.getIntData(0);

            c = new CHADType(CHADTYPE_INT, product, "");
        }

        return c;
    }

    public CHADType div(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() == CHADTYPE_VOID) || (getType() == CHADTYPE_VOID) ||
                (getType() == CHADTYPE_QUEUE) ||
                (b.getType() == CHADTYPE_QUEUE) ||
                (getType() == CHADTYPE_STACK) ||
                (b.getType() == CHADTYPE_STACK) ||
                (getType() == CHADTYPE_STRING) ||
                (b.getType() == CHADTYPE_STRING))
        {
            String errorMessage = "/ operator must be used on integer types!!";
            throw new CHADException(errorMessage);
        }

        if (getType() != b.getType())
        {
            String errorMessage = "/ operator requires same types!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            int quotient = getIntData(0) / b.getIntData(0);

            c = new CHADType(CHADTYPE_INT, quotient, "");
        }

        return c;
    }

    public CHADType mod(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() == CHADTYPE_VOID) || (getType() == CHADTYPE_VOID) ||
                (getType() == CHADTYPE_QUEUE) ||
                (b.getType() == CHADTYPE_QUEUE) ||
                (getType() == CHADTYPE_STACK) ||
                (b.getType() == CHADTYPE_STACK) ||
                (getType() == CHADTYPE_STRING) ||
                (b.getType() == CHADTYPE_STRING))
        {
            String errorMessage = "% operator must be used on integer types!!";
            throw new CHADException(errorMessage);
        }

        if (getType() != b.getType())
        {
            String errorMessage = "% operator requires same types!!";
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            int modu = getIntData(0) % b.getIntData(0);

            c = new CHADType(CHADTYPE_INT, modu, "");
        }

        return c;
    }

    public void asgn(CHADType b) throws CHADException
```

```
{
    // make sure stacks and queues cannot be assigned
    if ((getType() == CHADTYPE_QUEUE) || (getType() == CHADTYPE_STACK))
    {
        String errorMessage = "stacks and queues cannot be assigned!!!";
        throw new CHADException(errorMessage);
    }

    // handle assignments that are NOT array declarations
    if ((getType() == CHADTYPE_INT) || (getType() == CHADTYPE_STRING))
    {
        // check that types are the same
        if (getType() != b.getType())
        {
            String errorMessage = getName() +
                " cannot be assigned to type " + b.typeName();
            throw new CHADException(errorMessage);
        }

        if (getType() == CHADTYPE_INT)
        {
            if ((b.getIntData(0) < -999) || (b.getIntData(0) > 999))
            {
                // handled by walker
            }

            if (!static_checking)
            {
                setIntValue(b.getIntData(0));

                if ((intValue > 999) || (intValue < -999))
                {
                    intValue = 0;
                }
            }
        }

        if (getType() == CHADTYPE_STRING)
        {
            if (b.getStringData(0).length() > 30)
            {
                // handled by walker
            }

            if (!static_checking)
            {
                setStringValue(b.getStringData(0));

                if (stringValue.length() > 30)
                {
                    stringValue = "";
                }
            }
        }
    }

    // handle array type assignments
    else if ((getType() == CHADTYPE_ARRAY) &&
            (b.getType() == CHADTYPE_ARRAY))
    {
        // handle two declared arrays
        if ((getName() != null) && !(b.getName().equals("")))
        {
            if (getStorageType() != b.getStorageType())
            {
                String errorMessage = "array " + getName() + " holds " +
                    storageTypeName() + " types!!!";
                throw new CHADException(errorMessage);
            }

            // if array size smaller for array on left side of "=", throw error
            if (getLength() < b.getLength())
            {
                String errorMessage = "array " + getName() +
                    " is smaller than array " + b.getName() + "!!!";
                throw new CHADException(errorMessage);

            }

            if (!static_checking)
            {
                setArrayData(b.getArrayData());
            }
        }

        // handle an array list assignment
        if ((getName() != null) && b.getName().equals(""))
        {
            if (getStorageType() != b.getStorageType())
            {
                String errorMessage = "array " + getName() + " holds " +
                    storageTypeName() + " types!!!";
                throw new CHADException(errorMessage);
            }

            // if element list exceeds array boundary, throw error
            if (b.getLength() > getLength())
            {
                String errorMessage = "array " + getName() +
                    " can only hold " + getLength() + " elements!!";
                throw new CHADException(errorMessage);
            }

setArrayData(b.arrayData);
        }
    }

    else if (getStorageType() != b.getType())
    {
        String errorMessage = "array " + getName() + " holds " +
            typeName() + " types!!!";
        throw new CHADException(errorMessage);
    }
}

//----------------------------------------------------------------------------
// INCREMENT AND DECREMENT OPERATIONS
//----------------------------------------------------------------------------
```

```
public void increment() throws CHADException
{
    if (getType() != CHADTYPE_INT)
    {
        String errorMessage = "++ operator can only be applied to integers!!!";
        throw new CHADException(errorMessage);
    }

    if (!static_checking)
    {
        int inc = getIntData(0) + 1;
        setIntValue(inc);
    }
}

public void decrement() throws CHADException
{
    if (getType() != CHADTYPE_INT)
    {
        String errorMessage = "--operator can only be applied to integers!!!";
        throw new CHADException(errorMessage);
    }

    if (!static_checking)
    {
        int dec = getIntData(0) - 1;
        setIntValue(dec);
    }
}

//--------------------------------------------------------------------------
// COMPARISON OPERATORS
//--------------------------------------------------------------------------
public CHADType isEqual(CHADType b) throws CHADException
{
    CHADType c = new CHADType(CHADTYPE_INT, 0, "");

    if ((getType() != CHADTYPE_INT) && (getType() != CHADTYPE_STRING) &&
            (b.getType() != CHADTYPE_INT) &&
            (b.getType() != CHADTYPE_STRING))
    {
        String errorMessage = "== operator can only be used on int and string ty
        throw new CHADException(errorMessage);
    }

    if (!static_checking)
    {
        if (getType() == b.getType())
        {
            if (getType() == CHADTYPE_INT)
            {
                if (getIntData(0) == b.getIntData(0))
                {
                    c.setIntValue(1);
                }
            }
            else if (getType() == CHADTYPE_STRING)
            {
                if (getStringData(0).equals(b.getStringData(0)))
                {
                    c.setIntValue(1);
                }
            }
        }
    }

    return c;
}

public CHADType isGreaterThan(CHADType b) throws CHADException
{
    CHADType c = new CHADType(CHADTYPE_INT, 0, "");

    if ((getType() != CHADTYPE_INT) && (getType() != CHADTYPE_STRING) &&
            (b.getType() != CHADTYPE_INT) &&
            (b.getType() != CHADTYPE_STRING))
    {
        String errorMessage = "> operator can only be used to compare int and st
        throw new CHADException(errorMessage);
    }

    if (!static_checking)
    {
        if (getType() == b.getType())
        {
            if (getType() == CHADTYPE_INT)
            {
                if (getIntData(0) > b.getIntData(0))
                {
                    c.setIntValue(1);
                }
            }
            else if (getType() == CHADTYPE_STRING)
            {
                if (getStringData(0).compareTo(b.getStringData(0)) > 0)
                {
                    c.setIntValue(1);
                }
            }
        }
    }

    return c;
}

public CHADType isGreaterThanOrEqualTo(CHADType b)
    throws CHADException
{
    CHADType c = new CHADType(CHADTYPE_INT, 0, "");

    if ((getType() != CHADTYPE_INT) && (getType() != CHADTYPE_STRING) &&
            (b.getType() != CHADTYPE_INT) &&
            (b.getType() != CHADTYPE_STRING))
    {
        String errorMessage = ">= operator can only be used to compare int and s
        throw new CHADException(errorMessage);
    }
```

```
        if (!static_checking)
        {
            if (getType() == b.getType())
            {
                if (getType() == CHADTYPE_INT)
                {
                    if (getIntData(0) >= b.getIntData(0))
                    {
                        c.setIntValue(1);
                    }
                }
                else if (getType() == CHADTYPE_STRING)
                {
                    if ((getStringData(0).compareTo(b.getStringData(0)) > 0) |
                            (getStringData(0).compareTo(b.getStringData(0)) == 0))
                    {
                        c.setIntValue(1);
                    }
                }
            }
        }

        return c;
    }

    public CHADType isLessThan(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() != CHADTYPE_INT) && (getType() != CHADTYPE_STRING) &&
                (b.getType() != CHADTYPE_INT) &&
                (b.getType() != CHADTYPE_STRING))
        {
            String errorMessage = "< operator can only be used to compare int and st
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            if (getType() == b.getType())
            {
                if (getType() == CHADTYPE_INT)
                {
                    if (getIntData(0) < b.getIntData(0))
                    {
                        c.setIntValue(1);
                    }
                }
                else if (getType() == CHADTYPE_STRING)
                {
                    if (getStringData(0).compareTo(b.getStringData(0)) < 0)
                    {
                        c.setIntValue(1);
                    }
                }
            }
        }

        return c;
    }

    public CHADType isLessThanOrEqualTo(CHADType b) throws CHADException
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if ((getType() != CHADTYPE_INT) && (getType() != CHADTYPE_STRING) &&
                (b.getType() != CHADTYPE_INT) &&
                (b.getType() != CHADTYPE_STRING))
        {
            String errorMessage = "< operator can only be used to compare int and st
            throw new CHADException(errorMessage);
        }

        if (!static_checking)
        {
            if (getType() == b.getType())
            {
                if (getType() == CHADTYPE_INT)
                {
                    if (getIntData(0) <= b.getIntData(0))
                    {
                        c.setIntValue(1);
                    }
                }
                else if (getType() == CHADTYPE_STRING)
                {
                    if ((getStringData(0).compareTo(b.getStringData(0)) < 0) |
                            (getStringData(0).compareTo(b.getStringData(0)) == 0))
                    {
                        c.setIntValue(1);
                    }
                }
            }
        }

        return c;
    }

    //---------------------------------------------------------------------------
    // LOGICAL CONJUNCTION OPERATORS
    //---------------------------------------------------------------------------
    public CHADType or(CHADType b)
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if (!static_checking)
        {
            if ((getIntData(0) == 1) || (b.getIntData(0) == 1))
            {
                c.setIntValue(1);
            }
        }

        return c;
    }

    public CHADType and(CHADType b)
    {
        CHADType c = new CHADType(CHADTYPE_INT, 0, "");

        if (!static_checking)
        {
            if ((getIntData(0) == 1) && (b.getIntData(0) == 1))
            {
                c.setIntValue(1);
            }
        }

        return c;
    }

    //----------------------------------------------------------------------------
    // functions to create queues and stacks
    //----------------------------------------------------------------------------
    public static CHADType createQueue(AST q) throws CHADException
    {
        CHADType c = null;
        AST storageType = q.getFirstChild();
        AST name = storageType.getNextSibling();

        String var_name = name.getText();
        String var_store_type = storageType.getText();
        System.out.println();

        if (var_store_type.equals("int"))
        {
            c = new CHADType(var_name, CHADTYPE_QUEUE, CHADTYPE_INT, 16);
        }
        else if (var_store_type.equals("string"))
        {
            c = new CHADType(var_name, CHADTYPE_QUEUE, CHADTYPE_STRING, 16);
        }
        else
        {
            String errorMessage = "queue " + var_name +
                " can only contain integers and strings.";
            throw new CHADException(errorMessage);
        }

        return c;
    }

    public static CHADType createStack(AST st) throws CHADException
    {
        CHADType c = null;

        AST storageType = st.getFirstChild();
        AST name = storageType.getNextSibling();

        String var_name = name.getText();
        String var_store_type = storageType.getText();

        if (var_store_type.equals("int"))
        {
            c = new CHADType(var_name, CHADTYPE_STACK, CHADTYPE_INT, 16);
        }
        else if (var_store_type.equals("string"))
        {
            c = new CHADType(var_name, CHADTYPE_STACK, CHADTYPE_STRING, 16);
        }
        else
        {
            String errorMessage = "stack " + var_name +
                " can only contain integers and strings.";
            throw new CHADException(errorMessage);
        }

        return c;
    }
}
```

CHADWalker.g

```
{
/*********************************************************************
** CHADWalker.g
** Tree Walker for CHAD - Used as the interpreter
** @author Haronil Estevez
*********************************************************************/
import java.io.*;
import DataStructures.*;
import java.util.*;
import antlr.CommonAST;
import antlr.ASTIterator;
import antlr.collections.AST;

}

class CHADWalker extends TreeParser;


options
{
  // import vocab from chadAntlr parser
  importVocab = chadAntlr;
}


{
  public boolean checkingFunctions = false;
  // variable type codes
  int CHADTYPE_VOID = 0;
     int CHADTYPE_INT = 1;
     int CHADTYPE_STRING = 2;
      int CHADTYPE_ARRAY = 3;
      int CHADTYPE_QUEUE = 4;
      int CHADTYPE_STACK = 5;
      int CHADTYPE_FUNCTION = 6;

  static CHADType null_data = new CHADType();

  // used as global symbol table
  CHADEnvironment e = new CHADEnvironment();

  // used as symbol table for function names
  CHADEnvironment e2 = new CHADEnvironment();
  // used to store the name of called function
  String called_function = "";

  // used for array list evaluation
  Vector ls = new Vector();
  int previous_type = -1;

  // used for function arguments
  Vector function_variables = new Vector();
  Hashtable argsHolder = new Hashtable();

  // used to store return value of function
  CHADType rvalue = null;

  // used by userDefined function to traverse
  // tree
  public AST root = null;

  // used to store updates to visible
  // CHADType variables to be used
  // by the CHADGUI class
  public Vector changes = new Vector();
}


/**********************************************************************
** Evaluates AST nodes
**********************************************************************/
expr returns [CHADType c] throws CHADException
{
  CHADType[] v;
  CHADType a,b;
  CHADType d;
  c = null_data;
  b= null_data;

  // add keywords to symbol table
  CHADType integer = new CHADType("int",CHADTYPE_VOID,0,"");
  CHADType string = new CHADType("string",CHADTYPE_VOID,0,"");
  CHADType array = new CHADType("array",CHADTYPE_VOID,0,"");
  CHADType queue = new CHADType("queue",CHADTYPE_VOID,0,"");
  CHADType stack = new CHADType("stack",CHADTYPE_VOID,0,"");

  CHADType min = new CHADType("min",CHADTYPE_INT,0,"");
  CHADType max = new CHADType("max",CHADTYPE_INT,0,"");
  CHADType show = new CHADType("show",CHADTYPE_VOID,0,"");
  CHADType hide = new CHADType("hide",CHADTYPE_VOID,0,"");
  CHADType print = new CHADType("print",CHADTYPE_VOID,0,"");
    CHADType length = new CHADType("length",CHADTYPE_INT,0,"");


  e.addToCurrent(min);
  e.addToCurrent(max);
  e.addToCurrent(integer);
  e.addToCurrent(string);
  e.addToCurrent(array);
  e.addToCurrent(queue);
  e.addToCurrent(stack);
  e.addToCurrent(show);
```

```
    e.addToCurrent(hide);
    e.addToCurrent(print);
    e.addToCurrent(length);
  }
  :

    #(PLUS a=expr b=expr)
    {
            c = a.plus(b);

            if ((c.getType() == CHADTYPE_INT) &&
                    ((c.getIntData(0) > 999) || (c.getIntData(0) < -999)))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "+" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
            else if ((c.getType() == CHADTYPE_STRING) &&
                    (c.getStringData(0).length() > 30))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "+" + b.getName() +
                        " is too long a string.");
                changes.add(d);
            }
        }
    | #(PLUSEQ a=expr b=expr)
    {
            c = a.plus(b);

            if ((c.getType() == CHADTYPE_INT) &&
                    ((c.getIntData(0) > 999) || (c.getIntData(0) < -999)))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "+" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
            else if ((c.getType() == CHADTYPE_STRING) &&
                    (c.getStringData(0).length() > 30))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "+" + b.getName() +
                        " is too long a string.");
                changes.add(d);
            }
        }
    | #(MINUS a=expr b=expr)
    {
            c = a.minus(b);

            if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "-" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
    }

| #(UMINUS a=expr) { c = a.uminus(); }

| #(MINUSEQ a=expr b=expr)
{
        c = a.minus(b);

        if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
        {
            d = new CHADType("print", CHADTYPE_STRING, -1,
                    "Error:  " + a.getName() + "-" + b.getName() +
                    " exceeds integer bounds.");
            changes.add(d);
        }
   }


| #(MULT a=expr b=expr)
{
        c = a.mult(b);

        if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
        {
            d = new CHADType("print", CHADTYPE_STRING, -1,
                    "Error:  " + a.getName() + "*" + b.getName() +
                    " exceeds integer bounds.");
            changes.add(d);
        }
   }

| #(MULTEQ a=expr b=expr)
{
        c = a.mult(b);

        if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
        {
            d = new CHADType("print", CHADTYPE_STRING, -1,
                    "Error:  " + a.getName() + "-" + b.getName() +
                    " exceeds integer bounds.");
            changes.add(d);
        }
    }

| #(DIV a=expr b=expr)
{
        c = a.div(b);

        if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
        {
            d = new CHADType("print", CHADTYPE_STRING, -1,
                    "Error:  " + a.getName() + "/" + b.getName() +
                    " exceeds integer bounds.");
            changes.add(d);
        }
    }

| #(DIVEQ a=expr b=expr)
{
```

```
            c = a.div(b);

            if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "/" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
        }

    | #(MOD a=expr b=expr)
    {
            c = a.mod(b);

            if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "%" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
        }

    | #(MODEQ a=expr b=expr)
    {
            c = a.mod(b);

            if ((c.getIntData(0) > 999) || (c.getIntData(0) < -999))
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + a.getName() + "%" + b.getName() +
                        " exceeds integer bounds.");
                changes.add(d);
            }
        }

    | #(INCREMENT c=expr)
    {
            c.increment();

            if (c.getIntData(0) > 999)
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + c.getName() +
                        "+1 is too exceeds the bounds on integers.");
                changes.add(d);
            }
        }

    | #(DECREMENT c=expr)
    {
            c.decrement();

            if (c.getIntData(0) < -999)
            {
                d = new CHADType("print", CHADTYPE_STRING, -1,
                        "Error:  " + c.getName() +
                        "+1 is too exceeds the bounds on integers.");
                changes.add(d);
            }
        }

    | #(EQ a=expr b=expr) { c = a.isEqual(b); }

    | #(GT a=expr b=expr) { c=a.isGreaterThan(b); }

    | #(GE a=expr b=expr) { c=a.isGreaterThanOrEqualTo(b); }

    | #(LT a=expr b=expr) { c=a.isLessThan(b); }

    | #(LE a=expr b=expr) { c=a.isLessThanOrEqualTo(b); }


    | #("OR" a=expr right_or:.) { c = a.or(expr(#right_or)); }

    | #("AND" a=expr right_and:.) { c = a.and(expr(#right_and)); }

    | #(IF_STMT bl:.)

    | "endfunction"

    | #(nn:IFBLOCK (rr:.)*)
     {
            if (!checkingFunctions)
            {
                AST inner = null;
                c = null;
                b = null;

                nn = nn.getFirstChild();

                AST control = nn.getFirstChild();
                a = expr(control);

                if (control.getNextSibling() != null)
                {
                    inner = control.getNextSibling();
                }

                if ((inner != null) && (a.getIntData(0) == 1))
                {

                    while (inner != null)
                    {
                        c = expr(inner);
                        inner = inner.getNextSibling();
                    }
                }
                else
                {
                    if (nn.getNextSibling() != null)
                    {
                        nn = nn.getNextSibling();

                        int numOfSiblings = 0;

                        while (nn != null)
                        {
                            b = expr(nn);

                            if (b.getIntData(0) == 1)
                            {

                                int numOfstmts = nn.getNumberOfChildren();

                                if (numOfstmts != 0)
                                {
                                    nn = nn.getFirstChild();

                                    while (nn != null)
                                    {
                                        expr(nn);
                                        nn = nn.getNextSibling();
                                    }
                                }

                                break;
                            }

                            numOfSiblings++;
                            nn = nn.getNextSibling();
                        }
                    }
                }
            }
        }

    | #(ELSEIF_STMT (c=expr)?) { ; }

    | #(ELSE_STMT (est:.)?) { c = new CHADType(CHADTYPE_INT,1,""); }

    | #(fst:FOR_STMT first:. second:.  third:. (fourth:.)* )
    {
            if (!checkingFunctions)
            {
                int childCount = 0;
                AST fff = fst.getFirstChild();
                AST init = fff;
                AST control = init.getNextSibling();
                AST change = control.getNextSibling();
                AST inside = change.getNextSibling();
                boolean doneLoop = false;

                while (fff != null)
                {
                  childCount++;
                    if (childCount == 4)
                    {
                        AST in = fff;

                        int count = 0;

                        for (expr(init); (expr(control)).getIntData(0) == 1;
                                expr(change))
                        {
                            in = fff;
```

```
                            count++;

                            while (in != null)
                            {
                                CHADType c2 = expr(in);
                                in = in.getNextSibling();
                            }
                        }

                        break;
                    }

                    fff = fff.getNextSibling();
                }
            }
        }

    | #(SQA_DECL c=expr)

    | q:"queue"
    {
            c = CHADType.createQueue(q);
            e.addToCurrent(c);
        }

    | st:"stack"
    {
            c = CHADType.createStack(st);
            e.addToCurrent(c);
        }

    | (ar:"array")
     {
            AST storageType = ar.getFirstChild();
            AST size = storageType.getNextSibling();
            AST name = size.getNextSibling();
            int sizeOfArray = Integer.parseInt(size.getText());

            String var_name = name.getText();
            String var_store_type = storageType.getText();

            if (var_store_type.equals("int"))
            {
                c = new CHADType(var_name, CHADTYPE_ARRAY, CHADTYPE_INT, sizeOfArray);
            }
            else if (var_store_type.equals("string"))
            {
                c = new CHADType(var_name, CHADTYPE_ARRAY, CHADTYPE_STRING,
                        sizeOfArray
                    );
            }

            // if array is declared with list of elements,
            // repeatedly invoke array_list function to get array element list
            // and insert each into CHADType
            AST elements = ar.getNextSibling();

            while (elements != null)
            {
                array_list(elements);
                elements = elements.getNextSibling();
            }

            // add elements to declared array
            for (int i = 0; i < ls.size(); i++)
            {
                CHADType t = (CHADType) ls.elementAt(i);
                c.insert(i, t.getIntData(0), t.getStringData(0));
            }

            // assign zero to all entries not declared in { }
            for (int i = ls.size(); i < sizeOfArray; i++)
            {
                c.insert(i, 0, "");
            }

            // reset previous_type variable
            previous_type = -1;

            // reset Vector used for holding list of entries
            ls.removeAllElements();

            // add variable to symbol table
            e.addToCurrent(c);
        }

      // uses array_entry function to get array entry
    | #(ARRAY_ENTRY c=array_entry){  }

    | #(as:ASGN a=expr b=expr)
    {
            a.asgn(b);

            try
            {
                d = e.find(a.getName());

                if (d != null)
                {
                    c = (CHADType) ObjectCloner.deepCopy(d);

                    if (d.getVisible())
                    {
                        c.show();
                        changes.add(c);
                    }
                }
            }
            catch (Exception e)
            {
                ;
            }
        }

    | #(alist:ARRAY_LIST b=expr)
    {
            // use array_list method to loop through list
            AST elements = alist.getFirstChild();
        while (elements != null)
        {
            array_list(elements);
            elements = elements.getNextSibling();
        }

        // create array to store elements
        CHADType t = (CHADType) ls.elementAt(0);
        int Arraysize = ls.size();
        c = new CHADType("", CHADTYPE_ARRAY, t.getType(), Arraysize);

  // insert elements into array
        for (int i = 0; i < ls.size(); i++)
        {
            t = (CHADType) ls.elementAt(i);
            c.insert(i, t.getIntData(0), t.getStringData(0));
        }

        // reset previous_type variable
        previous_type = -1;

        // reset Vector used for holding list of entries
        ls.removeAllElements();
    }

| #("int" int_name:ID)
    {
       c = new CHADType(int_name.getText(), CHADTYPE_INT, 0, "");
       // add variable to symbol table
       e.addToCurrent(c);
    }

| #("string" string_name:ID)
      {

  c = new CHADType(string_name.getText(), CHADTYPE_STRING, 0, "");

  // add variable to symbol table
  e.addToCurrent(c);

   }

| var_id:ID
    {
        if (e.find(var_id.getText()) != null)
        {
            c = e.find(var_id.getText());
        }
        else
        {
            if (var_id.getText() != null)
            {
                c = e2.find(var_id.getText());
            }
        }
    }

| str_val:STRING
```

```
{
  c = new CHADType(CHADTYPE_STRING,0,str_val.getText());
}

| int_val:INTEGER
{
  c = new CHADType(CHADTYPE_INT,Integer.parseInt(int_val.getText()),null);

}

| #(fca:COLLECTION_FUNC_CALL a=expr (dt:DOT) (func:ID) (elist:EXPR_LIST)?)
{
        // handle array,queue,and stack built-in functions
        if (dt != null)
        {
            if (func.getText().equals("length"))
            {
                c = new CHADType(CHADTYPE_INT, a.getLength(), "");
            }

            if (func.getText().equals("push") |
                    func.getText().equals("enqueue"))
            {
                AST ex = func.getNextSibling();
                AST arg = ex.getFirstChild();

                b = expr(arg);

                if (func.getText().equals("push"))
                {
                        a.push(b);
                }
                else
                {
       a.enqueue(b);
                }

                if (a.getVisible())
                {
                    try
                    {
                        d = (CHADType) ObjectCloner.deepCopy(a);
                        changes.add(d);
                    }
                    catch (Exception e)
                    {
                        ;
                    }
                }
            }

            if (func.getText().equals("pop") |
                    func.getText().equals("dequeue"))
            {

                if (func.getText().equals("pop"))
                {

                    c = a.pop();
```

```
    }
    else
    {
        c = a.dequeue();
    }

    if (a.getVisible())
    {
        try
        {
            d = (CHADType) ObjectCloner.deepCopy(a);
            changes.add(d);
        }
        catch (Exception e)
        {
            ;
        }
    }
}

if (func.getText().equals("swap") |
        func.getText().equals("sortAZ") |
        func.getText().equals("sortZA"))
{

    AST first_arg = elist.getFirstChild();
    AST second_arg = first_arg.getNextSibling();

    b = expr(first_arg);
    d = expr(second_arg);

    if (func.getText().equals("swap"))
    {
        a.swap(b, d);
    }
    else if (func.getText().equals("sortAZ"))
    {
        a.sortAZ(b, d);
    }
    else
    {
        a.sortZA(b, d);
    }

    c = a;
    c.primaryFocus = b.getIntData(0);
    c.secondaryFocus = d.getIntData(0);

    try
    {
        d = e.find(a.getName());

        if (d != null)
        {
            c = (CHADType) ObjectCloner.deepCopy(d);

            if (d.getVisible())
            {
```

```
                                c.show();
                                changes.add(c);
                            }
                        }
                    }
                    catch (Exception e)
                    {
                        ;
                    }
                }
            }
        }

    | #(fca2:FUNC_CALL a=expr (elist2:EXPR_LIST)?)
    {
            // used to store passed arguments
            Vector passedArgs = new Vector();

            // handle min, max,show, hide and user-defined functions
            if (a.getName().equals("show"))
            {

                AST first_arg = elist2.getFirstChild();

                b = expr(first_arg);
                b.show();

                try
                {
                    d = (CHADType) ObjectCloner.deepCopy(b);
                    changes.add(d);
                }
                catch (Exception e)
                {
                    ;
                }
            }
            else if (a.getName().equals("hide"))
            {

                AST first_arg = elist2.getFirstChild();

                b = expr(first_arg);
                b.hide();

                try
                {
                    d = (CHADType) ObjectCloner.deepCopy(b);
                    changes.add(d);
                }
                catch (Exception e)
                {
                    ;
                }
            }

            else if (a.getName().equals("print"))
            {
```

```
            AST first_arg = elist2.getFirstChild();

            b = expr(first_arg);

            CHADType printString = new CHADType("print", CHADTYPE_STRING, 0, "");
            printString.setStringValue(b.getStringData(0));
            changes.add(printString);
        }

        else if (a.getName().equals("min"))
        {

            AST first_arg = elist2.getFirstChild();
            AST second_arg = first_arg.getNextSibling();

            a = expr(first_arg);
            b = expr(second_arg);

            c = CHADType.min(a, b);
        }

        else if (a.getName().equals("max"))
        {

            AST first_arg = elist2.getFirstChild();
            AST second_arg = first_arg.getNextSibling();

            a = expr(first_arg);
            b = expr(second_arg);

            c = CHADType.max(a, b);
        }

        // handle user-defined functions
        else
        {
            // get function from environment's symbol table
            CHADType f = e.find(a.getName());

            if (f == null)
            {
                f = e2.find(a.getName());
            }

            if (elist2 == null)
            {

                called_function = a.getName();

                if (checkingFunctions)
                {
                    c = new CHADType(f.getStorageType(), 0, "");
                }
                else
                {
                    userDefined(root);
                    c = rvalue;
                }
            }
            // loop through arguments
            else
            {
                int numOfArguments = elist2.getNumberOfChildren();
                int argNum = 0;

                AST arg = elist2.getFirstChild();

                while (numOfArguments > 0)
                {
                    // argument from actual function call
                    b = expr(arg);

                    // argument from function call definition
                    CHADType t = f.getArrayEntry(argNum);

                    arg = arg.getNextSibling();
                    numOfArguments--;
                    argNum++;

                    // store arguments to be used within function
                    passedArgs.add(b);
                }
            }

            called_function = a.getName();
            argsHolder.put(called_function, passedArgs);

            // call userDefined() to get return value
            // of UserDefined function
            if (checkingFunctions)
            {
                if (f.getStorageType() == CHADTYPE_ARRAY)
                {
                    c = new CHADType(a.getName(), CHADTYPE_ARRAY,
                            f.return_storage_type, f.return_array_length);

                    for (int i = 0; i < f.return_array_length; i++)
                    {
                        c.insert(i, 0, "");
                    }
                }

                else
                {
                    c = new CHADType(f.getStorageType(), 0, "");
                }
            }
            else
            {
                userDefined(root);
                c = rvalue;
            }
        }
    }

| fn:"function"
| #(fd:FUNC_DECL c=expr)

| #(RETURN (c=expr)?)

| #(FUNC_ARGS fr:.)

| #(STATEMENT (stmt:. {  c = expr(#stmt);} )*)

 {

 }
;

/*********************************************************************
** Code for getting an array entry
*********************************************************************/
array_entry returns [CHADType c] throws CHADException
{
  CHADType a,b;
  c = null_data;
  int arrayLength = 0;
}
:(arr_id:ID)
{

        int index = 0;

        // get the array from environment
        a = e.find(arr_id.getText());

        // get length of array
        arrayLength = a.getLength();

        // get AST node containing array index
        AST ind = arr_id.getFirstChild();

        CHADType t = expr(ind);

        index = t.getIntData(0);

        // exit if array index out of bounds
        if ((index < 0) || (index > (arrayLength - 1)))
        {
            String errorMessage = "ARRAY " + arr_id.getText() +
                " INDEX OUT OF BOUNDS!!!";
            CHADType d = new CHADType("print", CHADTYPE_STRING, -1,
            "Error:  " + errorMessage);
             changes.add(d);
             c = new CHADType(a.getStorageType(),0,"");
        }
  else
          c = a.getArrayEntry(index);

        if (checkingFunctions)
        {
            c = new CHADType(a.getStorageType(), 0, "");
        }
        else
        {
            c = a.getArrayEntry(index);
        }

        // set name of array entry to name of array itself
        // so as to faciliate error messages
        if (!checkingFunctions)
        {
            c.setName(arr_id.getText());
        }

        if (a.getVisible())
        {
            a.primaryFocus = index;

            try
            {
                CHADType d = (CHADType) ObjectCloner.deepCopy(a);
                changes.add(d);
            }
            catch (Exception e)
            {
                ;
            }
        }
    }

{    };

/**********************************************************************
** Code for evaluating array declaration list
**********************************************************************/
    array_list throws CHADException
{

}
:

an:.
{
        CHADType d = expr(an);

        try
        {
            CHADType d2 = (CHADType) ObjectCloner.deepCopy(d);
            ls.add(d2);
        }
        catch (Exception e)
        {
            ;
        }

        previous_type = d.getType();
    }

{ };

/**********************************************************************
** Code for user-defined function call handling
**********************************************************************/
```

```
userDefined returns [CHADType c] throws CHADException
{

  CHADType a;
  c = null_data;
  int function_found = 0;
    }

    :
    #(PLUS a=userDefined a=userDefined) { }
| #(PLUSEQ a=userDefined a=userDefined) { }

| #(MINUS a=userDefined a=userDefined) { }

| #(UMINUS a=userDefined) {  }

| #(MINUSEQ a=userDefined a=userDefined) { }

| #(MULT a=userDefined a=userDefined) { }

| #(MULTEQ a=userDefined a=userDefined) { }

| #(DIV a=userDefined a=userDefined) { }

| #(DIVEQ a=userDefined a=userDefined) { }

| #(MOD a=userDefined a=userDefined) { }

| #(MODEQ a=userDefined a=userDefined) { }

| #(INCREMENT a=userDefined) {  }

| #(DECREMENT a=userDefined) {  }

| #(EQ a=userDefined a=userDefined) {  }

| #(GT a=userDefined a=userDefined) {  }

| #(LT a=userDefined a=userDefined) {  }

| #(LE a=userDefined a=userDefined) {  }

| #(IFBLOCK a=userDefined)
{
}

| #("OR" a=userDefined) {  }

| #("AND" a=userDefined) {  }

| #(nn:IF_STMT a=userDefined itt:.){ }

| #(ELSEIF_STMT (a=userDefined)?) { ; }

| #(ELSE_STMT (et:.)?) {  }

| #(FOR_STMT yy:. yu:. kk:. km:.) {   }

| #(SQA_DECL a=userDefined)
| "queue" {  }

| "stack"{ }

| ("array"){ }

  // uses array_entry function to get array entry
| #(ARRAY_ENTRY ae:.){ }

| #(ASGN a=userDefined a=userDefined) { }

| #(ARRAY_LIST a=userDefined){   }

| #("int" ID){  }

| #("string" ID){}

| ID { }

| STRING { }

| INTEGER{}

| #(FUNC_CALL a=userDefined (DOT)? (ID)? (EXPR_LIST)?)

| fn:"function"
{

}

    | #(gh:FUNC_DECL ft:.)
{
        AST fargs = ft.getFirstChild();

        int return_type = 0;
        int return_storage_type = 0;
        String function_name = "";

        // loop through args list
        if (fargs.getNumberOfChildren() != 0)
        {
            CHADType nf = null;
            AST list = fargs.getFirstChild();

            for (int i = 0; i < fargs.getNumberOfChildren(); i++)
            {
                // determine function return type
                if (i == 0)
                {
                    if (list.getText().equals("int"))
                    {
                        return_type = CHADTYPE_INT;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_INT, (fargs.getNumberOfChildren() / 2));
                    }
                    else if (list.getText().equals("string"))
                    {
                        return_type = CHADTYPE_STRING;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_STRING,
                                (fargs.getNumberOfChildren() / 2));
                    }

                    else if (list.getType() == #ARRAY_RETURN)
                    {
                        return_type = CHADTYPE_ARRAY;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_ARRAY,
                                (fargs.getNumberOfChildren() / 2));

                        AST ty = list.getFirstChild();

                        if (ty.getText().equals("int"))
                        {
                            nf.return_storage_type = CHADTYPE_INT;
                            return_storage_type = CHADTYPE_INT;
                        }
                        else
                        {
                            nf.return_storage_type = CHADTYPE_STRING;
                            return_storage_type = CHADTYPE_STRING;
                        }
                    }

                    else
                    {
                        return_type = CHADTYPE_VOID;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_VOID, (fargs.getNumberOfChildren() / 2));
                    }
                }

                // add function name to environment
                else if (i == 1)
                {
                    function_name = list.getText();
                    nf.setName(function_name);

                    // check if function name matches called function
                    if (function_name.equals(called_function))
                    {
                        function_found = 1;
                    }

                    //e.addToCurrent(nf);
                    // create a new scope and enter it
                    e.createNewScope();
                    e.enterScope();
                }

                // add argument types
                else
                {
                    int arg_type = 0;

                    if ((i % 2) == 0)
                    {
```

```
CHADType atype = null;

if (list.getText().equals("int"))
{
    AST arg_name = list.getNextSibling();

    String argument_name = arg_name.getText();

    atype = new CHADType(argument_name, CHADTYPE_INT,
            0, "");

    e.addToCurrent(atype);

    // used to remove function arguments
    // from symbol table at end of function
    function_variables.add(argument_name);

    arg_type = CHADTYPE_INT;
}

else if (list.getText().equals("string"))
{
    AST arg_name = list.getNextSibling();

    String argument_name = arg_name.getText();

    atype = new CHADType(argument_name,
            CHADTYPE_STRING, 0, "");

    e.addToCurrent(atype);

    arg_type = CHADTYPE_STRING;

    // used to remove function arguments
    // from symbol table at end of function
    function_variables.add(argument_name);
}

else if (list.getType() == #ARRAY_ARGUMENT)
{
    AST ty = list.getFirstChild();
    AST Asize = ty.getNextSibling();
    AST arg_name = list.getNextSibling();

    int arrayType = 0;

    if (ty.getText().equals("int"))
    {
        arrayType = CHADTYPE_INT;
    }
    else
    {
        arrayType = CHADTYPE_STRING;
    }

    int arraySize = Integer.parseInt(Asize.getText());

    String argument_name = arg_name.getText();
```

```
                        atype = new CHADType(argument_name, CHADTYPE_ARRAY,
                                arrayType, arraySize);
                        e.addToCurrent(atype);

                        arg_type = CHADTYPE_ARRAY;

                        // used to remove function arguments
                        // from symbol table at end of function
                        function_variables.add(argument_name);
                    }

                    // insert list of types of arguments
                    // in array Data of CHADType corresponding
                    // to the function
                    nf.insert(((i / 2) - 1), arg_type, "");
                }
            }

            list = list.getNextSibling();
        }
    }

    // Evaluate body of found function
    if (function_found == 1)
    {
        Vector passedArgs = (Vector) argsHolder.get(called_function);

        // set function arguments to passed arguments
        int argSize = function_variables.size();

        for (int i = 0; i < argSize; i++)
        {
            String name = (String) function_variables.elementAt(0);
            function_variables.removeElementAt(0);

            CHADType t2 = (CHADType) passedArgs.elementAt(i);

            if (t2.getType() == CHADTYPE_INT)
            {
                if (e.find(name) != null)
                {
                    e.find(name).setIntValue(t2.getIntData(0));
                }
            }
            else if (t2.getType() == CHADTYPE_STRING)
            {
                if (e.find(name) != null)
                {
                    e.find(name).setStringValue(t2.getStringData(0));
                }
            }

            else if (t2.getType() == CHADTYPE_ARRAY)
            {
              if(t2.getLength() != e.find(name).getLength())
              {
                String errorMessage = "array argument not correct length!!!";
              CHADType d = new CHADType("print", CHADTYPE_STRING, -1,
                           "Error:  " + errorMessage);
```

```
                            changes.add(d);

                            CHADType ar = e.find(name);

                            if(ar != null)
                            {
                              for(int ju = 0; ju < ar.getLength(); ju++)
                              {
                                ar.arrayData[ju] = new CHADType(ar.getStorage
                              }
                            }

               }

                   else if (e.find(name) != null)
                   {
                       e.find(name).setArrayData(t2.arrayData);
                   }
             }
         }

         AST body = fargs;
         expr(fargs);

         AST rstmt = null;

         // check inside of function until
         // finding return statement
         while (body.getNextSibling() != null)
         {
             body = body.getNextSibling();

             if (body.getType() == #RETURN)
             {
                 rstmt = body;
             }

             CHADType inFunct = expr(body);
         }


   // evaluate return type
         AST rs = rstmt;

         if (rs.getFirstChild() != null)
         {
             rs = rs.getFirstChild();

             CHADType rt = expr(rs);

             // return appropriate type
             if (return_type == CHADTYPE_INT)
             {
                 c = new CHADType(CHADTYPE_INT, rt.getIntData(0), "");
                 rvalue = new CHADType(CHADTYPE_INT, rt.getIntData(0), "");
             }
             else if (return_type == CHADTYPE_STRING)
             {
                 c = new CHADType(CHADTYPE_STRING, 0, rt.getStringData(0));
```

```
                    rvalue = new CHADType(CHADTYPE_STRING, 0,
                            rt.getStringData(0));
                }
                else if (return_type == CHADTYPE_ARRAY)
                {
                    e2.find(function_name).return_array_length = rt.getLength();
                    rvalue = new CHADType(function_name, CHADTYPE_ARRAY,
                            return_storage_type, rt.getLength());
                    rvalue.arrayData = rt.arrayData;
                }
            }
            else
            {
                // return void type
                c = new CHADType(CHADTYPE_VOID, 0, "");
                rvalue = new CHADType(CHADTYPE_VOID, 0, "");
            }
        }

        // reset vector
        function_variables.removeAllElements();

        // exit scope
        e.exitScope();
    }

   // walk tree until called function is found
  | #(STATEMENT (stmt:. {c =userDefined(#stmt);} )*)
  {


    checkingFunctions = false;
   };
```

Hashable.java

```
/*********************************************************************************
 ** Hashable.java
 *********************************************************************************
    package DataStructures;

    /**
     * Protocol for Hashable objects.
     * @author Mark Allen Weiss
     */
    public interface Hashable
    {
        /**
         * Compute a hash function for this object.
         * @param tableSize the hash table size.
         * @return (deterministically) a number between
         *     0 and tableSize-1, distributed equitably.
         */
        int hash( int tableSize );
    }
```

HashEntry.java

```
/*********************************************************************************
 ** HashEntry.java
 ** @author Mark Allen Weiss
 *********************************************************************************
```

```
    package DataStructures;

    // The basic entry stored in ProbingHashTable

    class HashEntry
    {
        Hashable element;   // the element
        boolean  isActive;  // false is deleted

        public HashEntry( Hashable e )
        {
            this( e, true );
        }

        public HashEntry( Hashable e, boolean i )
        {
            element   = e;
            isActive  = i;
        }
    }
```

Makefile

```
############################################################
### MAKEFILE
############################################################
all:
  java antlr.Tool chad.g
  java antlr.Tool CHADWalker.g
  java antlr.Tool StaticCHADWalker.g
  javac ./DataStructures/*.java
  javac *.java

grammar:
  java antlr.Tool chad.g

static:
  java antlr.Tool StaticCHADWalker.g

walker:
  java antlr.Tool CHADWalker.g
run:
  java CHADGUI
```

ObjectCloner.java

```
/*******************************************************************************
 ** ObjectCloner.java
 ** class used to make a deep copy of serializable objects
 ** @author Dave Miller
 *******************************************************************************
import java.awt.*;
import java.io.*;
import java.util.*;


public class ObjectCloner
{
    // so that nobody can accidentally create an ObjectCloner object
    private ObjectCloner()
    {
    }

    // returns a deep copy of an object
    static public Object deepCopy(Object oldObj) throws Exception
    {
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;

        try
        {
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            oos = new ObjectOutputStream(bos);

            // serialize and pass the object
            oos.writeObject(oldObj);
            oos.flush();

            ByteArrayInputStream bin = new ByteArrayInputStream(bos.toByteArray());
            ois = new ObjectInputStream(bin);

            // return the new object
            return ois.readObject();
        }
        catch (Exception e)
        {
            System.out.println("Exception in ObjectCloner = " + e);
            throw (e);
        }
        finally
        {
            oos.close();
            ois.close();
        }
    }
}
```

Overflow.java

```
/*****************************************************************************
 ** Overflow.java
 *****************************************************************************
    package DataStructures;

    /**
     * Exception class for access in full containers
     * such as stacks, queues, and priority queues.
     * @author Mark Allen Weiss
     */
    public class Overflow extends Exception
    {
    }
```

QuadraticProbingHashTable.java

```
/*****************************************************************************
 ** QuadraticProbingHashTable.java
 *****************************************************************************

package DataStructures;
    // QuadraticProbingHashTable abstract class
    //
```

```java
// CONSTRUCTION: with an approximate initial size or a default.
//
// ******************PUBLIC OPERATIONS*********************
// void insert( x )       --> Insert x
// void remove( x )       --> Remove x
// Hashable find( x )     --> Return item that matches x
// void makeEmpty( )      --> Remove all items
// int hash( String str, int tableSize )
//                        --> Static method to hash strings

/**
 * Probing table implementation of hash tables.
 * Note that all "matching" is based on the equals method.
 * @author Mark Allen Weiss
 */
public class QuadraticProbingHashTable
{
    /**
     * Construct the hash table.
     */
    public QuadraticProbingHashTable( )
    {
        this( DEFAULT_TABLE_SIZE );
    }

    /**
     * Construct the hash table.
     * @param size the approximate initial size.
     */
    public QuadraticProbingHashTable( int size )
    {
        allocateArray( size );
        makeEmpty( );
    }

    /**
     * Insert into the hash table. If the item is
     * already present, do nothing.
     * @param x the item to insert.
     */
    public void insert( Hashable x )
    {
            // Insert x as active
        int currentPos = findPos( x );
        if( isActive( currentPos ) )
            return;

        array[ currentPos ] = new HashEntry( x, true );

            // Rehash; see Section 5.5
        if( ++currentSize > array.length / 2 )
            rehash( );
    }

    /**
     * Expand the hash table.
     */
    private void rehash( )
    {
            HashEntry [ ] oldArray = array;

                // Create a new double-sized, empty table
            allocateArray( nextPrime( 2 * oldArray.length ) );
            currentSize = 0;

                // Copy table over
            for( int i = 0; i < oldArray.length; i++ )
                if( oldArray[ i ] != null && oldArray[ i ].isActive )
                    insert( oldArray[ i ].element );

            return;
        }

        /**
         * Method that performs quadratic probing resolution.
         * @param x the item to search for.
         * @return the position where the search terminates.
         */
        private int findPos( Hashable x )
        {
/* 1*/      int collisionNum = 0;
/* 2*/      int currentPos = x.hash( array.length );

/* 3*/      while( array[ currentPos ] != null &&
                    !array[ currentPos ].element.equals( x ) )
            {
/* 4*/          currentPos += 2 * ++collisionNum - 1;  // Compute ith probe
/* 5*/          if( currentPos >= array.length )       // Implement the mod
/* 6*/              currentPos -= array.length;
            }

/* 7*/      return currentPos;
        }

        /**
         * Remove from the hash table.
         * @param x the item to remove.
         */
        public void remove( Hashable x )
        {
            int currentPos = findPos( x );
            if( isActive( currentPos ) )
                array[ currentPos ].isActive = false;
        }

        /**
         * Find an item in the hash table.
         * @param x the item to search for.
         * @return the matching item.
         */
        public Hashable find( Hashable x )
        {
            int currentPos = findPos( x );
      return isActive( currentPos ) ? array[ currentPos ].element : null;
        }

        /**
         * Return true if currentPos exists and is active.
```

```
     * @param currentPos the result of a call to findPos.
     * @return true if currentPos is active.
     */
    private boolean isActive( int currentPos )
    {
        return array[ currentPos ] != null && array[ currentPos ].isActive;
    }

    /**
     * Make the hash table logically empty.
     */
    public void makeEmpty( )
    {
        currentSize = 0;
        for( int i = 0; i < array.length; i++ )
            array[ i ] = null;
    }

    /**
     * A hash routine for String objects.
     * @param key the String to hash.
     * @param tableSize the size of the hash table.
     * @return the hash value.
     */
    public static int hash( String key, int tableSize )
    {
        int hashVal = 0;

        for( int i = 0; i < key.length( ); i++ )
            hashVal = 37 * hashVal + key.charAt( i );

        hashVal %= tableSize;
        if( hashVal < 0 )
            hashVal += tableSize;

        return hashVal;
    }

    private static final int DEFAULT_TABLE_SIZE = 11;

        /** The array of elements. */
    private HashEntry [ ] array;   // The array of elements
    private int currentSize;       // The number of occupied cells

    /**
     * Internal method to allocate array.
     * @param arraySize the size of the array.
     */
    private void allocateArray( int arraySize )
    {
        array = new HashEntry[ arraySize ];
    }

    /**
     * Internal method to find a prime number at least as large as n.
     * @param n the starting number (must be positive).
     * @return a prime number larger than or equal to n.
     */
    private static int nextPrime( int n )
```

```
        {
            if( n % 2 == 0 )
                n++;

            for( ; !isPrime( n ); n += 2 )
                ;

            return n;
        }

        /**
         * Internal method to test if a number is prime.
         * Not an efficient algorithm.
         * @param n the number to test.
         * @return the result of the test.
         */
        private static boolean isPrime( int n )
        {
            if( n == 2 || n == 3 )
                return true;

            if( n == 1 || n % 2 == 0 )
                return false;

            for( int i = 3; i * i <= n; i += 2 )
                if( n % i == 0 )
                    return false;

            return true;
        }

        public int numOfEntries()
        {
          int return_val = 0;

          for(int i=0; i < array.length; i++)
          {
            if(array[i] != null)
              return_val++;
          }

          return return_val;
        }


    }
```

QueueAr.java

```
/*************************************************************************************
 ** QueueAr.java
 *************************************************************************************

package DataStructures;

    // QueueAr class
    //

    // CONSTRUCTION: with or without a capacity; default is 10
    //
```

```java
// ******************PUBLIC OPERATIONS*********************
// void enqueue( x )      --> Insert x
// Object getFront( )     --> Return least recently inserted item
// Object dequeue( )      --> Return and remove least recent item
// boolean isEmpty( )     --> Return true if empty; else false
// boolean isFull( )      --> Return true if capacity reached
// void makeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// Overflow thrown for enqueue on full queue

/**
 * Array-based implementation of the queue.
 * @author Mark Allen Weiss
 */
public class QueueAr implements java.io.Serializable
{
    /**
     * Construct the queue.
     */
    public QueueAr( )
    {
        this( DEFAULT_CAPACITY );
    }

    /**
     * Construct the queue.
     */
    public QueueAr( int capacity )
    {
        theArray = new Object[ capacity ];
        makeEmpty( );
    }

    /**
     * Test if the queue is logically empty.
     * @return true if empty, false otherwise.
     */
    public boolean isEmpty( )
    {
        return currentSize == 0;
    }

    /**
     * Test if the queue is logically full.
     * @return true if full, false otherwise.
     */
    public boolean isFull( )
    {
        return currentSize == theArray.length;
    }

    /**
     * Make the queue logically empty.
     */
    public void makeEmpty( )
    {
        currentSize = 0;
        front = 0;
        back = -1;
    }

    /**
     * Get the least recently inserted item in the queue.
     * Does not alter the queue.
     * @return the least recently inserted item in the queue, or null, if empty.
     */
    public Object getFront( )
    {
        if( isEmpty( ) )
            return null;
        return theArray[ front ];
    }

    /**
     * Return and remove the least recently inserted item from the queue.
     * @return the least recently inserted item in the queue, or null, if empty.
     */
    public Object dequeue( )
    {
        if( isEmpty( ) )
            return null;
        currentSize--;

        Object frontItem = theArray[ front ];
        theArray[ front ] = null;
        front = increment( front );
        return frontItem;
    }

    /**
     * Insert a new item into the queue.
     * @param x the item to insert.
     * @exception Overflow if queue is full.
     */
    public void enqueue( Object x ) throws Overflow
    {
        if( isFull( ) )
            throw new Overflow( );
        back = increment( back );
        theArray[ back ] = x;
        currentSize++;
    }

    /**
     * Internal method to increment with wraparound.
     * @param x any index in theArray's range.
     * @return x+1, or 0, if x is at the end of theArray.
     */
    public int increment( int x )
    {
        if( ++x == theArray.length )
            x = 0;
        return x;
    }

    public Object [ ] theArray;
    public int        currentSize;
    public int        front;
```

```
        public int        back;

        static final int DEFAULT_CAPACITY = 10;

    }
```

StackAr.java

```
/*****************************************************************************************
 ** StackAr.java
 *****************************************************************************************

package DataStructures;

    // StackAr class
    //
    // CONSTRUCTION: with or without a capacity; default is 10
    //
    // ******************PUBLIC OPERATIONS*********************
    // void push( x )         --> Insert x
    // void pop( )            --> Remove most recently inserted item
    // Object top( )          --> Return most recently inserted item
    // Object topAndPop( )    --> Return and remove most recently inserted item
    // boolean isEmpty( )     --> Return true if empty; else false
    // boolean isFull( )      --> Return true if full; else false
    // void makeEmpty( )      --> Remove all items
    // ******************ERRORS********************************
    // Overflow and Underflow thrown as needed

    /**
     * Array-based implementation of the stack.
     * @author Mark Allen Weiss
     */
    public class StackAr implements java.io.Serializable
    {
        /**
         * Construct the stack.
         */
        public StackAr( )
        {
            this( DEFAULT_CAPACITY );
        }

        /**
         * Construct the stack.
         * @param capacity the capacity.
         */
        public StackAr( int capacity )
        {
            theArray = new Object[ capacity ];
            topOfStack = -1;
        }

        /**
         * Test if the stack is logically empty.
         * @return true if empty, false otherwise.
         */
        public boolean isEmpty( )
        {
            return topOfStack == -1;
```

```java
    }

    /**
     * Test if the stack is logically full.
     * @return true if full, false otherwise.
     */
    public boolean isFull( )
    {
        return topOfStack == theArray.length - 1;
    }

    /**
     * Make the stack logically empty.
     */
    public void makeEmpty( )
    {
        topOfStack = -1;
    }

    /**
     * Get the most recently inserted item in the stack.
     * Does not alter the stack.
     * @return the most recently inserted item in the stack, or null, if empty.
     */
    public Object top( )
    {
        if( isEmpty( ) )
            return null;
        return theArray[ topOfStack ];
    }

    /**
     * Remove the most recently inserted item from the stack.
     * @exception Underflow if stack is already empty.
     */
    public void pop( ) throws Underflow
    {
        if( isEmpty( ) )
            throw new Underflow( );
        theArray[ topOfStack-- ] = null;
    }

    /**
     * Insert a new item into the stack, if not already full.
     * @param x the item to insert.
     * @exception Overflow if stack is already full.
     */
    public void push( Object x ) throws Overflow
    {
        if( isFull( ) )
            throw new Overflow( );
        theArray[ ++topOfStack ] = x;
    }

    /**
     * Return and remove most recently inserted item from the stack.
     * @return most recently inserted item, or null, if stack is empty.
     */
    public Object topAndPop( )
```

```
        {
            if( isEmpty( ) )
                return null;
            Object topItem = top( );
            theArray[ topOfStack-- ] = null;
            return topItem;
        }

        public Object [ ] theArray;
        public int        topOfStack;

        static final int DEFAULT_CAPACITY = 10;

    }
```

StaticCHADWalker.g

```
{
/**********************************************************************
** StaticCHADWalker.g
** Static Tree Walker for CHAD - Used to check static semantics
** @author Haronil Estevez
**********************************************************************/
import java.io.*;
import DataStructures.*;
import java.util.*;
import antlr.CommonAST;
import antlr.ASTIterator;
import antlr.collections.AST;



}

class StaticCHADWalker extends TreeParser;


options
{
  // import vocab from chadAntlr parser
  importVocab = chadAntlr;
  genHashLines = true;
}


{
  public boolean checkingFunctions = false;

  // variable type codes
  int CHADTYPE_VOID = 0;
     int CHADTYPE_INT = 1;
     int CHADTYPE_STRING = 2;
      int CHADTYPE_ARRAY = 3;
      int CHADTYPE_QUEUE = 4;
      int CHADTYPE_STACK = 5;
      int CHADTYPE_FUNCTION = 6;

  static CHADType null_data = new CHADType();

  CHADEnvironment e = new CHADEnvironment();
  // used as symbol table for function names
  CHADEnvironment e2 = new CHADEnvironment();


  // used for array list evaluation
  Vector ls = new Vector();
  int previous_type = -1;

  // used for function arguments
  Vector function_variables = new Vector();

  // used by userDefined function to traverse
  // tree
  public AST root = null;

}


/*********************************************************************
** Evaluates AST nodes and checks static semantics
*********************************************************************/
expr returns [CHADType c] throws Overflow, CHADException
{
  CHADType[] v;
  CHADType a,b;
  CHADType d;
  c = null_data;

  // add keywords to symbol table
  CHADType integer = new CHADType("int",CHADTYPE_VOID,0,"");
  CHADType string = new CHADType("string",CHADTYPE_VOID,0,"");
  CHADType array = new CHADType("array",CHADTYPE_VOID,0,"");
  CHADType queue = new CHADType("queue",CHADTYPE_VOID,0,"");
  CHADType stack = new CHADType("stack",CHADTYPE_VOID,0,"");

  CHADType min = new CHADType("min",CHADTYPE_INT,0,"");
  CHADType max = new CHADType("max",CHADTYPE_INT,0,"");
  CHADType show = new CHADType("show",CHADTYPE_VOID,0,"");
  CHADType hide = new CHADType("hide",CHADTYPE_VOID,0,"");
  CHADType print = new CHADType("print",CHADTYPE_VOID,0,"");
  CHADType length = new CHADType("length",CHADTYPE_INT,0,"");

  e.addToCurrent(min);
  e.addToCurrent(max);
  e.addToCurrent(integer);
  e.addToCurrent(string);
  e.addToCurrent(array);
  e.addToCurrent(queue);
  e.addToCurrent(stack);
  e.addToCurrent(show);
  e.addToCurrent(hide);
  e.addToCurrent(print);
  e.addToCurrent(length);
}
:

  #(PLUS a=expr b=expr) {  c = a.plus(b); }
  | #(PLUSEQ a=expr b=expr) {  c = a.plus(b);}
  | #(MINUS a=expr b=expr) {  c = a.minus(b);}

  | #(UMINUS a=expr) {  c = a.uminus(); }

  | #(MINUSEQ a=expr b=expr) {  c = a.minus(b);}

  | #(MULT a=expr b=expr) {  c = a.mult(b);}

  | #(MULTEQ a=expr b=expr) {  c = a.mult(b);}

  | #(DIV a=expr b=expr) {  c = a.div(b);}

  | #(DIVEQ a=expr b=expr) {  c = a.div(b);}

  | #(MOD a=expr b=expr) {  c = a.mod(b);}

  | #(MODEQ a=expr b=expr) {  c = a.mod(b);}

  | #(INCREMENT c=expr) { c.increment(); }

  | #(DECREMENT c=expr) {  c.decrement(); }

  | #(EQ a=expr b=expr) {  c = a.isEqual(b); }

  | #(GT a=expr b=expr) {  c=a.isGreaterThan(b); }

  | #(GE a=expr b=expr) {  c=a.isGreaterThanOrEqualTo(b); }

  | #(LT a=expr b=expr) {  c=a.isLessThan(b); }

  | #(LE a=expr b=expr) {  c=a.isLessThanOrEqualTo(b); }

  | #(IFBLOCK c=expr)
  {
  }

  | #("OR" a=expr right_or:.) {  c = a.or(expr(#right_or)); }

  | #("AND" a=expr right_and:.) {  c = a.and(expr(#right_and)); }

  | #(nn:IF_STMT a=expr inner:.)
  {
          int numOfstmts = 0;

          if (inner.getNumberOfChildren() == 1)
          {
              numOfstmts = inner.getNumberOfChildren();
          }
          else
          {
              numOfstmts = inner.getNumberOfChildren() + 1;
          }

          while (numOfstmts != 0)
          {
              c = expr(inner);

              if (inner.getNextSibling() != null)
              {
                  inner = inner.getNextSibling();
              }

              numOfstmts--;
          }

          if (nn.getNextSibling() != null)
          {
              nn = nn.getNextSibling();

              int numOfSiblings = 0;

              while (nn != null)
              {
                  b = expr(nn);

                  numOfstmts = nn.getNumberOfChildren();

                  if (numOfstmts != 0)
                  {
                      nn = nn.getFirstChild();

                      while (nn != null)
                      {
                          c = expr(nn);
                          nn = nn.getNextSibling();
                      }
                  }

                  if (nn != null)
                  {
                      numOfSiblings++;
                      nn = nn.getNextSibling();
                  }
              }
          }
      }

  | #(ELSEIF_STMT (c=expr)?) { ; }

  | #(ELSE_STMT (est:.)?) { c = new CHADType(CHADTYPE_INT,1,""); }

  | #(FOR_STMT init:. control:. change:. {Vector lp = new Vector();} (inside:. {lp.a
  {
          expr(init);
          expr(control);
          expr(change);

          for (int i = 0; i < lp.size(); i++)
          {
              inside = (AST) lp.elementAt(i);
              expr(inside);
          }
      }

  | #(SQA_DECL c=expr)

  | q:"queue"
```

```
        {
                c = CHADType.createQueue(q);

                if (e.find(c.getName()) != null)
                {
                    String errorMessage = c.getName() +
                        " has already been declared!!!";
                    throw new CHADException(errorMessage);
                }

          // add variable to symbol table
                e.addToCurrent(c);
            }

        | st:"stack"
        {
                c = CHADType.createStack(st);

                if (e.find(c.getName()) != null)
                {
                    String errorMessage = c.getName() +
                        " has already been declared!!!";
                    throw new CHADException(errorMessage);
                }

          // add variable to symbol table
                e.addToCurrent(c);
           }

        | (ar:"array")
        {
                AST storageType = ar.getFirstChild();
                AST size = storageType.getNextSibling();
                AST name = size.getNextSibling();
                int sizeOfArray = Integer.parseInt(size.getText());

                String var_name = name.getText();
                String var_store_type = storageType.getText();

                if (e.find(name.getText()) != null)
                {
                    String errorMessage = name.getText() +
                        " has already been declared!!!";
                    throw new CHADException(errorMessage);
                }

                if (var_store_type.equals("int"))
                {
                    c = new CHADType(var_name, CHADTYPE_ARRAY, CHADTYPE_INT, sizeOfArray);
                }
                else if (var_store_type.equals("string"))
                {
                    c = new CHADType(var_name, CHADTYPE_ARRAY, CHADTYPE_STRING,
                            sizeOfArray);
                }
                else
                {
                    String errorMessage = "array can only contain integers and strings.";
                    throw new CHADException(errorMessage);
                }

                // if array is declared with list of elements,
                // repeatedly invoke array_list function to get array element list
                // and insert each into CHADType
                AST elements = ar.getNextSibling();

                while (elements != null)
                {
                    array_list(elements);
                    elements = elements.getNextSibling();
                }

                // exit if # of elements in list is out of bounds
                if (ls.size() > sizeOfArray)
                {
                    String errorMessage = "ARRAY " + var_name + " can only hold " +
                        sizeOfArray + " elements!!!";
                    throw new CHADException(errorMessage);
                }

                // add elements to declared array
                for (int i = 0; i < ls.size(); i++)
                {
                    CHADType t = (CHADType) ls.elementAt(i);
                    c.insert(i, 0, "");
                }

                // assign zero to all entries not declared in { }
                for (int i = ls.size(); i < sizeOfArray; i++)
                {
                    c.insert(i, 0, "");
                }

                // reset previous_type variable
                previous_type = -1;

                // reset Vector used for holding list of entries
                ls.removeAllElements();

                // add variable to symbol table
                e.addToCurrent(c);
            }

          // uses array_entry function to get array entry
        | #(ARRAY_ENTRY c=array_entry){ }


        | #(as:ASGN a=expr b=expr) {    a.asgn(b); }

        | #(alist:ARRAY_LIST b=expr)
        {
                // use array_list method to loop through list
                AST elements = alist.getFirstChild();

                while (elements != null)
                {
                    array_list(elements);
                    elements = elements.getNextSibling();
                }

                // create array to store elements
                CHADType t = (CHADType) ls.elementAt(0);
                int Arraysize = ls.size();
                c = new CHADType("", CHADTYPE_ARRAY, t.getType(), Arraysize);

                // reset previous_type variable
                previous_type = -1;

                // reset Vector used for holding list of entries
                ls.removeAllElements();
           }

        | #("int" int_name:ID)
        {
                if (e.find(int_name.getText()) != null)
                {
                    String errorMessage = int_name + " has already been declared!!!";
                    throw new CHADException(errorMessage);
                }

                c = new CHADType(int_name.getText(), CHADTYPE_INT, 0, "");

                // add variable to symbol table
                e.addToCurrent(c);
            }

        | #("string" string_name:ID)
        {
                if (e.find(string_name.getText()) != null)
                {
                    String errorMessage = string_name +
                        " has already been declared!!!";
                    throw new CHADException(errorMessage);
                }

                c = new CHADType(string_name.getText(), CHADTYPE_STRING, 0, "");

                // add variable to symbol table
                e.addToCurrent(c);
            }

        | var_id:ID
        {
                // check if the variable ID is not part of the environment
                // including user-defined function names
                if ((e.find(var_id.getText()) == null) &&
                        (e2.find(var_id.getText()) == null))
                {
                    String errorMessage = var_id.getText() +
                        " has not been declared!!!";
                    throw new CHADException(errorMessage);
                }

                if (e.find(var_id.getText()) != null)
                {
                    c = e.find(var_id.getText());
                }
                else
                {
                    if (var_id.getText() != null)
                    {
                        c = e2.find(var_id.getText());
                    }
                }
            }

        | str_val:STRING
        {
                    c = new CHADType(CHADTYPE_STRING, 0, str_val.getText());
            }

        | int_val:INTEGER
        {
                c = new CHADType(CHADTYPE_INT, Integer.parseInt(int_val.getText()), null);
            }

        | #(fca:COLLECTION_FUNC_CALL a=expr (dt:DOT) (func:ID) (elist:EXPR_LIST)?)
        {
                // handle array,queue,and stack built-in functions
                if (dt != null)
                {
                    if (func.getText().equals("length"))
                    {
                        if (func.getNextSibling() != null)
                        {
                            String errorMessage = "length function does not take any argum
                            throw new CHADException(errorMessage);
                        }

                        if (a.getType() != CHADTYPE_ARRAY)
                        {
                            String errorMessage = a.getName() + " is not an array!!!";
                            throw new CHADException(errorMessage);
                        }

                        c = new CHADType(CHADTYPE_INT, a.getLength(), "");
                    }

                    if (func.getText().equals("push") |
                            func.getText().equals("enqueue"))
                    {
                        AST ex = func.getNextSibling();
                        AST arg = ex.getFirstChild();

                        b = expr(arg);

                        // verify that the argument type matches the storage type
                        if (b.getType() != a.getStorageType())
                        {
                            String errorMessage = a.getName() + " holds " +
                                a.storageTypeName() + " types!!!";
                            throw new CHADException(errorMessage);
                        }

                        if (func.getText().equals("push"))
                        {
                             if (a.stackData.isFull())
                            {
                                String errorMessage = a.getName() + " is full.";
                                throw new CHADException(errorMessage);
                            }

                            a.push(b);
                        }
                        else
                        {

                            if (a.queueData.isFull())
                            {
                                String errorMessage = a.getName() + " is full.";
                                throw new CHADException(errorMessage);
                            }

                            a.enqueue(b);
                        }
                    }

                    if (func.getText().equals("pop") |
                            func.getText().equals("dequeue"))
                    {
                        if (func.getNextSibling() != null)
                        {
                            String errorMessage = "pop function does not take any argument
                            throw new CHADException(errorMessage);
                        }

                        if (func.getText().equals("pop"))
                        {
                            if (a.stackData.isEmpty())
                            {
                                String errorMessage = a.getName() + " is empty.";
                                throw new CHADException(errorMessage);
                            }

                            c = a.pop();
                        }
                        else
                        {
                            if (a.queueData.isEmpty())
                            {
                                String errorMessage = a.getName() + " is empty.";
                                throw new CHADException(errorMessage);
                            }

                            c = a.dequeue();
                        }
                    }

                    if (func.getText().equals("swap") |
                            func.getText().equals("sortAZ") |
                            func.getText().equals("sortZA"))
                    {
                        if (elist.getNumberOfChildren() != 2)
                        {
```

```
                    String errorMessage = func.getText() +
                        " only takes two arguments!!!";
                    throw new CHADException(errorMessage);
                }

                AST first_arg = elist.getFirstChild();
                AST second_arg = first_arg.getNextSibling();

                b = expr(first_arg);
                d = expr(second_arg);

                if (func.getText().equals("swap"))
                {
                    a.swap(b, d);
                }
                else if (func.getText().equals("sortAZ"))
                {
                    a.sortAZ(b, d);
                }
                else
                {
                    a.sortZA(b, d);
                }
            }
        }
    }

| #(fca2:FUNC_CALL a=expr (elist2:EXPR_LIST)?)
{
        // used to store passed arguments
        Vector passedArgs = new Vector();

        // handle min, max,show, hide and user-defined functions
        if (a.getName().equals("show"))
        {
            if (elist2.getNumberOfChildren() != 1)
            {
                String errorMessage = "show(a) only takes oneargument!!!";
                throw new CHADException(errorMessage);
            }

            AST first_arg = elist2.getFirstChild();

            if (e.find(first_arg.getText()) == null)
            {
                String errorMessage = "variable passed to show(a) has not been dec
                throw new CHADException(errorMessage);
            }

            a = expr(first_arg);
            a.show();
            c = a;
        }

        else if (a.getName().equals("hide"))
        {
            if (elist2.getNumberOfChildren() != 1)
            {
                String errorMessage = "hide(a) only takes one argument!!!";
                throw new CHADException(errorMessage);
            }

            AST first_arg = elist2.getFirstChild();

            if (e.find(first_arg.getText()) == null)
            {
                String errorMessage = "variable passed to hide(a) has not been dec
                throw new CHADException(errorMessage);
            }

            a = expr(first_arg);
            a.hide();
            c = a;
        }

        else if (a.getName().equals("print"))
        {
            if (elist2.getNumberOfChildren() != 1)
            {
                String errorMessage = "print(a) only takes one argument!!!";
                throw new CHADException(errorMessage);
            }

            AST first_arg = elist2.getFirstChild();

            b = expr(first_arg);

            if (b.getType() != CHADTYPE_STRING)
            {
                String errorMessage = "print(a) can only print string types!!!";
                throw new CHADException(errorMessage);
            }
        }

        else if (a.getName().equals("min"))
        {
            if (elist2.getNumberOfChildren() != 2)
            {
                String errorMessage = "min(a,b) only takes two arguments!!!";
                throw new CHADException(errorMessage);
            }

            AST first_arg = elist2.getFirstChild();
            AST second_arg = first_arg.getNextSibling();

            a = expr(first_arg);
            b = expr(second_arg);

            c = CHADType.min(a, b);
        }

        else if (a.getName().equals("max"))
        {
            if (elist2.getNumberOfChildren() != 2)
            {
                String errorMessage = "max(a,b) only takes two arguments!!!";
                throw new CHADException(errorMessage);
            }
            AST first_arg = elist2.getFirstChild();
            AST second_arg = first_arg.getNextSibling();

            a = expr(first_arg);
            b = expr(second_arg);

            c = CHADType.max(a, b);
        }

        // handle user-defined functions
        else
        {
            // get function from environment's symbol table
            CHADType f = e.find(a.getName());

            if (f == null)
            {
                f = e2.find(a.getName());
            }

            // check if no arguments were given
            if (elist2 == null)
            {
                // check that # of arguments is the same as function declaration
                if ((f.getLength() - 1) != 0)
                {
                    String errorMessage = a.getName() +
                        " has wrong # of arguments!!!";
                    throw new CHADException(errorMessage);
                }

                c = new CHADType(f.getStorageType(), 0, "");
            }

            // loop through arguments
            else
            {
                int numOfArguments = elist2.getNumberOfChildren();
                int argNum = 0;
                ;

                // check that # of arguments is the same as function declaration
                if ((f == null) || (numOfArguments != (f.getLength() - 1)))
                {
                    String errorMessage = a.getName() +
                        " has wrong # of arguments!!!";
                    throw new CHADException(errorMessage);
                }

                AST arg = elist2.getFirstChild();

                while (numOfArguments > 0)
                {
                    // argument from actual function call
                    b = expr(arg);

                    // argument from function call definition
                    CHADType t = f.getArrayEntry(argNum);
```

```
                    // check that argument types match
                    // function declaration
                    if (b.getType() != t.getType())
                    {
                        String errorMessage = "Argument # " + (argNum + 1) +
                            " of function call " + a.getName() +
                            " does not correspond to its function declaration!!!";

                        throw new CHADException(errorMessage);
                    }


                    arg = arg.getNextSibling();
                    numOfArguments--;
                    argNum++;

                    // store arguments to be used within function
                    passedArgs.add(b);
                }
            }

            if (f.getStorageType() == CHADTYPE_ARRAY)
            {
                c = new CHADType(a.getName(), CHADTYPE_ARRAY,
                        f.return_storage_type, f.return_array_length);

                for (int i = 0; i < f.return_array_length; i++)
                {
                    c.insert(i, 0, "");
                }
            }
            else
            {
                c = new CHADType(f.getStorageType(), 0, "");
            }
        }
    }

| fn:"function" {}

| #(fd:FUNC_DECL c=expr) {}

| #(RETURN (c=expr)?) { }

| #(STATEMENT (stmt:. {  c = expr(#stmt);} )*) { } ;

/*********************************************************************
** Code for getting an array entry. checks for out of bounds errors
*********************************************************************/
array_entry returns [CHADType c] throws Overflow, CHADException
{
  CHADType a,b;
  c = null_data;
  int arrayLength = 0;
}
:(arr_id:ID)
{
        // check if array is in current environment, exit with error message if no
        if (e.find(arr_id.getText()) == null)
        {
            String errorMessage = "array " + arr_id.getText() +
                " has not been declared!!!";
            throw new CHADException(errorMessage);
        }

        int index = 0;

        // get the array from environment
        a = e.find(arr_id.getText());

        // get length of array
        arrayLength = a.getLength();

        // get AST node containing array index
        AST i = arr_id.getFirstChild();

        // evalute index
        CHADType ind = expr(i);

        // handle indexes that are integer literals
        if (ind.getType() == CHADTYPE_INT)
        {
            index = ind.getIntData(0);
        }

        else
        {
            String errorMessage = "array index must be integer!!";
            throw new CHADException(errorMessage);
        }

        // exit if array index out of bounds
        if ((index < 0) || (index > (arrayLength - 1)))
        {
            String errorMessage = "ARRAY " + arr_id.getText() +
                " INDEX OUT OF BOUNDS!!!";
            throw new CHADException(errorMessage);
        }

        // set name of array entry to name of array itself
        // so as to faciliate error messages
        if (!checkingFunctions)
        {
            (a.getArrayEntry(index)).setName(arr_id.getText());
        }

        // return the array entry
        if (checkingFunctions)
        {
            c = new CHADType(a.getStorageType(), 0, "");
        }
        else
        {
            c = a.getArrayEntry(index);
        }
    }
```

```
{    };
/**********************************************************************
** Code for evaluating array declaration list
**********************************************************************/
    array_list throws Overflow, CHADException
{

}
:

an:.
{
        CHADType d = expr(an);

        if ((previous_type != -1) && (previous_type != d.getType()))
        {
            String errorMessage = "All elements in array must be of same type!!";
            throw new CHADException(errorMessage);
        }

        ls.add(new CHADType(d.getType(), 0, ""));

        previous_type = d.getType();
    }

{ };

/**********************************************************************
** Code for walking tree and looking only for function declarations
**********************************************************************/
functions returns [CHADType c] throws Overflow, CHADException
{
  checkingFunctions = true;
  c = null_data;
  CHADType a,b;
    }

    :
    #(PLUS a=functions b=functions) { }
| #(PLUSEQ a=functions b=functions) { }

| #(MINUS a=functions b=functions) { }

| #(UMINUS a=functions) {  }

| #(MINUSEQ a=functions b=functions) { }

| #(MULT a=functions b=functions) { }

| #(MULTEQ a=functions b=functions) { }

| #(DIV a=functions b=functions) { }

| #(DIVEQ a=functions b=functions) { }

| #(MOD a=functions b=functions) { }

| #(MODEQ a=functions b=functions) { }
| #(INCREMENT a=functions) {  }

| #(DECREMENT a=functions) {  }

| #(EQ a=functions a=functions) {  }

| #(GT a=functions a=functions) {  }

| #(LT a=functions a=functions) {  }

| #(LE a=functions a=functions) {  }

| #(IFBLOCK a=functions){ }

| #("OR" a=functions ror:.) {  }

| #("AND" a=functions rand:.) {  }

| #(nn:IF_STMT a=functions itt:.){ }

| #(ELSEIF_STMT (a=functions)?) { ; }

| #(ELSE_STMT (et:.)?) {  }

| #(FOR_STMT yy:. yu:. kk:. km:.) {   }

| #(SQA_DECL a=functions)

| "queue" {  }

| "stack"{ }

| ("array"){ }

| #(ARRAY_ENTRY ae:.){ }

| #(ASGN a=functions b=functions) { }

| #(ARRAY_LIST b=functions){   }

| #("int" ID){  }

| #("string" ID){}

| ID { }

| STRING { }

| INTEGER{}

| #(COLLECTION_FUNC_CALL a=functions (DOT) (ID) (EXPR_LIST)?)

| #(FUNC_CALL a=functions (DOT)? (ID)? (EXPR_LIST)?)

| fn:"function"
{

}
```

```
    | #(gh:FUNC_DECL ft:.)
{
        AST fargs = ft.getFirstChild();
        int return_type = 0;
        int return_storage_type = 0;
        int return_array_length = 0;

        String function_name = "";

        // loop through args list
        if (fargs.getNumberOfChildren() != 0)
        {
            CHADType nf = null;
            AST list = fargs.getFirstChild();

            for (int i = 0; i < fargs.getNumberOfChildren(); i++)
            {
                // determine function return type
                if (i == 0)
                {
                    if (list.getText().equals("int"))
                    {
                        return_type = CHADTYPE_INT;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_INT, (fargs.getNumberOfChildren() / 2));
                    }
                    else if (list.getText().equals("string"))
                    {
                        return_type = CHADTYPE_STRING;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_STRING,
                                (fargs.getNumberOfChildren() / 2));
                    }

                    else if (list.getType() == #ARRAY_RETURN)
                    {
                        return_type = CHADTYPE_ARRAY;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_ARRAY,
                                (fargs.getNumberOfChildren() / 2));

                        AST ty = list.getFirstChild();

                        if (ty.getText().equals("int"))
                        {
                            nf.return_storage_type = CHADTYPE_INT;
                            return_storage_type = CHADTYPE_INT;
                        }
                        else
                        {
                            nf.return_storage_type = CHADTYPE_STRING;
                            return_storage_type = CHADTYPE_STRING;
                        }
                    }

                    else
                    {
                        return_type = CHADTYPE_VOID;
                        nf = new CHADType(list.getText(), CHADTYPE_FUNCTION,
                                CHADTYPE_VOID, (fargs.getNumberOfChildren() / 2));
                    }
                }

                // add function name to environment
                else if (i == 1)
                {
                    function_name = list.getText();
                    nf.setName(function_name);

        // check if function with same name has already
        // been declared
        if(e.find(function_name) != null
        || e2.find(function_name) != null)
        {
           String errorMessage = function_name + " has already been declared!!";
                throw new CHADException(errorMessage);
        }

                    // add function to symbol table
                    e.addToCurrent(nf);
                    e2.addToCurrent(nf);

                    // create a new scope and enter it
                    e.createNewScope();
                    e.enterScope();
                }

                // add argument types
                else
                {
                    int arg_type = 0;

                    // create new scope to store function arguments
                    // and function body declarations
                    if ((i % 2) == 0)
                    {
                        CHADType atype = null;

                        if (list.getText().equals("int"))
                        {
                            AST arg_name = list.getNextSibling();

                            String argument_name = arg_name.getText();

                            atype = new CHADType(argument_name, CHADTYPE_INT,
                                    0, "");

                            // check if argument with same name
                            // already used
                            if(e.find(argument_name) != null)
                            {
                              String errorMessage = argument_name + " has already
                  throw new CHADException(errorMessage);
                            }

                            e.addToCurrent(atype);

                            arg_type = CHADTYPE_INT;
                        }

                        else if (list.getText().equals("string"))
                        {
                            AST arg_name = list.getNextSibling();
                            String argument_name = arg_name.getText();

                            atype = new CHADType(argument_name,
                                    CHADTYPE_STRING, 0, "");

                            // check if argument with same name
                            // already used
                            if(e.find(argument_name) != null)
                            {
                              String errorMessage = argument_name + " has already
                  throw new CHADException(errorMessage);
                            }

                            e.addToCurrent(atype);

                            arg_type = CHADTYPE_STRING;
                        }

                        else if (list.getType() == #ARRAY_ARGUMENT)
                        {
                            AST ty = list.getFirstChild();
                            AST Asize = ty.getNextSibling();
                            AST arg_name = list.getNextSibling();

                            int arrayType = 0;

                            if (ty.getText().equals("int"))
                            {
                                arrayType = CHADTYPE_INT;
                            }
                            else
                            {
                                arrayType = CHADTYPE_STRING;
                            }

                            int arraySize = Integer.parseInt(Asize.getText());

                            String argument_name = arg_name.getText();

                            atype = new CHADType(argument_name, CHADTYPE_ARRAY,
                                    arrayType, arraySize);

                           // check if argument with same name
                           // already used
                            if(e.find(argument_name) != null)
                            {
                              String errorMessage = argument_name + " has already
                  throw new CHADException(errorMessage);
                            }

                            e.addToCurrent(atype);


                            arg_type = CHADTYPE_ARRAY;
                        }

                        nf.insert((i / 2) - 1, arg_type, "");
                    }
                }

                list = list.getNextSibling();
            }
        }

        // check body of function
        AST body = fargs;

        // check inside of function until
        // finding return statement
        while (body.getNextSibling().getType() != #RETURN)
        {
            body = body.getNextSibling();
            expr(body);
        }

        AST rstmt = body.getNextSibling();

        // check that the expression after "return" matches
        // the declared return type
        AST rs = rstmt;

        if (rs.getFirstChild() != null)
        {
            rs = rs.getFirstChild();

            CHADType rt = expr(rs);

            if (rt.getType() != return_type)
            {
                String errorMessage = function_name +
                    "'s return statement does not match its return type!!!";
                throw new CHADException(errorMessage);
            }

            if (return_type == CHADTYPE_ARRAY)
            {
                e2.find(function_name).return_array_length = rt.getLength();
            }
        }
        else
        {
            if (return_type != CHADTYPE_VOID)
            {
                String errorMessage = function_name +
                    "'s return statement does not match its return type!!!";
                throw new CHADException(errorMessage);
            }
        }

        // reset vector
        function_variables.removeAllElements();

        // exit scope
```

```
        e.exitScope();
    }

| #(STATEMENT (stmt:. {  c = functions(#stmt); } )*)

 {

 };
```

Underflow.java

```
/************************************************************************************
 ** Underflow.java
 ***********************************************************************************
 package DataStructures;

    /**
     * Exception class for access in empty containers
     * such as stacks, queues, and priority queues.
     * @author Mark Allen Weiss
     */
    public class Underflow extends Exception
    {
    }
```